XILINX®

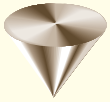
ONLINE

DEVELOPMENT SYSTEM

REFERENCE GUIDE VOLUME 2

TABLE OF CONTENTS

INDEX



GO TO OTHER BOOKS

XACT™ STEP

0401406

Contents

Chapter 6 PPR

The XNFMerge Program

This program is compatible with the following families.

- XC2000
- XC2000L
- XC3000
- XC3000A
- XC3000L
- XC3100
- XC3100A
- XC4000
- XC4000A
- XC4000H
- XC5200

XNFMerge combines Xilinx Netlist Format (XNF) files to form a single, flat (nonhierarchical) file with an .xff extension. An input XNF file that contains FILE= references or non-primitive symbols such as macro symbols is hierarchical and must be flattened by XNFMerge.

XNFMerge also propagates some location parameters down the design hierarchy, resolves relative location constraint parameters, and performs some preprocessing functions for XACT-Performance. Therefore, it is necessary to process designs through XNFMerge even if the design is already flattened.

Terms

This section defines several important terms that are used in this chapter.

Hierarchical File

An XNF file that contains references to other files. A hierarchical file contains non-primitive symbols and XBLOX symbols.

Flattened File

An XNF file that contains no references to other files. XNFMerge generates a flattened file (XFF). The only non-primitive symbols that should exist in the output file of XNFMerge are XBLOX symbols.

Top-Level File

An XNF file that references other XNF files. XNFMerge is always run on a top-level file.

Lower-Level File

An XNF file that is referenced by a top-level XNF file.

Signal Binding

The process of joining nets from a lower-level XNF file to pins in a top-level XNF file.

Resolving a Symbol

The process of replacing a symbol in a top-level XNF file that references a lower-level XNF file with the logic contained in the lower-level file.

Syntax

Use the following syntax to execute XNFMerge. XNFMerge accepts any output file extension, but the automatic design flow program (XDM/XMAKE) requires the default XFF extension.

xnfmerge *[options] input_name*[**.xnf**]
output_name[**.xff** |.*extension*]

Note: The output_name file is optional. If you do not specify this option, XNFMerge defaults to the input_name.xff.

XNFMerge reads an XNF file that describes the top-level of a hierarchical design. When XNFMerge finds a symbol that represents logic contained in another XNF file, it searches for and reads the referenced file. XNFMerge then replaces the symbol in the top-level file with the logic contained in the lower-level file and continues this process of flattening until no symbols that reference other XNF files remain. Then XNFMerge writes out the flattened design into a new XFF file.

A process called signal binding links the different levels of the hierarchical design. In this process, a signal contained in the top level of the hierarchy is bound to a corresponding signal contained in a lower level. Which signals are bound together is an important aspect of XNFMerge's operation.

When XNFMerge flattens the hierarchical structure of a design, it might modify the names of some signals and symbols in the design. This modification of symbol and signal names is necessary because logic contained in different parts of the hierarchy might have the same name. In the flattened version of the design, these names conflict. XNFMerge modifies the name of every signal and logic symbol (except those contained in the top level) to prevent name conflicts. Wherever possible, the new name consists of the original name prefixed with the instance name of the encompassing macro symbol.

File Name Extensions on Design Names

You can specify file name extensions. It is not necessary to include the XNF extension for the input file name. If the file name extension for the input file is omitted, XNFMerge reads in an XNF file.

Note: During execution, XNFMerge displays informational as well as error or warning messages on the screen.

Files

XNFMerge requires one top-level input file and generates two output files.

Input Files

input_name.xnf

The file that describes the top level of a hierarchical design.

Output Files

output_name.xff

The file to which XNFMerge writes the flattened design.

output_name.mrg

The merge report file that is created by XNFMerge. The default file name is the same as the name of the output design file, but with the MRG extension. You can use the -o option to change the name of the merge report file.

The merge report file contains the following information.

- The files that XNFMerge read
- The errors and warnings
- The signals bound together
- The symbols not expanded
- The hard macros translated to RPMs (relationally placed macros)
- The RLOC sets created
- The number of signals, primitive symbols, and unresolved symbols in the output design

Options

These options require that you type a hyphen in front of the option letter.

–a Abbreviate Messages to Report File

This option causes XNFMerge to report each unique file name only once in the MRG file and on the screen. Even if a file is read more than once for multiple instantiations in the design, its file name appears only once in the reports. This option is not recommended for general use, since it suppresses useful information about file flattening.

–d Directory Search for XNF Files

This option searches for XNF files in a specified directory. Relative and absolute directory path names are acceptable. Use this option for each directory when adding multiple directories to the list. XNFMerge searches the directories in the order of their appearance on the command line. XNFMerge always searches for XNF files in the current directory first unless the input file is specified to be in another directory. In this case, XNFMerge first searches all the lower-level files in the directory of the specified input file.

```
xnfmerge -d counters -d \xnflib input output
```

This example directs XNFMerge to search for XNF files first in the current directory, then in the counters subdirectory (relative to the current directory), and then in the \xnflib directory.

–f Do Not Record Hierarchy in Output File

By default, XNFMerge reserves hierarchy boundary information in the flattened output file. XNFMAP uses this data (when either the -a or -q option is used) to constrain the mapping of logic into blocks based on hierarchical boundaries. This is a re-implementation of the former map-then-merge design flow. Do not use the -f option if you intend to use XNFMAP with either the -a or -q options.

–i Ignore RLOC-Related Information

XNFMerge resolves all RPM (relationally placed macros) sets of design symbols that are associated by relative location (RLOC) attributes. XNFMerge also resolves all RLOC-related attributes (RLOC_ORIGIN, USE_RLOC, and so forth) in the design. This option directs XNFMerge to remove all RPM (RLOC) information from the design, with the exception of RLOC attributes on carry logic symbols. The XACT Development system requires carry logic symbols to have RLOC or LOC attributes.

–o Change the Merge Report Filename

This option changes the name of the merge report (MRG) file that XNFMerge creates. By default, the report file has the name *output_design*.mrg. If you specify a different extension, XNFMerge issues an error message and stops. If no extension is specified, XNFMerge adds the MRG extension. You can include a leading path specifier as part of the merge report file.

–p Specify the Target FPGA Device Type

This option tells XNFMerge which FPGA device type is being used. It overrides any FPGA device type information contained in the input design.

–q Allow Unresolved Hierarchy References

This option directs XNFMerge to treat those symbols, which reference XNF files that could not be found, as if they were primitive elements of an unknown function. It also directs XNFMerge to pass these symbols, unaltered, into the output design. If this option is not used, XNFMerge treats these symbols as errors and aborts before creating the XFF file. This option is useful if you only want a partial flattening of the hierarchy.

Determining Which Files are Symbol References

An XNF file contains “primitive” or “non-primitive” symbols. A primitive symbol is a simple, standard, predefined logic element such as an AND gate, an OR gate, a TBUF, or a flip-flop. Non-primitive symbols represent user logic modules or macros that are contained in separate XNF files.

XNFMerge passes each primitive symbol from the input design into the output design, and tries to replace each non-primitive symbol with the logic it represents by using a process called “resolving a symbol.” The information associated with the symbol itself determines exactly which XNF file XNFMerge uses to resolve a symbol.

Some symbols can have an explicit file name (by using the File= parameter). For each such symbol, XNFMerge searches for and uses an XNF file specified by the FILE= parameter.

If a symbol does not have an explicit file name, XNFMerge uses the symbol type to construct the name of the XNF file. It is important to distinguish between the type and the name of a symbol. Every AND symbol has the same type, but each symbol has a unique name. A design might contain several non-primitive symbols of the same type, each with a different name. Each symbol is replaced with the logic contained in the XNF file whose file name is the symbol type with an XNF file name extension. In other words, if XNFMerge finds a non-primitive symbol that has no File parameter, it tries to resolve the symbol by searching for a *symbol.*xnf file. If XNFMerge cannot find this file, it issues a warning message unless the -q option is used to suppress the warning message.

Searching for XNF Files

XNFMerge always searches for XNF files in the current directory first, unless the input file is specified in another directory by the FILE= parameter on the symbol. If it is unable to find a file, it looks in other directories, if -d is specified. If XNFMerge fails to find an XNF file either in the current directory or in the list of other directories, it issues an error message and does not create the XFF file.

XNFMerge looks in one additional directory for XNF files if the symbol includes a DEF=HM parameter to reference an old hard

macro file. In the XACT 5.0 release, the Xilinx-library of hard macro files are replaced with equivalent XNF files. These XNF files are installed in the /hmlib directory associated with the $XACT environment variable. Prior to XACT 5.0, the hard macro files were merged into the design by PPR. In XACT 5.0, the equivalent XNF files are merged into the design by XNFMerge. Therefore, if a symbol has a DEF=HM parameter, XNFMerge looks in the $XACT/hmlib directory if it has not found the XNF file in the current directory or the directories specified with the -d option.

Binding Signals Between Levels

Every symbol in an XNF file has pins to which signals can be attached. In a hierarchical design, symbols in a top-level XNF file reference lower-level XNF files. The symbol in the top-level file has pins to which signals are attached. These pins also reference signals in the lower-level file. Therefore, the pins represent the connectivity between the signals on the top-level file and the lower-level file. XNFMerge connects the signals attached to the symbol pins in the top-level file to the signals in the lower-level file. This process is called signal binding. XNFMerge uses two methods to determine which signals to bind between hierarchy levels: binding by signal name and binding by pin name.

Binding by Signal Name

If the pin name of the symbol in the top-level XNF file matches a signal name in the lower-level XNF file, the pin is considered connected to the signal.

Figure 1-1 illustrates binding by signal name. The top level of the design top.xnf references a lower-level design file mac1.xnf. The lower-level XNF file comes from another schematic. In this case, XNFMerge binds the pin names on the symbol in top.xnf to the corresponding signal names in the lower-level file mac1.xnf.

The pin name on the symbol matches the signal name in the lower-level XNF file. Signal IN1 is attached to pin A in the top-level schematic. A lower-level schematic contains a signal called A. XNFMerge binds pin A from the top-level file to signal A on the lower-level file, thus binding signal IN1 to signal A.

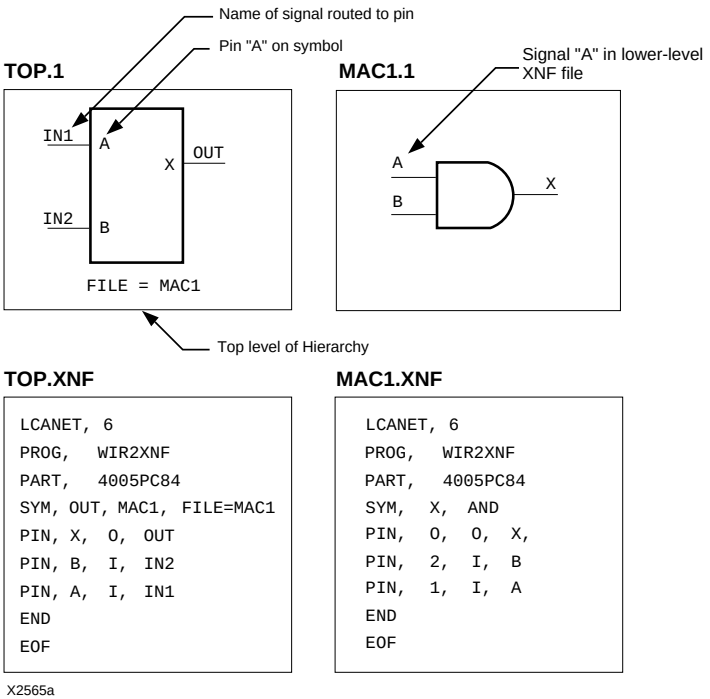


Figure 1-1 Binding Symbols by Signal Name

Binding one signal to another signal means that the two signals no longer have separate identities; they are the same signal. To reflect this, XNFMerge omits, from the output file, any mention of the signal contained in the hierarchy's lower level. All connections to the lower-level signal are replaced by connections to the top-level signal. The top-level signal inherits all information from the lower-level signal, except for the signal name. If two signals with conflicting attributes are bound, XNFMerge issues an error message.

Note: XNFMerge removes any S flag on a signal in a lower-level file. XNFMerge preserves all S flags that exist in the top-level file since they might be necessary if that file is to be merged into a higher-level file. The S flag marks a net that should not be removed in the logic trimming function of XNFPrep.

Renaming Signals and Symbols

Each symbol that references an XNF file represents a hidden level of logic. The signals and symbols contained in the top-level XNF file might have names that are the same as the names of signals or symbols contained in the lower level.

Since the output design must have no conflicting names, XNFMerge must modify some of the signal and symbol names in the input design before writing the output design. Logic contained in the top-level of the design in the XNF file is not renamed. Logic contained in lower levels is renamed according to the following procedure.

XNFMerge places the instance of the symbol referencing the logic and a "/" at the beginning of each signal and symbol name. For example, if the signal named "Reset" were contained in a part of the hierarchy named "Counter," then the new name of the signal would be "Counter/Reset." Figure 1-2 shows an example of merging levels of hierarchy.

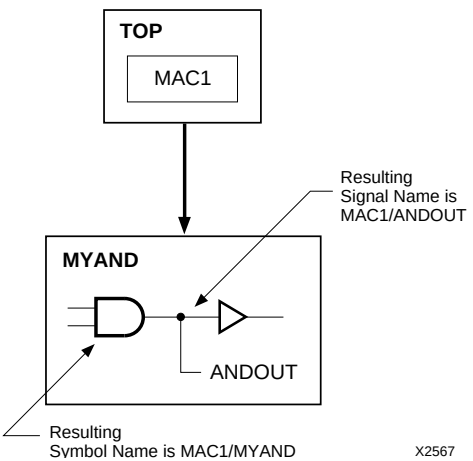


Figure 1-2 Merging Levels of Hierarchy

When merging levels of hierarchy, XNFMerge places the instance name of the symbol referencing the logic and a "/" at the beginning of each signal and symbol name. The names might differ slightly depending on the schematic editor used.

Propagating Location Parameters

XNFMerge propagates location constraints specified on non-primitive symbols to primitive symbols in the lower-level files. It examines the location constraint to determine its syntactical type. Then it propagates the location constraint to lower-level primitives that match the syntactical type. The syntactical types include location constraints for flip-flops, latches, IO symbols, TBUFs, and PULLUPs. For example, LOC constraints applicable to those symbols which are placed in CLBs are only passed to primitive symbols that can be contained within a CLB, such as DFF and CLBMAP. LOC constraints applicable to symbols which are placed in IOBs are only passed to primitive symbols that can be contained within an IOB such as IBUFs and INFF. Table 1-1 lists the LOC constraints that are propagated to primitives.

Table 1-1 LOC Constraints Propagated to Primitives

Location Constraints	Propagated To
CLBs	CLB, DFF, DLAT, CLBMAP, FMAP, HMAP, EQN, RAM, ROM, WORAND, CY4, CY4_MODE
IOBs	IOB, INFF, IBUFs, OBUF, OBUFT, INLAT, OUTFF, OUTFFT, WAND without DECODE parameter
TBUFs	TBUF
PULLUPs	PULLUP
Decoders	WAND with DECODE parameter

If another non-primitive symbol exists without a location constraint in the lower-level file, the location information is placed on the lower-level file's non-primitive symbol and is propagated further down the hierarchy. If another location constraint exists on the lower level non-primitive symbol, only the lower-level constraint is propagated to its sub-levels. If a primitive symbol exists with a Relationally Placed Macro (RPM), XNFMerge propagates LOC parameters only to the symbols in the RPM that do not have RLOC parameters.

To propagate the location constraints, XNFMerge looks at the first LOC statement and determines the category of primitive types to which the location constraint can be propagated. Therefore, it is not advisable to mix the different location constraint types on a non-primitive symbol.

Expressing Hierarchy in an XNF File

In the XNF file, a symbolic pin name is associated with a signal by a PIN=*pin_name* parameter on a SIG (signal) record for that signal.

An explicit file name is associated with a symbol by having a FILE=*file_name* parameter on the SYM (symbol) record for that symbol. If a file name extension is not specified, XNFMerge appends the XNF extension.

XNFMerge retains information about the design hierarchy in the XFF file. This information is useful if you want to map different XNF files before merging them. You can map different XNF files, before merging, for XC2000 and XC3000 designs in XNFMAP using the -a or -q option. This is a re-implementation of the former map-then-merge design flow. If you do not want XNFMerge to generate hierarchy information, use the -f option.

Relationally Placed Macros

Relationally Placed Macros (RPMs) are soft macros that contain logic symbols with relative location (RLOC) parameters. RLOC parameters define the spatial relationship between logic symbols. They do not define the absolute placement (unless RLOC_ORIGIN is used) of the logic in the part, but they do define the up/down/left/right relationships between two or more symbols. PPR maintains the spatial relationships defined by the RLOC parameters as it searches for the best, absolute placement of the logic on the part. Xilinx supplies an RPM macro library of XNF files to replace the prior hard macro library. You can also design your own RPMs using RLOC and RLOC-related parameters. See the *Libraries Guide* for details about using RLOC parameters.

If a design contains symbols with RLOC parameters, XNFMerge assigns each of these symbols to an RLOC set. The RLOC parameters within a given RLOC set define the spatial relationship between those

symbols; RLOC parameters in different RLOC sets have NO relationship to each other.

An RLOC set is defined either implicitly by the design hierarchy, or explicitly by user-specified U_SET and/or HU_SET parameters. All RLOC sets, whether defined by the you or by XNFMerge, are reported in the MRG report file that XNFMerge generates. Every symbol with an RLOC parameter is a member of exactly one RLOC set when XNFMerge is done.

XNFMerge also processes other RLOC-related parameters that affect the RLOC sets. The RLOC_ORIGIN= parameter locks down the symbols in a set to an absolute location on the part, while preserving the spatial relationships defined by the RLOC parameters. XNFMerge applies the RLOC_ORIGIN= parameter to the set and generates the appropriate absolute LOC= parameter for each symbol.

The RLOC_RANGE= parameter defines a rectangular row and column range on the part within which each symbol in the set must remain. XNFMerge ensures that any valid RLOC_RANGE= parameter for the set is added to each symbol in the set.

The USE_RLOC={TRUE,FALSE} parameter includes or excludes symbols from the set. If a USE_RLOC=FALSE parameter is found on any macro symbol in a set, XNFMerge removes the RLOC parameters from all the symbols for that set below the macro, with the exception of carry logic symbols. Carry logic symbols are required to always have either a LOC= or RLOC= parameter.

XNFMerge does additional checking on the valid use of RLOC and RLOC-related parameters and reports any errors in the MRG file.

For more information on using RPMs, refer to the *Libraries Guide*.

XACT-Performance Parameter Manipulation in XNFMerge

Design performance requirements are specified using XACT-Performance, as described in the "XACT-Performance Utility" chapter in this reference guide. XNFMerge prepares for XACT-Performance by pre-processing the TNM= and TSid= parameters.

If a macro symbol has a TNM=*spec* parameter, XNFMerge will propagate the parameter to the symbols that comprise the macro. The

spec field of the TNM=*spec* parameter has an optional type classification portion that specifies if the parameter should be propagated to 'FFS' (flipflops), 'PADS' (external pad signals), 'RAMS' (4K RAM symbols), or 'LATCHES' (IOB latch symbols). If the TNM= parameter has a type classification, XNFMerge propagates the parameter only to the specified type of symbols in the macro. If the TNM= parameter does not have a type classification, then XNFMerge propagates the parameter to the symbols in the macro only if the symbols are of one type (all flipflops, or all RAMS, or all IOB latches, or all pad signals). If there are mixed symbol types/signals in the macro and the TNM= spec field does not contain a type classification, then XNFMerge exits with an error.

XNFMerge also pushes forward any TNM= and TSid= parameters on signals to load pins on symbols. If the load pin is on a macro symbol, XNFMerge propagates the parameter down through the hierarchy until it finds a load pin on a non-macro symbol.

XNFMerge only pre-processes these XACT-Performance parameters. XNFPREP and PPR completes the preparation and implementation of XACT-Performance.

For a more information about using the XACT-Performance parameters, see the "XACT-Performance Utility" chapter in the *Development System Reference Guide*.

Warnings and Error Messages

Warnings and Recovery Techniques

`Warning 260. Parameter FAST on non-primitive symbol` *name* `ignored.`

`Parameter SLOW on non-primitive symbol` *name* `ignored.`

`Parameter INTERNAL on non-primitive symbol` *name* `ignored.`

`Parameter DOUBLE on non-primitive symbol` *name* `ignored.`

`Location parameters on non-primitive symbol` *name* `ignored.`

`User parameter` *parameter* `on non-primitive symbol` *symbol_name* `ignored.`

These are issued when invalid parameters for the specified part type are placed on a non-primitive symbol.

`Warning 282. Unable to open report file` *filename*`.`

Check for a disk-full condition, or verify the file permissions are set properly.

`Warning 285. Binding mismatch on pin` *pinname* `in symbol` *symbol_name* `file` *filename*`.`

The lower-level net name corresponding to the upper-level pin name on a symbol is missing. Likewise, if there is a bus pin on the upper-level symbol, and not all of the individual signals belonging to that bus are used at the lower-level, the above warning is issued. However, no action is needed if the warning accurately reflects the drawing.

`Warning 290. Location parameters on the upper-level signal` *name* `will take precedence over the lower-level signal` *name*`.`

This warning includes a message regarding the incompatibility between the LOC parameters on the upper and lower-level signals. Typically, this applies when the I/O buffer is on one level of the design hierarchy and the I/O pad is on another.

`Warning 291. No HIERG =` *id* `information attached to input` *sym name*`. Cannot determine exact place in hierarchy.`

If the input design includes hierarchy information but XNFMerge cannot determine the previous hierarchy assignment for a symbol, XNFMerge assigns the symbol to belong to the level of hierarchy assigned to the XNF file in which the symbol is located.

`Warning 292. Not acceptable syntax for LOC statement on non-primitive symbol` *sym name*`. Not propagating LOC statement to lower-level files.`

XNFMerge propagates the LOC= parameter on a macro symbol to the symbols incorporated within the macro. To do this, XNFMerge must know the type of symbol to which the LOC= parameter refers. Therefore, it examines the LOC= parameter value to determine

whether it should propagate the parameter to IO and edge symbols, CLB-type symbols, TBUFs, or PULLUPs. In this case, XNFMerge could not determine the proper type from the value (right-hand side) of the
LOC= parameter. Check the LOC= parameter for the proper specification. See the *Libraries Guide* for more information about using LOC= parameters.

```
Warning 295. The LOC parameter loc value on the
macro symbol sym name is not supported for XC7000
designs. The parameter will be ignored. RLOC
parameters are not supported on macro symbols for
XC7000 designs.
```

```
Warning 300. The HU_SET, hu_set name on symbol sym
name is a single element RLOC set. It will be
deleted.
```

An RLOC set must have two or more non-macro symbols with an RLOC= parameter, since RLOC= parameters define a relationship between symbols.

```
Warning 301. An RLOC_ORIGIN/RLOC_RANGE is found on
the primitive symbol sym name of an h_set. It will
be removed from the symbol.
```

RLOC_ORIGIN= and RLOC_RANGE= parameters must be placed on the top-level macro symbol for a set of RLOC= symbols that are defined through the design hierarchy ('H_SET'). These parameters can only be placed on a non-macro symbol if the symbol is part of a set of symbols that are tagged with a U_SET= or HU_SET= parameters.

```
Warning 311. TNM= tnm name parameter on macro
symbol sym name could not be propagated to any
lower level symbols. Check symbol type specified
on TNM with lower level symbol types.
```

If a macro symbol has a TNM=*spec* parameter, XNFMerge propagates the parameter to the symbols that comprise the macro. The *spec* field of the TNM=*spec* parameter has an optional type 'classification' portion that specifies if the parameter should be propagated to 'FFS' (flipflops), 'PADS' (external pad signals), 'RAMS' (4K RAM symbols), or 'LATCHES' (IOB latch symbols). If the TNM= parameter does not have a type classification, XNFMerge propagates the parameter to the

symbols in the macro only if the symbols are of one 'type' (all flipflops, all RAMS, all IOB latches, or all pad signals). It is an error if there are mixed symbol types/signals in the macro and the 'spec' does not contain a type classification.

> Warning 312. TSID parameter(s) found on output pin *pin name* of symbol *sym name*. TNM parameters are not allowed on symbol output pins. To tag all of the load pins on a signal, attach a TNM parameter to the signal itself.

> Warning 313. TNM parameter(s) found on signal *signal name* could not be propagated.

XNFMerge attempts to push forward any TNM= parameters found on signals to load pins on symbols. See the "XACT-Performance Utility" chapter in the *Development System Reference Guide* for more details.

> Warning 314. The TNM=*tnm value* parameter on macro symbol *mac name* will only be propagated to *type* symbols. XNFMERGE cannot examine XBLOX symbols *sym name* of type *type* to determine the symbol types below them.

> Warning 315. The TNM parameter was found on VCC and/or GND signal.

In some design entry interfaces, the same VCC and GND signal names are used throughout the design. If this is the case for your interface package, the TNM parameters listed below may be applied more broadly than was intended.

Signal Name = *sig name*, TNM value = *tnm value*

> Warning 318. The inversion on the *pin name* pin of the macro symbol *sym name* for the signal *sig name* will not be propagated to the underlying macro logic. Pins on macro symbols cannot be inverted.

> Warning 322. The U_SET/HU_SET *set name* on macro symbol sym name has no symbol with an RLOC parameter below it. It will be removed.

The presence of an HU_SET= or U_SET= parameter on a macro symbol requires that a set of symbols with RLOC= parameters exist

beneath the macro. In this case, XNFMerge could not find any symbols with RLOC= parameters to associate with the HU_SET= or U_SET= parameters.

```
Warning 323. The U_SET/HU_SET, set name on symbol
sym name has only one RLOC parameter on a
primitive symbol. The set will be removed.
```

The presence of an HU_SET= parameter on a non-macro symbol requires the presence of additional non-primitive symbols with the same HU_SET= parameter at the same level of the hierarchy. The presence of a U_SET= parameter on a non-macro symbol requires the presence of at least one other non-macro symbol with the same U_SET= parameter in the design. In this case, XNFMerge could find only one symbol with the unique RLOC= parameter to associate with the HU_SET= or U_SET= parameters.

```
Warning 324. Although the ignore_rlocs option was
specified, some RLOC parameters could not be
ignored because they are attached to CY4 symbols.
Every CY4 symbol in the design must have either a
RLOC constraint or a single-location LOC
constraint.
```

```
Warning 325. Although the USE_RLOC=FALSE
parameter was specified, the RLOC parameter on
the CY4 sym name could not be removed. Every CY4
symbol in the design must have either a RLOC
constraint or a single-location LOC constraint.
```

```
Warning 342. The FMAP symbol sym name has a TNM
parameter attached. This TNM parameter will be
ignored.
```

TNM parameters may be used only on flip-flops, I/O pads, RAM symbols, or on macros that contain those elements.

```
Warning 344. The FMAP symbol sym name has a TSid
parameter attached to its pin name pin.
```

Since a MAP symbol does not represent any actual logic, this TSid parameter will be ignored, and will not be traced forward to identify any other logic.

Error Messages and Recovery Techniques

`Error 1. at line` *number*`: Field too long.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 2. at line` *number*`: Unexpected LCANET record.`

Issued when an LCANET record is found on any line besides the first. This is an indication of an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 3. at line` *number*`: ENDMOD with no matching MODEL record.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 4. at line` *number*`: Symbol sub-record outside of symbol group.`

Issued when the XNF syntax is violated, and records that should be within other records, such as MODEL, ENDMOD, or PIN, are found outside the group.

This is an indication of an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 5. at line` *number*`: Illegal record inside symbol group.`

Issued when the XNF syntax is violated, and a record that is not allowed inside other symbol groups is found within the group. For example, a $\overline{\text{PROG}}$ (an active low signal; in the ASCII file, the tilde (~) indicates active low) is record inside a SYM group.

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 6. at line` *number*`: Premature EOF record in symbol group.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 7. at line` *number*`: Illegal record inside MODEL group.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 8. at line` *number*`: Premature EOF record in MODEL group.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 9. at line` *number*`: Premature End-of-file. No EOF record found.`

An EOF record must conclude every XNF file. This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 10. Unknown record type` *type*`.`

An invalid XNF record type was encountered in the file.

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 11. at line` *number*`: There are characters` *characters* `on the end of a record line, after all expected data has been received.`

`Error 12. at line` *number*`: Invalid LCA netlist file. Invalid or missing LCANET record.`

Every valid XNF file must start with the LCANET record that lists the XNF version of the file. Regenerate the XNF file.

`Error 13. at line` *number*`: Unsupported XNF netlist version` *number*`. Supported versions are: version` *number*`.`

Only XNF versions 1, 2, 4, and 5 are supported by this program. Regenerate XNF file with compatible software.

`Error 15. at line` *number*`: Valid part type must be specified before netlist symbols can be read.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.Error 16at line *number*: Invalid part record, missing parttype.

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 17. at line` *number*`: Missing name on SYM record.`

`At line` *number*`: Missing type on SYM record.`

These messages indicate an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`At line` *number*`: Unknown symbol` *type*`.`

`Error 18. At line` *number*`: Invalid PIN record. Missing name field.`

`At line` *number*`: Invalid PIN record. Missing or invalid direction field.`

`At line` *number*`: Invalid PIN record. Invalid direction field.`

`At line` *number*`: Invalid PIN record. Non-numeric delay field.`

`At line` *number*`: A bidirectional field (B) must only be a macro.`

These messages indicate an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 19. At line` *number*`: Pin name` *name* `used multiple times on symbol` *name*`.`

You can only have one instance of a particular pin name for each symbol.

`Error 20. At line` *number*`: Missing command on CFG record.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 21. At line` *number*`: At line` *number*`: CFG records allowed only in CLB and IOB symbols.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 22. At line` *number*`: Invalid SIG record. Missing signal name.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 23. At line` *number*`: Invalid EXT record. Invalid direction` *dir*`.`

EXT records must have a direction field of I, O, T, B, or U.

`At line` *number*`: Invalid EXT record. Missing signal name.`

Add the signal name to the EXT record.

`At line` *number*`: Invalid EXT record. Bad or missing direction field.`

This error message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 24. At line` *number*`: Invalid BUS record. Missing bus name.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 25. At line` *number*`: Invalid PULSE record. Missing pin name field.`

`At line` *number*`: Invalid PULSE record. Missing or invalid polarity field.`

`At line` *number*`: Invalid PULSE record. Invalid polarity field.`

`At line` *number*`: Invalid PULSE record. Invalid or missing minimum width field.`

These messages indicate an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

`Error 26.At line` *number*`: Invalid PWR record. Bad or missing polarity field.`

PWR record must have a 1 or a 0 in the polarity field.

`At line` *number*`: Invalid PWR record. Polarity is` *polarity*`. Must be 0 or 1.`

This error message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

```
Error 27. At line number: Invalid SETUP record.
Missing pin name field.

At line number: Invalid SETUP record. Missing
clock pin name field.

At line number: Invalid SETUP record. Missing or
invalid clock edge field.

At line number: Invalid SETUP record. Invalid
clock edge field.

At line number: Invalid SETUP record. Missing or
invalid setup time.

At line number: Invalid SETUP record. Missing or
invalid hold time.
```

These messages indicate an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

```
Error 28. At line number: Missing id on HIERG
record.

At line number: Missing type on HIERG record.

At line number: Missing name on HIERG record.

At line number: Missing filename on HIERG record.

At line number: Missing parent id on HIERG record.
```

These messages indicate an invalid or corrupt XNF file. Rerun the program that created this XNF file and try again.

```
Error 30. For general use of invalid parameters

At line number: Unknown MAP symbol type for SYM
name. The valid parameters are PUC, PLC, PLO, and
PUO.
```

This message is issued if a CLBMAP symbol has a MAP parameter value that is not PUC, PUO, PLC, or PLO. See the "XNFMAP" chapter in this reference guide for more information on MAP parameters. XNFMAP defaults to MAP=PUC if any A through E inputs pins are used. It defaults to MAP=PUO if no input pins are used.

`At line` *number*`: Invalid MAP symbol type HMAP symbol` *name*`. Will use default MAP type PUC.`

The only MAP parameter that is valid for an HMAP symbol is PUC.

`Type` *name* `has invalid parameter value` *parameter*`.`

This message is issued if a floating point value of the THI and TLO parameters have trailing characters that are not "ns" (for nanoseconds). Only the suffix "ns" may be added to a THI or TLO value.

`At line` *number*`: Unknown SYM record parameter` *parameter* `ignored.`

This message is issued if an unknown XNF symbol parameter is found. Check the parameter specification in the design file.

`At line` *number: invalid parameter* `parameter found on SYM` *name,* `type` *symbol_type.*

This message is issued if a parameter is assigned to a symbol that does not support that parameter, such as a FAST tag on a DFF symbol. Check the parameter for that symbol.

`At line` *number*`: Unknown PIN record parameter` *parameter.*

This message is issued if an invalid parameter is assigned to a PIN. Check the parameter for that symbol pin.

`At line` *number*`: invalid PIN record parameter` *parameter* `found on MAP symbol.`

This message is issued if an invalid parameter is assigned to a PIN statement for a CLBMAP. Only a P (pin-lock) parameter is allowed on CLBMAP symbol pins.

`At line` *number*`: Unknown SIG record parameter` *parameter*`.`

This message is issued if an invalid parameter is assigned to a SIG (signal) record. Check the parameter for that signal.

`At line` *number*`: Unknown EXT record parameter` *parameter.*

This message is issued if an invalid parameter is assigned to an EXT record. Check the parameter for that I/O pad.

`At line` *number*`: FILE parameter found on` *name* `SYM, type` *symbol_type*`.(Non-flattened design).`

This message is issued if a File parameter is found on a non-macro symbol. File parameters should be added only to unflattened macro symbols. This error message might occur if a reserved name, such as AND or OR, is used as the name of a File macro.

`At line` *number*`: Extra LOC parameter` *parameter* `found on signal` *name.*

This message is issued if more than one LOC parameter is found on a record. Multiple-block LOC parameters should be separated by semicolon (;) characters. If commas (,) are used to separate LOC parameters, this error is issued.

`At line` *number*`: Invalid MAP symbol type [PLO, PUO or PLC] for HMAP symbol` *symbol*`. PUC is the only valid MAP= parameter for an HMAP symbol.`

The only MAP parameter that is valid for an HMAP symbol is PUC.

`At line` *number*`:` *param* `is not complete on PIN` *pin name*`.`

This message is issued if there is an invalid parameter on the PIN record.

`At line` *number*`: SYM record parameter TNM not found on timespec symbols.`

`At line` *number*`: SYM record parameter` *parameter* `only allowed on timespec symbols.`

`Error 31.Invalid LOC parameter` *name* `on symbol-type` *name*`.`

An invalid location specification was found on the indicated symbol. Check the legal LOC constraints for the LCA family in use.

`Error 33. More than one` *parm* `is specified on symbol` *name*`. Only one is allowed per symbol`

This warning message refers to the fact that there is more than one RLOC, RLOC_ORIGIN or RLOC_RANGE on one symbol. XNFMerge only reads the first RLOC specified.

`Error 34. Invalid RLOC parameter` *parm* `on symbol-type` *name.*

`Error 35. Both TTL and CMOS parameters are specified on symbol` *name.* `Only one of these parameters is allowed per symbol.`

`Error 36. RLOC specified on symbol` *sym name* `of type` *type*`. RLOCs may not be used with decoders, clocks, combinatorial logic symbols or IO primitives.`

`Error 37. FAST/SLOW/MEDFAST/MEDSLOW and FAST/SLOW/MEDFAST/MEDSLOW parameters have been found on symbol/ext` *sym name/ext name*`.`

These parameters should be mutually exclusive.

`Error 38. TNM parameter TNM =` spec `on symbol` *sym name* `has illegal type. The only legal types are FFs, RAMS, PADS or LATCHES. The syntax should be TNM=`*name_value* `or TNM=`*type*`:`*name*`.`

`Error 40. At line` *number*`: Two different IOBs use the same external signal` *signalname*`.`

Two IOBs cannot use the same pad signal. Every pad signal must have a unique name. This error can occur if you have a pad signal connecting an input I/O symbol and an output I/O symbol, and the two symbols are in separate modules of the design. If the two modules are mapped independently using the XNFMAP -a or -q options (map-then-merge design processing), then the two symbols are made into two separate IOBs that both use the same pad signal. Although the intention might have been to make the pad signal a bidirectional signal, the separation of the two symbols into different modules causes this error. Keep all the I/O elements of a single IOB (pad signal, input and output symbols) in the same level of hierarchy of the design.

`Error 53. Out of memory. Needed` *number* `bytes.`

There is insufficient memory to complete the XNF file processing. Check the memory requirements for the LCA part type in use. This can be an indication of an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again.

Error 220. Can't open file *file name*.

Error 221. Filename *file name* called recursively.

Cannot accept recursive designs. A file name cannot have a macro that refers to itself or any subfiles that refer to itself.

Error 230. Aborting due to errors in the netlist file.

Includes specific message about error in netlist file.

Error 240. Input file cannot be overwritten. Specify an output file different from the input file.

Error 241. Unknown part type *type* specified.

Error 250. Error while writing XNF information to disk. Some information may be found in file *filename.*

Check for full disk condition.

Error 261. FAST, SLOW, MEDFAST and MEDSLOW parameters found on symbol *sym name*.

These parameters should be mutually exclusive.

Error 262. LOC/RLOC parameter *loc/rloc string* on CARRY MODE symbol *carry mode symbol name* does not correspond with LOC/RLOC parameter *loc/rloc string* on CARRY symbol *carry symbol name*.

A carry symbol and its associated carry-mode symbol cannot have different LOC/RLOC parameters.

Error 270. Invalid parttype *type*. Please re-run using the -P command line option.

Error 271. Conflicting parttypes *type* and *type*.

You cannot mix part families from different files that are merged into one design.

Error 281. Unable to open temporary work file *filename*.

Check disk full condition or file permissions.

`Error 283. Too many file names - check usage of program.`

There are too many file names on the input line.

`Error 284. Improper use of` *option* `option.`

`Error 296. The macro symbol` *sym name* `(type=`*type*`) has a LOC parameter that refers to location` *loc value*`.`

Since this macro contains symbols that will be mapped into both CLBs and IOBs, it is unclear if this location refers to a CLB location or to a half-edge of IOBs. To remove the ambiguity, move the LOC parameter further down the hierarchy, so that only one type of symbol is underneath it.

`Error 301. An RLOC_ORIGIN is found on the primitive symbol` *sym name* `of an h_set. The RLOC_ORIGIN will be removed from the symbol. An RLOC_RANGE is found on the primitive symbol` *sym name* `of an h_set. The RLOC_RANGE will be removed from the symbol.`

`Error 302. The upper-level symbol` *sym name* `in set` *set name* `has a different RLOC extension` *ext* `from the extension` *ext* `on lower-level symbol` *sym name*`.`

`Error 304. It is illegal to have both RLOC and RLOC_ORIGIN/RLOC_RANGE attributes on the same member of an H_SET. You must attach the RLOC_ORIGIN to a macro at the top level of the hierarchy of an H_SET.`

`Error 305. RLOC parameter` *rloc parameter* `found on XBLOX symbol` *sym name*`.`

RLOC parameters are not supported on XBLOX symbols.

`Error 306. Both RLOC parameter` *rloc parameter* `and LOC parameter` *loc parameter* `found on symbol` *sym name*`.`

`Error 307. HU_SET/U_SET/H_SET on symbol` *sym name* `has the same name as the HU_SET/U_SET/H_SET on symbol` *sym name*`.`

Change the U_SET/HU_SET name.

Error 308. Primitive symbol *sym name* in HU_SET/U_SET= *set name* has an RLOC_ORIGIN/RLOC_RANGE but no RLOC parameter.

Error 310. Macro *name* with TNM= *spec* contains different *type* symbols underneath. A TNM parameter without a *FFS*, *RAMS*, *LATCHES*, or *PADS* type classification cannot be used on a macro with multiple types of symbols.

If you want TNM parameters to apply to the different types of symbols in the macro, replace the unclassified TNM= *spec* parameter with a parameter that classifies the type of symbol for each specification. For example:
TNM=FFS:*name1*;RAMS:*name2*;LATCHES:*name3*;PADS:*name4*

Error 312. TNM= *tnm name* parameter found on output pin *pin name* of symbol *sym name*. This is illegal.

See the "XACT-Performance Utility" chapter in this reference guide for more information.

ERROR 319. XBLOX %s symbol named *sym name* has been classified as a *type name* TNM type. This symbol cannot have both *class name* and *class name* classifications in the TNM attribute.

ERROR 320. The U_SET *name* has both a RLOC_ORGIN on symbol *sym name* and a RLOC_RANGE on symbol *sym name*.

ERROR 321. The H_SET on symbol *sym name* has both an RLOC_ORIGIN and an RLOC_RANGE.

Error 326. The symbol *symbol name type=symbol type* with DEF=HM parameter is assigned a LOC value of *value*. This LOC value will require the macro to exceed the top boundary of the device.

Note that a LOC value on a macro with the DEF=HM parameter is interpreted as the location of the lower-left corner of that macro.

Error 327. The LOC value *loc value* on symbol *sym name* is not permitted in 4000 designs.

This error occurs when XNFMerge tries to adjust the RLOC in the lower-left hand corner of the RPM to the LOC value on the macro symbol.

`Error 328. The macro symbol` *symbol name type=symbol type* `with the DEF=HM parameter is assigned a LOC value of value.`

LOCs with extensions cannot be attached properly to the symbol in the lower-left hand corner of the RPM.

`ERROR 329. The hard macro symbol` *name*`, of type` *type*`, is assigned an LOC value of loc` *value*`.`

A hard macro with a LOC constraint that references more than a single block location (that is, multiple CLBs, range, or wildcard LOC constraint) cannot be automatically converted to an RPM. An RPM may be constrained to a particular area of the device with an RLOC_RANGE or RLOC_ORIGIN parameter. The DEF=HM and LOC parameter must be removed from the hard macro symbol in order for it to be treated as a standard RPM.

`ERROR 340. The macro symbol` *name*`,of type` *type*`, with the 'DEF=HM' parameter is assigned an LOC value of` *value*`.`

LOC parameters that prohibit locations cannot be attached properly to the symbol in the lower left hand corner of the macro. An RPM may be constrained to a particular area of the device with an RLOC_RANGE parameter. The DEF=HM and LOC parameter must be removed from the hard macro symbol in order for it to be treated as a standard RPM.

`Error 341. H_SET parameter` *H_SET param* `found on symbol sym` *name* `of type` *type* `Use HU_SET or U_SET parameter instead.`

`Error 343. Invalid use of TSid parameter on MAP symbol` *sym name* `of type` *type*`. TSid parameters cannot be used on this ty1`

XNFPrep

This program is compatible with the following families.

- XC2000
- XC2000L
- XC3000
- XC3000A
- XC3000L
- XC3100
- XC3100A
- XC4000
- XC4000A
- XC4000H
- XC7200
- XC7300

The XNFPrep program performs a design rule check (DRC) on a flattened XNF file and removes unused and redundant logic so that processing will be accurate in the rest of the design flow. It also checks the syntax of the XACT-Performance parameters found in the design and prepares delay information for PPR path analysis. In performing all three of these functions, XNFPrep issues error and warning messages concerning the problems that it finds.

Design Flow

XNFPrep is part of three design flows: one involves X-BLOX for XC4000, XC3000A/L, and XC3100A designs; one involves these same families without X-BLOX; and the third pertains to XC2000, XC2000L, XC3000, and XC3100 designs. These flows are shown in the following two figures. XMake automatically executes the flow shown in Figure 2-1.

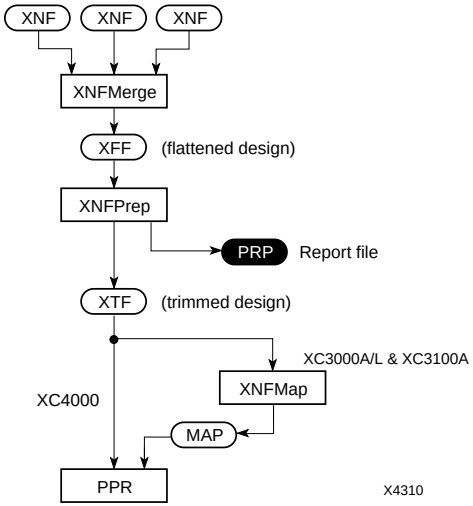


Figure 2-1 XNFPrep Design Flow Without X-BLOX for XC4000, XC3000A/L, and XC3100A Designs

XMake executes the flow shown in Figure 2-2 when you select the -b option in XMake.

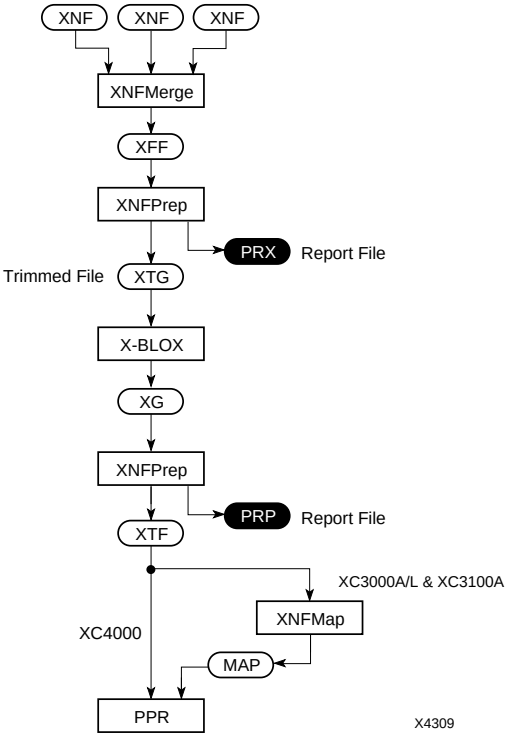


Figure 2-2 XNFPrep Design Flow with X-BLOX for XC4000, XC3000A/L, and XC3100A Designs

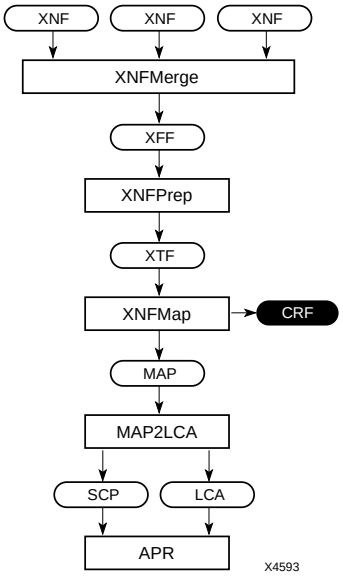


Figure 2-3 Design Flow for XC2000, XC2000L, XC3000, and XC3100 Designs

Files

Input Files

When X-BLOX is not involved in the design flow, the input to XNFPrep is a flattened XNF file with an .xff extension. When X-BLOX is involved in the design flow, the input file is a flattened XNF file with an .xff extension before X-BLOX processing and an XNF file with an .xg extension after X-BLOX processing. You must specify this .xg extension; otherwise, XNFPrep assumes that the input has an .xff extension.

Output Files

XNFPrep generates a report file that lists warnings, errors, logic trimmed, and clock fan-out. It also includes an XACT-Performance specification summary. The default name of the report file is

input_design.prp for designs without X-BLOX symbols and *input_design*.prx for designs with X-BLOX symbols. You can specify a different report file name with the Report option. The report file is created in the directory in which XNFPrep is run unless a different path is specified with the Report option.

If XNFPrep finds no errors, it produces a trimmed output XNF file with a default name of *input_design*.xtf for designs without X-BLOX symbols. For designs containing X-BLOX symbols, it produces a trimmed output XNF file with a default name of *input_design*.xtg the first time that the input file is submitted to XNFPrep. It also produces a file called *input_design*.xtf during the second submission. You can specify a different name for either of the trimmed output XNF files with the Outfile option.

The trimmed logic is shown in indented style in the report file; the indentations reflect the causal relationship of the trimming. For example, if ground on the input pin of an AND gate causes the gate itself to be removed, the record would look similar to the following example.

```
Due to GND signal on pin '3' of AND symbol 'AND2':
  Signal 'AND2_0' merged into signal 'GND'.
    Disabled AND symbol 'AND2' removed.
```

XNFPrep also outputs a log file, xnfprep.log, which contains all information output to the screen during XNFPrep execution. You can specify a log file name other than xnfprep.log by using the Logfile option described in the "XNFPrep Options" section.

How to Use XNFPrep

This section describes how to perform XNFPrep's functions. More information on the options that implement these functions is given in the "XNFPrep Options" section of this chapter.

XNFPrep commands are case-insensitive, even though they are shown with initial capital letters in the descriptive text.

Invoking XNFPrep

You can access XNFPrep through the command line or through XDM.

From the Command Line

To invoke XNFPrep on the command line, use the following syntax.

xnfprep *designname* **[outfile=***filename***]** {*options*}

where *designname* is the name of the flattened XNF file. You do not have to attach any extension to *designname* when the input file has an .xff extension; by default, XNFPrep searches for an XNF file with this extension. When the input file is an XNF file after X–BLOX processing, you must specify its .xg extension so that XNFPrep does not assume it has an .xff extension.

You can specify *filename* to be the name of the trimmed output XNF file. If you do not specify this file name, XNFPrep uses the name of the input design as the default.

Options can be any of the following, which are described in detail in the "XNFPrep Options" section of this chapter.

Cstfile=*filename*
-Helpall
Ignore_rlocs={True | False}
Ignore_timespec=*value*
Ignore_xnf_locs=*value*
Logfile=*filename*
Paramfile=*filename*
Parttype=*parttype*
Report=*reportname*
Savesig={True | False}

From XDM

In XDM, you can invoke XNFPrep by following these steps.

1. Type **xdm** on the command line.
2. Select **XNFPREP** from the Translate menu.
3. Select the name of the input file from the list that appears, or type in the name at the prompt in the upper left corner of the screen.

4. Select the name of the output file from the list that appears, or type in the name at the prompt in the upper left corner of the screen. You can also select **DONE** to allow XNFPrep to use the input file as the default output name.
5. Select the options that you wish from the list of XNFPrep options.
6. Click on **DONE** to run XNFPrep.

Running XNFPrep in XMake

To execute any of the design flows shown in Figure 2-1, 2-2, or 2-3, follow the instructions given in the "XMake" chapter of this reference guide. XMake automatically detects the presence of X-BLOX symbols and chooses the appropriate flow.

Obtaining Help

You can obtain help in two ways when using XNFPrep. You can type **xnfprep** ↵ or **xnfprep -helpall**. The Xnfprep ↵ command brings up a description of the –Helpall, Cstfile, Outfile, Parttype, Report, and Savesig options. –Helpall brings up a description of all the options available in XNFPrep and their settings. Any other options entered at the same time as -Helpall are ignored.

Trimming Signals

You can direct XNFPrep to retain or trim signals that are either sourceless or loadless. On the command line, type the following syntax.

xnfprep *designname* **savesig={true|false}**

When it is set to True, XNFPrep retains sourceless or loadless signals and adds the Savesig "S" parameter to them. When it is set to False, it trims sourceless or loadless signals. The default is False. This option is useful when you work with partial designs.

Although you can place and route a design that has loadless and sourceless nets, when you are trying to create a design you can download into a device, you should run XNFPrep with savesig=false.

Ignoring Parameters

XNFPrep can ignore LOC, RLOC, and/or XACT-Performance parameters in the design.

To ignore LOC parameters in specified places, except on carry symbols, type the following.

xnfprep *designname* **ignore_xnf_locs=***value*

where *value* can be All, Interior, Io, or None. These settings are described in the "XNFPrep Options" section of this chapter under the Ignore_xnf_locs option. Similarly, XNFPrep can ignore RLOC parameters, except on carry symbols, but they are processed as a whole; you cannot specify that they be removed only in certain places in the design. Use the following syntax.

xnfprep *designname* **ignore_rlocs={true|false}**

True removes RLOC parameters; False leaves them in the design. The default is False.

You can ignore XACT-Performance specifications in the design by entering the following.

xnfprep *designname* **ignore_timespec=***value*

Value specifies where the data is to be removed; it can be All, Design, Cst, or None. These settings are described in the "XNFPrep Options" section of this chapter under the Ignore_timespec option.

Submitting a Constraints File

To submit a constraints file to XNFPrep, enter this option.

xnfprep *designname* **cstfile=***filename*

You must use the Cstfile option if the CST file contains TIMESPEC or TIMEGRP statements, and if the CST file has a name other than *designname*.cst. By default, NXFPrep reads the *designname*.cst file.

Naming Files

You can specify the names of the output report file and the trimmed XNF file if you do not wish to use the defaults.

The default name of the output report file is *input_design*.prp for

designs without X-BLOX symbols or *input_design*.prx for designs with X-BLOX symbols. Use the Report option to specify an alternate name for the report file.

xnfprep *designname* **report=***filename*

To specify an alternate name for the trimmed output XNF file, enter the Outfile option.

xnfprep *designname* **outfile=***filename*

Specifying Part Type

To specify the part type of the design, type the following syntax.

xnfprep *designname* **parttype=***parttype*

This option overrides the part type specified in the PART record of the XFF file.

Examples

Following are three examples showing how to use XNFPrep with certain criteria.

The first example assumes that you want to prepare the "bigchip" design for submission to PPR for placement and routing. The trimmed output file is to be named "top." The report file is to have the same name as the input file but should be placed in a different directory: /home/test/results. Enter the following syntax on a single command line.

```
xnfprep bigchip outfile=top
report=/home/test/results/bigchip
```

In a second example, the "final" design is also to be submitted to PPR. The trimmed output file should be placed in a different directory: /home/test. LOC parameters on the I/O symbols should be ignored but all others used. Lastly, the part type is 4005PC84 instead of what is listed in the design. Enter the following syntax on a single command line.

```
xnfprep final outfile=/home/test/final
parttype=4005pc84 ignore_xnf_locs=io
```

In the third example, suppose that you wanted to submit the "best" design to PPR for placement and routing. You want all XACT-Performance parameters found in the design to be ignored, and you want to use a constraints file, "bestcst," found in /home/test/fix. To perform these functions, enter the following syntax on a single command line.

```
xnfprep best ignore_timespec=design
cstfile=/home/test/fix/bestcst
```

Options

The options available with the Xnfprep command are listed here in alphabetical order.

Cstfile

The Cstfile option specifies the name of the constraints file to use. You must use this option if the CST file contains TIMESPEC or TIMEGRP statements, and if the CST file has a name other than *designname*.cst.

Command line syntax:	`cstfile=`*filename*
XDM command:	`Translate` → `XNFPREP` → `cstfile=`*filename*
Values:	*filename*
Default value:	None
Applicable family:	XC3000A, XC3000L, XC3100A, XC4000

This option specifies the name and path of the constraints file to submit to XNFPrep.

-Helpall

The -Helpall option brings up a description of all the XNFPrep options and their settings. Do not enter other options at the same time that you enter the -Helpall option.

Ignore_xnf_locs

The Ignore_xnf_locs option ignores the LOC parameters in the input design file for specific types of logic.

Command line syntax:	`ignore_xnf_locs=`*type*
XDM command:	`Translate` → `XNFPREP` → `ignore_xnf_locs=`*type*
Values:	`all`, `io`, `interior`, `none`
Default value:	`none`
Applicable family:	All families

Type specifies the logic on which to ignore LOC parameters.

- All ignores LOC parameters on all logic.
- Io ignores LOC parameters on I/O symbols.
- Interior ignores LOC parameters on internal logic, that is, on everything except I/O symbols.
- None does not ignore any LOC parameters.

Note: XC4000 carry logic (CY4) symbols must have either RLOC or LOC constraints. If a design contains CY4 symbols with LOC constraints, the Ignore_xnf_locs option will not ignore these LOC constraints. Any register-based LOCs or RLOCs not associated with carry logic structures will get ignored.

Ignore_rlocs

The Ignore_rlocs option ignores all RLOC parameters in the design file.

Command line syntax:	`ignore_rlocs={true|false}`
XDM command:	`Translate` → `XNFPREP` → `-ignore_rlocs`
Values:	`true`, `false`
Default value:	`false`
Applicable family:	XC4000

When this option is set to True, XNFPrep ignores RLOC parameters; when it is set to False, it does not.

Note: XC4000 carry logic (CY4) symbols must have either RLOC or LOC constraints. If a design contains CY4 symbols with RLOC constraints, the Ignore_rlocs option will not ignore these RLOC constraints. Any register-based LOCs or RLOCs not associated with carry logic structures will get ignored

Ignore_timespec

This option ignores timing requirements specified in the schematic.

Command line syntax:	`ignore_timespec=`*value*
XDM command:	`Translate` → `XNFPREP` → `ignore_timespec=`*value*
Values:	`all`, `design`, `cst`, `none`
Default value:	`none`
Applicable family:	XC3000A, XC3000L, XC3100A, XC4000

Value can be one of the following.

- All ignores XACT-Performance specifications in the input design and in the constraints file.
- Design ignores XACT-Performance specifications in the input design.
- Cst ignores XACT-Performance specifications in the constraints file.
- None does not ignore any XACT-Performance specifications.

The default is None.

For more information on XACT-Performance parameter checking, refer to the "XACT-Performance Utility" chapter in this reference guide.

Logfile

The Logfile option specifies an alternate name for the xnfprep.log file.

Command line syntax:	`logfile=`*filename*
XDM command:	None
Values:	*filename*
Default value:	xnfprep.log
Applicable family:	All families

Use the Logfile option to assign a different name to the xnfprep.log file. The .log extension is appended if you do not specify an extension with the new file name. If the Logfile option is not used, the screen output is written to the xnfprep.log file, overwriting any previous versions of this log file.

Outfile

This option specifies the path and name of the trimmed output XNF file if you do not wish to use the default file name of *input_design*.xtf for designs without X-BLOX symbols, or *input_design*.xtg for designs containing X-BLOX symbols. You do not have to specify the .xtf or .xtg extension.

Command line syntax:	`outfile=`*filename*
XDM command:	None; output file name is specified while invoking XNFPrep.
Values:	*filename*
Default value:	XTF or XTG file name
Applicable family:	All families

Paramfile

Using the Paramfile option, you can specify XNFPrep options in a separate file, called a parameter file, instead of from the operating system prompt. A parameter file is a text file containing a list of desired options and their respective values, as in the following example.

```
parttype=4005pg156
```

```
cstfile=juke.cst
ignore_xnf_locs=io
```

Command line syntax: `paramfile=`*filename*

XDM command: `Translate` → `XNFPREP` → `paramfile=`*file*

Values: *filename*

Default value: None

Applicable family: XC3000A, XC3000L, XC3100A, XC4000

Use the name of the parameter file as the value for the Paramfile option.

You can specify additional options on the command line whenever a parameter file is used; these override similar options specified in the parameter file.

Parttype

This option specifies a part type for the design. It overrides the part type specified in the XFF file. A part type must be specified either in the .xff file or with this option. *Parttype* must be a valid part type, such as 4005PG156-5 or 3042APG132-6.

Command line syntax: `parttype=`*parttype*

XDM command: None; XDM invokes XNFPrep with the correct part type.

Values: Part type name

Default value: None

Applicable family: All families

Report

This option specifies the name of the report file if you do not wish to use the default file name of *input_design*.prp for designs without X-BLOX symbols or *input_design*.prx for designs with X-BLOX symbols. You do not have to specify the .prp or .prx extension.

Command line syntax: `report=`*filename*

XDM command:	`Translate` → `XNFPREP` → `report=`*filename*
Values:	*filename*
Default value:	PRP or PRX file name
Applicable family:	All families

Savesig

This option directs XNFPrep to retain or trim signals that are either sourceless or loadless. When it is set to True, XNFPrep retains sourceless or loadless signals and adds the Savesig "S" parameter to them. When it is set to False, it trims sourceless or loadless signals. The default is False. This option is useful when you work with partial designs.

Command line syntax:	`savesig={true|false}`
XDM command:	`Translate` → `XNFPREP` → `-savesig`
Values:	`true`, `false`
Default value:	`false`
Applicable family:	All families

Note: If your design contains carry symbols, every carry chain must be completely defined. Sourceless or loadless connections to CY4 symbols cannot be preserved, either with the Savesig option or with individual "S" parameters.

Split_report

This option is used by the Flow Manager to generate multiple reports.

Command line syntax:	`split_report ={true|false}`
XDM command:	None
Values:	true, false
Default value:	false
Applicable family:	All families

The XNFMAP Program

This program is compatible with the families indicated.

- XC2000
- XC2000L
- XC3000
- XC3000A
- XC3000L
- XC3100
- XC3100A

XNFMAP maps the logic defined by an XTF file (Xilinx netlist file trimmed flat) into FPGA blocks — CLBs, IOBs, TBUFs — that can be placed and routed by APR or PPR. The XNFPrep program converts your XNF into an XTF file. XNFMAP first maps logic associated with CLBMAP symbols in the design file or in the partitioning guide file (PGF). The program then efficiently maps the remaining logic into CLB and IOB blocks.

XNFMAP accepts the XTF file generated by the design rule check and logic trimming program, XNFPrep. The input file for XNFMAP must not contain design errors or untrimmed logic.

The output of XNFMAP is a MAP file that describes logic partitioning into FPGA blocks. This file is formatted differently depending upon the FPGA family type and the next program in the design flow. MAP files for XC2000, XC2000L, XC3000, and XC3100 designs are formatted for the MAP2LCA and APR programs. MAP files for XC3000A, XC3000L, and XC3100A designs are formatted for the PPR program.

XNFMAP also produces a design-to-LCA cross-reference report file (CRF) and a PGF. The CRF contains the cross-reference between original logic elements and the resulting FPGA blocks; a report on the successful use of CLBMAP guide symbols; a listing of symbols ordered in registers; and a summary of the CLB and IOB blocks used in implementation. You can use the PGF on subsequent design iterations to guide a modified design with the logic mapping from a previous iteration. Guided mapping is necessary for success with guided placement and routing in APR and PPR, and is an important component in the incremental and iterative design flows.

XNFMAP can *map-then-merge* the logic using the design hierarchy information stored in the design file by the file merging program, XNFMerge. The map-then-merge options independently map the logic at specified levels of the design hierarchy before merging all the levels to generate the output file. These options prevent the mapping of logic from different levels of the design hierarchy together into CLBs. Because XNFMAP can map-then-merge a single input file, it does not accept as input any files it previously generated, such as MAP files.

XNFMAP orders registers by pairing register flip-flops into CLBs according to the alphabetic order of their output signal names.

Note: The design rules checker program, XNFPrep, must be run on XFF files before executing XNFMAP. See the "XNFPrep" chapter in this reference guide for more information.

Syntax

Use the syntax shown below to map logic defined by an XTF file into the elements of a MAP file.

xnfmap [*options*] *design*.**xtf** *output*.**map**

Using XACT Design Manager (XDM)

You can invoke XNFMAP through XDM, a menu-driven interface for executing all FPGA development operations. Refer to the "XDM" chapter in the *Development System Reference Guide* for a complete description.

Once you access XDM, select **Translate→XNFMAP** from the main menu. A cascading menu appears that lists all available XNFMAP commands. Refer to the "Options" section below for a detailed description of each menu selection.

Files

This section describes the XNFMAP input and output files.

Input Files

XNFMAP uses the following files as inputs.

- *design*.xtf — The XTF netlist file that describes the design.
- *design*.pgf — An optional partitioning guide file (PGF) that describes a previous run of XNFMAP. The PGF file, generated from a previous run of XNFMAP, is used when the -k or -h option is specified. See the "Using the Partitioning Guide File" section in this chapter for more information.

Output Files

XNFMAP creates the following output files.

- *design*.map — The MAP file that contains the logic partitioning information, which is the input file for the MAP2LCA or PPR program.
- *design*.crf — The logic-to-LCA cross-reference report file that contains cross-references between the original logic elements and the resulting FPGA design elements. The CRF also contains information about the effects of CLBMAPs and guide files, and a design summary.
- *design*.pgf — The partitioning guide file that describes how XNFMAP partitioned the logic for the specified design. The PGF can be used to guide the partitioning of the specified design the next time you run XNFMAP. See the "Using the Partitioning Guide File" section for more information.
- *design*.pbk — The partitioning guide file backup. This file is a copy of the design.pgf file that exists in the current directory before

running XNFMAP. See the "Using the Partitioning Guide File" section for more information.

Options

Command-line options enable you to change the way XNFMAP operates. The following sections discuss each option in detail.

–a Respect Hierarchy Boundaries

The -a option maps only logic elements from the same level of hierarchy in a single CLB. A level of hierarchy is defined as the logic contained within a single user-specified macro symbol. Use this option when you want to preserve all original design hierarchy during implementation. Without this option, XNFMAP might place logic elements from different levels of the hierarchy in the same CLB.

If the file contains no hierarchy information, this option has no effect on the mapping process.

Note: Use of this option can lead to less efficient use of the FPGA resources since it usually restricts XNFMAP's mapping functions. Do not use this option if you used the XNFMerge option that suppresses storing the hierarchy information in the design file.

–c Ease the Requirements for Combining Logic

The -c option relaxes the requirement for a high degree of signal-sharing between logic elements in a CLB, resulting in more densely packed CLBs.

Note: Although this option makes a design denser, it can also affect place and route performance, resulting in higher delays and more unrouted nets. Use this option if you are willing to trade performance for density.

–e Estimate LCA Resources

The -e option allows a preliminary estimate of the number of CLBs and IOBs required to implement a design. XNFMAP accepts a file not previously processed by XNFPrep. Since XNFPrep did not process the output file, do not use it for further design implementation.

–f Force Dense Partitioning of Logic

The -f option partitions logic more densely. Normally, XNFMAP partitions logic to maximize signal sharing within CLBs and to minimize routing congestion. Using the -f option is equivalent to using the -c, -s, and -i options together.

Note: Although this option makes a design denser, it can also affect place and route performance, resulting in higher delays and more unrouted nets. Use this option if you are willing to trade performance for density.

–g Limit the Number of Gates Combined into One CLB

The -g option limits the number of gates combined into one CLB function generator. This option requires a number as an argument, as follows.

xnfmap -g*number design***.xtf** *output***.map**

If you specify the -g option without a number, XNFMAP uses the default of 20 gates. If you do not specify the -g option, XNFMAP can combine an unlimited number of gates into one function generator. However, if XNFMAP builds an equation that overflows the equation buffer, it issues a message suggesting that you use this option.

–h Read Named Guide File and Keep Same Logic Mapping

Use the -h option as follows.

xnfmap *design* **-h** *filename***.pgf**

When you specify this option, XNFMAP reads the specified PGF file and uses the information in this file to guide the mapping of a new design file. It is necessary to re-create the same logic mapping to guide the placement and routing of your design with an LCA guide file in APR or PPR.

This option differs from the -k option because you can specify a guide file name that is different from your design's file name. The guide file specified with the -h option can exist in a different directory from your design file.

Warning: XNFMAP always writes the output guide file to the same directory as your design file and uses the same file name as your design file with a .pgf extension. XNFMAP does not use the file name specified with the -h option when writing the output guide file.

–i Permit the Use of Direct Flip-Flop Input Pins

The -i option uses the DI (direct flip-flop input) pins on CLBs for the XC3000, XC3000A, XC3000L, XC3100, and XC3100A families. Using the DI pin reduces the setup time required for the CLB flip-flop, but also introduces a hold time requirement.

Note: Although this option makes a design denser, it can also affect place and route performance, resulting in higher delays and more unrouted nets. Use this option if you are willing to trade performance for density.

–j Ignore IO Location Constraints

The -j option ignores and discards any location (LOC) attributes on IO logic. This option does not affect constraints on TBUF or CLB logic.

–k Guide the Partitioning of a Previous Design Iteration

The -k option guides the partitioning of a previous design iteration using the design.pgf file. See the “Using the Partitioning Guide File” section for more information.

–m Ignore CLBMAP Symbols in the Design File

The -m option ignores any CLBMAP guide symbols in the input design file. Because the CLBMAP symbols in the design file override the information in the partitioning guide file (PGF), this option can be used to give precedence to the guide file information. Use this option if you suspect that over-constraining the design through the misuse of CLBMAP symbols caused poor placement and routing.

–n Ignore Interior-Array Location Constraints

The -n option ignores and discards any location (LOC) attributes on TBUF or CLB logic, including D-type flip-flops, CLBMAP symbols, or CLB primitives. Constraints on IO logic are not affected.

–o Suppress Registered Signal Ordering

The -o option suppresses registered signal ordering. It leaves the sorting of registered data bits to the standard pin-saving algorithms. This option also prevents XNFMAP from isolating register logic within CLBs. See the section "Register Ordering" for an explanation of how XNFMAP handles registered signal ordering.

Note: Although this option makes a design denser, it can also affect place and route performance, resulting in higher delays and more unrouted nets. Use this option if you are willing to trade performance for density.

–p Specify the LCA Package Type

The -p option changes the part/package specification for the design. Xilinx strongly recommends that you only change the package specification, and not the part, at this point in the design flow. On the command line, the new part/package specification follows the -p option, as illustrated by this example.

`xnfmap -p` *specification input*`.xtf` *output*`.map`

Note: If you need to modify the part size, it should be changed in the original design or by using the XNFMerge option to change the part. Be sure to run XNFPrep on the design with the correct part type prior to running XNFMAP.

–q Respect File= Macro Boundaries

The -q option maps only logic elements from the same level of hierarchy in a single CLB, with each level of hierarchy defined by the presence of a "FILE=" attribute on the original macro symbol. This option differs from the -a option, which defines a single level of hierarchy for every user-defined macro in the design.

Note: Use of this option can lead to less efficient use of the FPGA resources since it usually restricts XNFMAP's mapping functions. Do

not use this option if you used the XNFMerge option that suppresses storing the hierarchy information in the design file.

–r Reduce the Number of CLBs

The -r option reduces the number of CLBs used in the design at the expense of logic levels. XNFMAP does this by favoring 3-input functions over 4-input functions for XC2000-family designs, or by favoring 4-input functions over 5-input functions for XC3000-family designs. Functions with few inputs can generally be packed more tightly into CLBs than functions with many inputs. Also they can often be combined with a flip-flop into one CLB.

The CLB reduction might not be optimal on some designs, causing the number of CLBs to increase rather than decrease. This might occur on small designs or on designs using relatively few flip-flops.

Note: This option results in a more efficient use of CLB resources, but it may increase the number of levels of logic needed to implement wide functions, and can increase the routing resources needed to implement the design in an FPGA.

–s Relax the Signal Combining Requirements

The -s option attempts to fit more logic into CLBs by reducing the minimum signal combining requirements.

Note: Although this option makes a design denser, it can also affect place and route performance, resulting in higher delays and more unrouted nets. Use this option if you are willing to trade performance for density.

–u Respect Guide File Hierarchy

The -u option maps only logic elements from the same level of hierarchy in a single CLB, with each level of hierarchy defined by the hierarchy in the guide file. Therefore, XNFMAP ignores the hierarchy in the design file.

The XNFMAP Process

When partitioning an XTF file, the XNFMAP program performs the following tasks.

- Reads the input design file
- Reads the design guide file, if applicable
- Uses any design file CLBMAP symbols and the guide file to map logic
- Maps remaining logic into FPGA resources
- Creates output files

The following sections discuss each step in detail.

Input Design and Design Guide Files

XNFMAP first performs the following tasks.

- Reads the input *design*.xtf file.
- Checks for conflicting location control constraints (LOC= or LOC<>) on all symbols and external pads. Any conflicting constraints produce an error.
- Reads the *design*.pgf or *filename*.pgf guide file if you specify the -k or -h option, respectively.
- Resolves any discrepancies between the guide file and design file information. See the "Using the Partitioning Guide File" section for more information.

CLB Mapping

XNFMAP maps CLBs according to user-defined CLBMAPs and guide file information. XNFMAP processes CLBMAPs in the following order.

1. Closed CLBMAPs in the design file
2. Closed CLBMAPs in the guide file, if applicable
3. Open CLBMAPs in the design file (guide file CLBMAPs supercede, if applicable)

Logic Mapping into FPGA Resources

XNFMAP then performs the following tasks.

- Maps remaining logic into FPGA resources — CLBs, IOBs, and TBUFs
- Names CLBs and IOBs

Output Files

XNFMAP creates the following output files.

- Creates a *design*.map output file, if no errors exist
- Creates a *design*.pgf output guide file and renames the previous guide file with a .pbk extension, if no errors exist
- Creates a *design*.crf cross-reference file

Register Ordering

For APR or PPR to achieve an optimal placement of register logic, XNFMAP must properly partition the flip-flops. XNFMAP achieves these goals by performing *register ordering*.

For example, the program combines register bits 0 and 1 into one CLB, bits 2 and 3 in another CLB, and so on. If XNFMAP cannot combine a register flip-flop with another flip-flop from the same register, the flip-flop should be isolated in a CLB to prevent unrelated logic from distorting the placement.

XNFMAP pairs flip-flops in CLBs in sequential alphabetic name order based on their output signal names. XNFMAP identifies flip-flops as belonging to a specific register if at least four flip-flops share a common clock signal and have similarly named outputs. See "Naming Conventions" below for more information.

XNFMAP isolates a register flip-flop in a CLB if it cannot combine that flip-flop with another flip-flop from the same register. In this manner, the APR and PPR programs can place the register CLBs correctly, since there is no non-register logic in those CLBs.

Naming Conventions

For XNFMAP to recognize and order register bits properly, use the following naming conventions.

- Give register names at least one alphabetic character, and do not use the underscore (_) character. The allowances made for OrCAD designs are described in the following section, "Register Ordering for OrCAD/SDT Designs."
- Name these signals in sequential alphabetic order. XNFMAP attempts to pair the register flip-flops in CLBs according to this order, provided the names are identical except for the final two characters.

For example, consider the following flip-flop outputs.

```
rega09
rega10
rega11
rega12
```

XNFMAP would group rega09 with rega10, and rega11 with rega12, since the names differ only in the final two characters.

If the final character is non-alphanumeric, such as the trailing ">" of a bus signal, XNFMAP uses all but the last three characters in the name for matching. For example, consider the following register outputs.

```
rega<34>
rega<35>
regb<76>
regc<23>
```

XNFMAP would group only rega<34> and rega<35>. The others all have differences beyond the final three characters. For another example, consider the following register outputs.

```
bit045
data86
des032
rega95
```

XNFMAP does not consider any of these similar and would not group them together into CLBs.

Register Ordering for OrCAD/SDT Designs

Since OrCAD appends an underscore (_) followed by a sheet number to each output signal name, XNFMAP uses the two characters before the underscore to determine register ordering. XNFMAP groups these signals if the sheet numbers, as well as all characters except the two before the underscore, are identical.

For example, XNFMAP groups the following signals together.

```
rega34_30
rega35_30
```

However, XNFMAP does not group the following signals.

```
rega24_1
regb35_1
```

XNFMAP does not group the next two signals, since the sheet numbers do not match.

```
rega34_23
rega35_1
```

If a non-alphanumeric character precedes the underscore (_), XNFMAP uses all but the three characters before the underscore for matching. For example, XNFMAP groups the following signals.

```
rega<24>_1
rega<35>_1
```

However, XNFMAP does not group the next two signals.

```
rega<24>_1
regb<35>_1
```

Using a Partitioning Guide File

XNFMAP enables you to guide your design by using either of the following methods.

- Guide by XNF — Using the design.pgf files created by a previous XNFMAP run.
- Guide by LCA — Using a placed-and-routed LCA file previously created by the LCA2XNF program.

The XNFMAP guide file is created from either a placed and routed LCA file, or from a previous design iteration. To use a guide file, you must specify the -k or -h option. XNFMAP reads the partitioning guide file and uses the indicated mapping for the new design iteration wherever possible.

APR and PPR attempt to match CLBs and IOBs in the guide design with those in the new design. For best results, the design file and the guide file should be similarly partitioned, only differing where logic has been changed or added.

Guide by PGF

XNFMAP automatically creates a partitioning guide file, *design*.pgf, each time the program is run without any errors. XNFMAP writes CLBMAP symbols in the PGF file that record the partitioning of the logic; therefore, the CLBMAP symbols can be used to guide the mapping process of a future iteration.

XNFMAP performs the following steps when using a PGF file to guide your design.

- Reads the file *design*.pgf and uses the CLBMAP symbols to guide the partitioning process.
- Compares the CLBMAPs in the guide file to any CLBMAPs in the design file.

If any CLBMAPs in the guide file have output signals (CLB X or Y pins) that no longer exist in the design file, XNFMAP disregards that CLBMAP. If any guide file CLBMAPs have input signals that no longer exist XNFMAP issues a warning but keeps the CLBMAP.

XNFMAP disregards a CLBMAP in the guide file if either of its output signals are also indicated as outputs on a closed CLBMAP in the design file. If an open CLBMAP in the design file has the same output signals as a guide file CLBMAP, XNFMAP disregards the design file CLBMAP, since the open CLBMAP does not define complete boundaries for a CLB. Refer to the following section, "Partitioning Logic on a Schematic," for more information about opened and closed CLBMAPs.

Note: To force XNFMAP to disregard all CLBMAPs in the design file, specify both the -m and -k options.

- Rewrites the PGF to record the modified mapping and incorporate any logic changes once partitioning is complete. Therefore, the PGF always represents the most recent partitioning of the design.
- Backs up the previous PGF as *design*.pbk.
- Issues a summary of the guided partitioning, written to both the screen and the CRF.

The following figure shows the steps performed when a PGF file is used to guide a design.

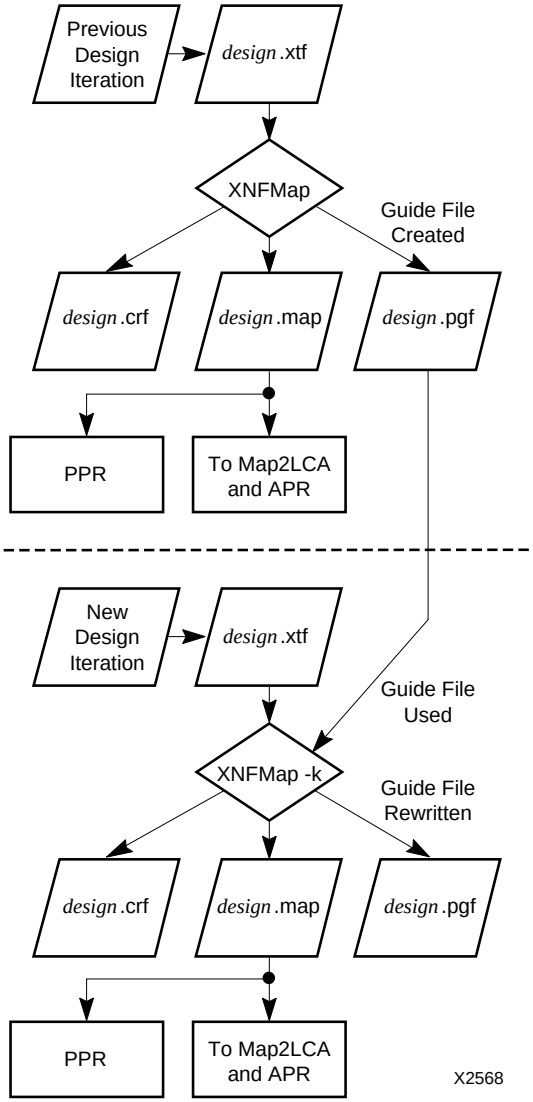


Figure 3-1 Guide by PGF

Guide by LCA File

If a design has already been placed and routed, the LCA file can be used to create a PGF for XNFMAP. This file can be the same LCA file that is used to guide placement and routing in APR and PPR.

To use an LCA file as a guide file, you must perform the following procedures.

- Create a PGF guide file from an LCA file
- Use the guide file to partition your design
- Preserve original partitioning

The following sections describe each procedure in detail.

Creating a Guide File from an LCA File

To create a guide file from a placed and routed LCA file, perform the following steps.

1. Run the LCA2XNF program without specifying any options on the placed and routed LCA file. Access LCA2XNF via XDM or enter the following syntax on the command line.

   `lca2xnf` *design*`.lca`

   This process generates an equivalent XNF including a record of each CLB in the design. You can use this XNF as the PGF; therefore, rename it with the PGF extension and the new design name, for example, *design*.pgf.

2. Rename the AKA file to match the new design name, if applicable. MAP2LCA with the -a option generates the AKA file, which contains the mapping of short alias names to long hierarchical names. The recommended design flow does not include creating an AKA file.

If the placed and routed LCA file contains names shortened by the MAP2LCA program, the AKA file must be present in the current directory. If necessary, rename this file to match the new design name, for example, *design*.aka. XNFMAP reads this file to restore the original names. XNFMAP only restores prefixes of the form $<number>; therefore, do not edit AKA file prefixes if performing guided partitioning.

Using the Guide File to Partition Your Design

After creating the guide file, run XNFMAP with the -k option on the XNF for the new design iteration. XNFMAP performs the following steps to use the guide file to partition your design.

- Reads the file *design*.pgf and extracts all CLBs from the design, creating an equivalent CLBMAP for each.
- Adds P (pin-lock) attributes to each CLBMAP it creates, ensuring that the guide file is able to match CLBs.
- Uses these CLBMAPs to guide the partitioning process.

If any CLBMAPs in the guide file have output signals (CLB X or Y pins) that no longer exist in the design file, XNFMAP disregards those CLBMAPs. If any guide file CLBMAPs have input signals that no longer exist, XNFMAP issues a warning but keeps the CLBMAPs.

XNFMAP disregards a CLBMAP in the guide file if either of its output signals are also indicated as outputs on a closed CLBMAP in the design file. If an open CLBMAP in the design file has the same output signals as a guide file CLBMAP, XNFMAP disregards the design file CLBMAP, since the open CLBMAP does not define complete boundaries for a CLB. Refer to the following section, "Partitioning Logic on a Schematic," for more information about opened and closed CLBMAPs.

Preserving Original Partitioning

The objective of guiding with an LCA file is to preserve as much of the original partitioning as possible; therefore, Xilinx recommends that you disregard all CLBMAPs in the design file. To force XNFMAP to ignore all such CLBMAPs, combine the -m and -k options.

After completing partitioning, XNFMAP performs the following steps to create the output files.

- Rewrites the PGF to record the modified mapping and incorporate any logic changes. This PGF uses CLBMAPs to represent the partitioning and replaces the file created by LCA2XNF. Consequently, the PGF always represents the most recent partitioning of the design.
- Issues a summary of the guided partitioning and sends output to both the screen and the CRF.

Note: If you used an AKA file in the original design iteration, it must not contain any user-created prefixes. XNFMAP only restores aliases of the form $-number in the LCA guide process.

The following figure illustrates the steps performed when an LCA file is used to guide a design.

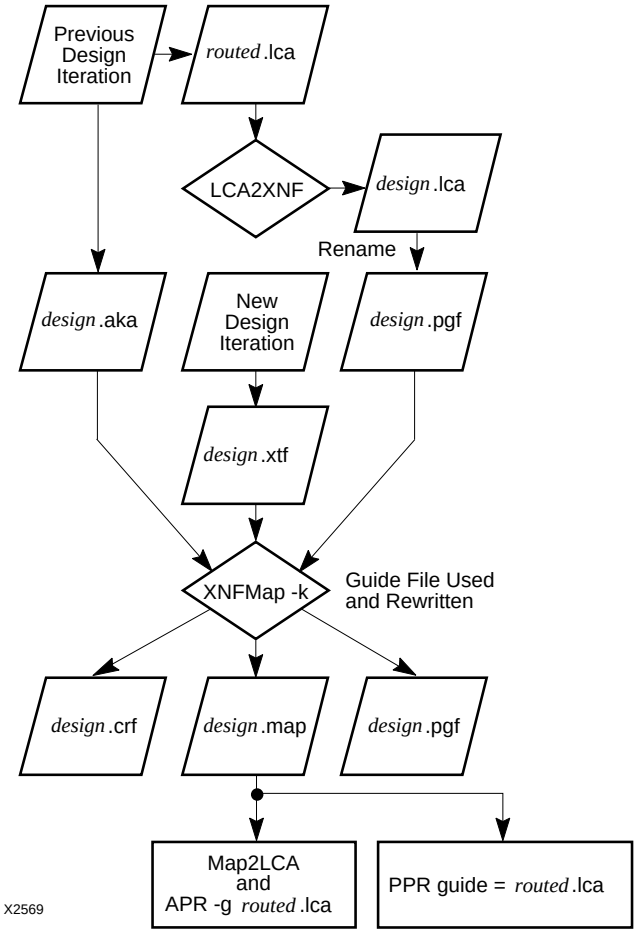


Figure 3-2 Guide by LCA File

Partitioning Logic on a Schematic

You can specify logic partitioning at the schematic level by using the CLBMAP symbol. A CLBMAP is used in conjunction with standard logic elements, such as gates and flip-flops. A CLBMAP implicitly specifies the configuration of a CLB by defining the signals on its pins. Use the CLBMAP symbol to control partitioning when the default partitioning is not acceptable.

You can enter the CLBMAP symbol directly on the schematic and assign signals to its pins. XNFMAP processes this symbol and the appropriate logic, as defined by the input and output signals, and maps them into one CLB. The easiest way to define a CLBMAP is to connect a labeled wire segment to each pin, which connects that pin to the net carrying the same label.

If a CLBMAP specifies an illegal CLB configuration, XNFMAP issues an error and explanation about why the CLBMAP is illegal.

The following table displays the MAP parameters you can attach to CLBMAP symbols.

Table 3-1 MAP Parameters for CLBMAP Symbols

	Closed CLB	Open CLB
Pins Locked	MAP=PLC	MAP=PLO
Pins Unlocked	MAP=PUC	MAP=PUO

Opened and Closed CLBMAPs

A CLBMAP can be either closed or open. You can specify a CLBMAP as either open or closed by attaching the appropriate MAP parameter to the symbol. See the preceding table for the exact conventions.

A closed CLBMAP must specify both the input and output signals for that CLB. XNFMAP partitions a closed CLBMAP exactly as specified, unless the indicated configuration is illegal. XNFMAP does not add any additional logic to a CLB specified with a closed CLBMAP.

An open CLBMAP specifies only the output signals for the CLB. XNFMAP assigns those signals to the CLB output pins and partitions the source logic into the CLB as appropriate. Use an open CLBMAP to specify the function of a CLB without specifying the exact

configuration. If a CLBMAP symbol does not include any function generator input pins (A, B, C, D, E), then XNFMAP automatically assigns a MAP=PUO attribute to the symbol so that is regarded as an open CLBMAP.

Locked or Unlocked CLBMAP Pins

The pins on a CLBMAP can be either locked or unlocked. You can either lock or unlock pins by attaching the appropriate MAP parameter to the symbol. See the preceding table for the exact conventions.

If the CLBMAP pins are locked, the place and route programs do not swap these CLB pins. In most cases, the pins on a CLBMAP remain unlocked, so the pins can be swapped to achieve better routing results. Only the output pins of an open CLBMAP are locked.

If a CLBMAP has unlocked pins, individual CLBMAP pins can be locked by attaching a P (pin-lock) attribute to the corresponding pin. On an open CLBMAP, you can assign P attributes only to output pins.

If a CLBMAP symbol specifies a CLB in the base FGM configuration, this must be indicated to XNFMAP by locking the select pin for the 2-to-1 multiplexer in the CLB. This is done by attaching a P (pin-lock) attribute to the signal on the E-pin (XC3000) or B-pin (XC2000) of the CLBMAP. If XNFMAP cannot configure the CLB using base FGM, it attempts to use a base F or base FG configuration instead.

Note: You do not need to specify the special purpose pins on XC3000 family CLBs (K, RD and EC) on the CLBMAP symbol. XNFMAP assigns the correct signals to these pins. XNFMAP generates a warning, however, stating that no clock, reset, or clock enable signal was specified and gives the XNFMAP-assigned signal name.

Using a CLBMAP in a XC3000 Design

The following figure illustrates an example of a possible mapping of logic into CLBs. In this example, the two flip-flops that generate signals BIT0 and BIT1 are mapped into the same CLB, while the flip-flop that generates signal BIT2 is mapped into a second CLB. If this is not the desired result, a CLBMAP can be used to change the mapping.

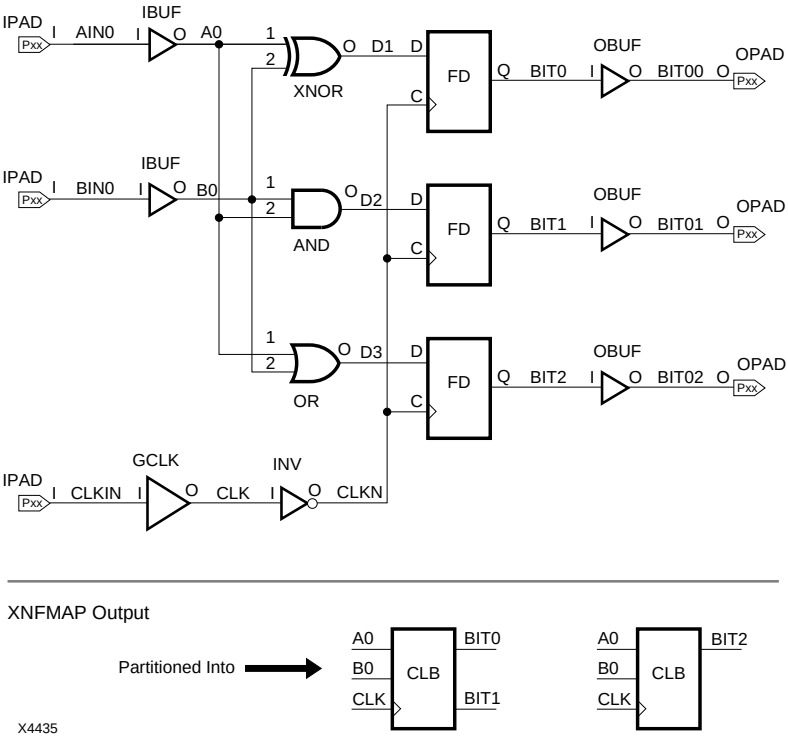


Figure 3-3 Possible Logic Partitioning

The following figure illustrates the same circuit with the addition of a CLBMAP symbol to control the partitioning. Because the CLBMAP specifies signals BIT0 and BIT1 on the X and Y pins, XNFMAP puts the flip-flops that generate BIT0 and BIT1 into the same CLB.

Since the CLBMAP also has input signals specified for the A, B, and K pins, XNFMAP traces the circuit to find the gates that use the specified input signals. In this case, it also puts the XNOR and AND gates that source the two flip-flops into the same CLB with the flip-flops.

After the CLBMAP is processed, XNFMAP partitions the remaining logic. In this example, one more CLB is created for the flip-flop that generates signal BIT2 and its associated OR gate.

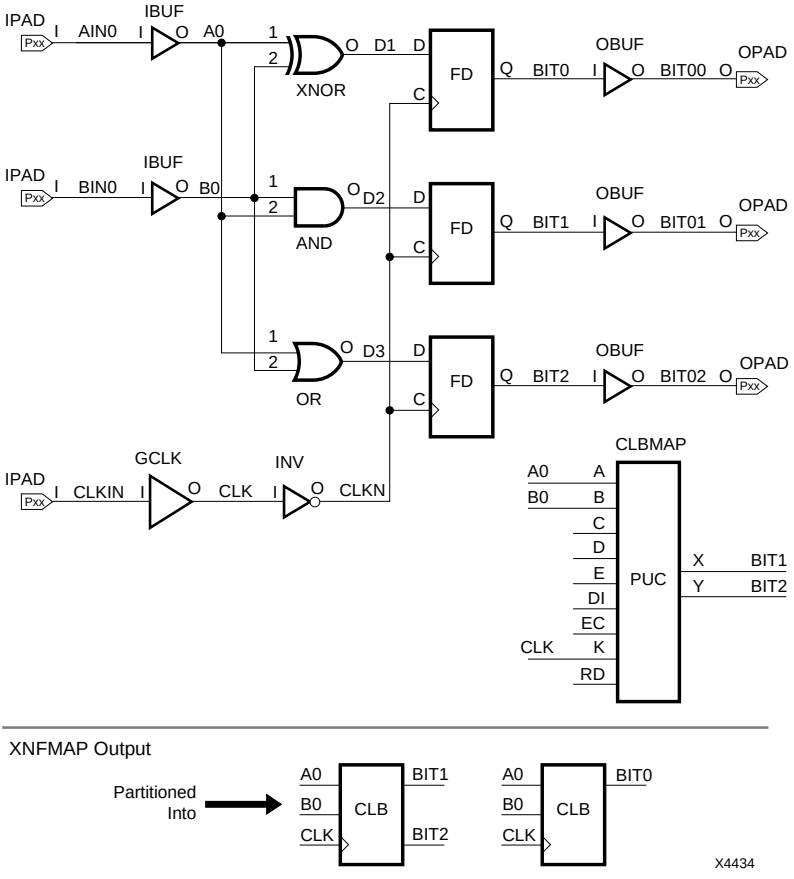


Figure 3-4 Logic Partitioning Controlled by CLBMAP Symbol

Using Explicit (X) Attributes

An explicit (X) attribute on a signal makes that signal an explicit net in the FPGA. An explicit net is external to a CLB. Signals connected to non-combinational logic, such as clock buffers and I/O buffers or registers, are always explicit. However, if you use no explicit attribute, some signals can be absorbed inside a CLB. Such signals are as follows.

- A signal that connects only to combinational logic, which is partitioned into a single CLB function generator
- A function generator output signal that drives only a flip-flop in the same CLB
- The output of a flip-flop that is fed back within the CLB

Use an explicit attribute on your schematic to control the partitioning in the following cases.

- The default partitioning is not acceptable because a particular net must not be absorbed inside a CLB.

Using the BLKNM, HBLKNM, and LOC Parameters to Partition and Place Logic

The BLKNM, HBLKNM, and LOC parameters can influence both partitioning and placement of logic. When attached to flip-flops, BLKNM, HBLKNM, or LOC parameters can influence the partitioning of CLBs. See your specific interface user guide for more information about attaching BLKNM, HBLKNM, and LOC parameters to logic.

LOC parameters are ignored on symbols that already have a fixed location, such as GCLK, ACLK, and OSC.

The following sections describe each parameter in detail.

BLKNM Assignments

The BLKNM parameter assigns FPGA block names to CLB and IOB primitives; I/O primitives; TBUFs; pull-ups; D-type flip-flops; and CLBMAP symbols. The block name must not be a name that is reserved in the XACT*step* Development System. The reserved names include all physical names for the FPGA resources, such as AA, P12, GCLK, and TBUF. The name is applied to the block in the LCA file; to see where any of the blocks are finally placed, give them BLKNMs on the schematic and then search for that BLKNM inside the XACT Design Editor (XDE).

A BLKNM can also specify the pairing of flip-flops within an XC3000 CLB. If you assign two flip-flop primitives to the same BLKNM parameter, XNFMAP pairs them together inside one CLB, if possible.

If you assign different BLKNM parameters to two flip-flops, XNFMAP does not pair them together, unless they are forced together by a single-block LOC constraint or a CLBMAP.

HBLKNM Assignments

The HBLKNM parameter is similar to the BLKNM parameter, except that the file merge program, XNFMerge, hierarchically qualifies and prefixes the name value with the design hierarchy level. For example, XNFMerge changes the HBLKNM value "bit0" on a D-type flip-flop located in the "top/sub1/sub2" design level to an HBLKNM value of "top/sub1/sub2/bit0." This HBLKNM parameter value allows designers to place HBLKNM parameters in macros that are instantiated more than once in the design, relying upon the modification of the HBLKNM to create unique name values.

If two HBLKNM parameters with the same value are used more than once at the same level of design hierarchy, then XNFMAP generates the same HBLKNM value. Identical HBLKNM values cause XNFMAP to group the tagged symbols in the same block, as it does with identical BLKNM values.

LOC= and LOC< > Constraints

You can use the LOC parameter to place flip-flops, TBUFs, I/O pins, or CLBMAPs in one specific location, or to specify multiple allowable locations for these elements. You can also place a LOC= parameter on an RPM and let the XNFMerge program pass the location information down to the logic on the lower level.

Note: You cannot use LOC parameters to place logic gates (ANDs, ORs, and so on). To control logic gate placement, use a CLBMAP and place the LOC parameter on the CLBMAP.

Use the following statements to specify or prohibit locations for the associated logic.

- LOC=*blocks* — Place the associated logic within the area defined by *blocks*.
- LOC<>*blocks* — Do not place the associated logic within the area defined by *blocks*.

The examples below demonstrate how the *blocks* parameter is used to specify locations.

LOC=BC	Place logic in CLB BC.
LOC<>DE	Do not place logic in CLB DE.
LOC=*B	Place logic within the second (B) column of CLBs. The asterisk (*) is a wildcard character that indicates you can place logic in any row of specified column.
LOC=C*	Place logic within the third (C) row of CLBs. The asterisk (*) is a wildcard character that indicates you can place logic in any column in the third row.
LOC=AB;LOC=BC;LOC =DE	Place logic in CLB AB, BC or DE. There is no significance to the order of the LOC= statements. Separate each statement with a semicolon (;).
LOC=BC:DF	Place logic within the rectangular block defined by the CLB BC in the upper-left corner and CLB DF in the lower-right corner.

Single-Block CLB Placement

You can assign CLB primitives, CLBMAP symbols, and individual D-type flip-flops to a single CLB location, a list of CLB locations, a row or column of CLB locations, or a rectangular block of CLB locations. The standard two-letter block name used in XDE identifies CLB locations. The upper-left CLB is identified as AA.

You can use a single-block LOC constraint, that is, an LOC= statement specifying a single CLB location, to influence the pairing of flip-flops within an XC3000 CLB. If you assign two flip-flops to the same single-block location, XNFMAP pairs them together inside that CLB, if possible.

Any single-block LOC constraint specified on a flip-flop is transferred to the resulting CLB, unless superseded by a CLBMAP location parameter. XNFMAP passes LOC constraints on CLB primitives and CLBMAPs directly to the resulting CLB.

Multiple-Block LOC Placement

XNFMAP uses a multiple-block LOC constraint on a flip-flop specifying a range of CLB locations as a tie-breaker for possible pairings of flip-flops in CLBs. If XNFMAP finds two flip-flops that it would normally combine in one CLB, it combines them if one or both flip-flops carry no LOC constraints, or the LOC constraints on the two flip-flops intersect. If there is an intersection, XNFMAP combines the two flip-flops and passes the intersection of the LOC constraints to the resulting CLB. If a flip-flop has several optimal pairs, XNFMAP chooses the pair for the largest possible LOC intersection.

XNFMAP directly passes a multiple-block LOC constraint on a CLB primitive or CLBMAP symbol to the resulting CLB. The examples in the table above illustrate the format of CLB constraints. If the indicated logic does not fit into the specified blocks, XNFMAP generates an error.

If XNFMAP generates an error, change the LOC constraints on your schematic.

IOB Placement Examples

You can assign I/O pads, buffers, registers and pull-ups to individual IOB locations on an edge or half-edge of the part. The corresponding package pin designation for a specific location identifies IOB locations. To specify an edge or half-edge of the part, use the following designations.

T	—	top edge
R	—	right edge
B	—	bottom edge
L	—	left edge
TL	—	top-left
TR	—	top-right
RT	—	right-top
RB	—	right-bottom
BR	—	bottom-right
BL	—	bottom-left
LB	—	left-bottom
LT	—	left-top

For the XC3000 family, XNFMAP automatically assigns I/O pads for the direct clock inputs (TCLKIN and BCLKIN) to the appropriate IOBs. You do not need to specify an LOC for these inputs.

The following examples illustrate the format of IOB constraints.

LOC=P13	Place I/O element in location P13. For PGA packages, the letter-number designation is used, for example B3.
LOC<>A5;LOC<>B3	Do not place I/O element in PGA locations A5 or B3.
LOC=TL	Place I/O element on the top-left half of the part.

TBUF and Pull-up Placement

You can assign internal 3-state buffers (TBUFs) to an individual TBUF location; a list of TBUF locations; or a row or column of TBUF locations. TBUF locations are identified by the adjacent CLB. Therefore, the two TBUFs above and to the left of CLB BB are TBUF.BB.1 and TBUF.BB.2.

You can also assign horizontal longline pull-ups resistors (pull-ups) to various pull-up locations. If you assign the TBUFs associated with a pull-up to specific locations, the pull-up resistors follow as appropriate. Pull-up locations are identified by the adjacent TBUF. Therefore, the pull-up adjacent to TBUF.HA.1 is labeled PU.HA.1.

The following examples illustrate the format of TBUF constraints. Pull-up constraints are similar, with PU replacing TBUF. The asterisk (*) is a wildcard character that indicates any string of characters.

LOC=TBUF.BC.1	Place TBUF in the location BC.1.
LOC=TBUF.B*	Place TBUF in any location in the second row of TBUFs.
LOC<>TBUF.*D	Do not place TBUF in any location in the fourth column of TBUFs.

Files

The following sections illustrate the output files produced by XNFMAP. The following figure illustrates sample output for this example, sample.xtf.

In this example, XNFMAP has been run previously on the XTF file, creating a partitioning guide file for the design, sample.pgf. XNFMAP is invoked with the following command line entry first to change the part type.

```
xnfmap -p 3020pc68-125 sample
```

Then XNFMAP is run with the -k option, which indicates that the sample.pgf file guides partitioning.

```
xnfmap -k sample
```

XNFMAP reports its progress on the screen and stores the same information in the sample.crf file. XNFMAP stores any error or warning messages in the CRF (cross-reference file) described later in this section.

Output File

The output file includes the messages sent to the screen during processing. The following figure illustrates an example of an output file. Each section of the output file corresponds to a letter on the figure, as follows.

Header

The header, as indicated by A in the following figure, gives the program version number; the date and time of the translation; and indicates which options you selected, for example, Guide.

Status Messages

The status messages, as indicated by B in the following figure, show the progress of XNFMAP during processing of the design.

Design Summary

The design summary, as indicated by C in the following figure, gives various statistics about the partitioned design, including CLB and IOB counts.

```
A  XNFMAP Ver. 5.00
   (c) Copyright 1987-1994 Xilinx Inc. All rights reserved
   Options selected:   Guide

B  Reading file SAMPLE.xtf...

   Checking network for logical errors...

   Checking Guide file for errors...

   Preparing design...

   Processing guide symbols...

   Guide SYMBOL SUMMARY:
   2 of 2 Guide file guide symbols successfully used

   Assigning logic to LCA elements...

   Preparing Guide file...

   Checking mapped logic blocks...

C  Design SUMMARY:
   Part type=3020PC68-100
   2 of 64 CLBs used
   6 of 58 I/O pins used
   0 of 6 internal IOBs used
   0 of 16 internal 3-state signals used (0 TBUFS used)

   2   CLB flipflops used

   Writing design file SAMPLE.map...

   Writing guide file SAMPLE.pgf...
```

Figure 3-5 The Screen Output of XNFMAP

Cross-Reference File

XNFMAP writes most output information to the sample.crf file. The CRF also contains other information about the partitioned design, and gives a cross-reference report of all CLBs and IOBs XNFMAP creates. The following figure illustrates a portion of the sample.crf file. The CRF includes the following sections, each designated by a letter.

File Header

The file header, as indicated by A in the following figure, lists the program version number, input design name, and which options were selected, in this example, Guide.

Guide Symbol Summary

The guide symbol section, as indicated by B in the following figure, shows how the design file CLBMAP symbols and sample.pgf files were used. The listing includes the number of guide symbols from the guide file that were successfully used.

Check of Mapped Logic Blocks

This section, as indicated by C in the following figure, lists any problems corrected in the final mapping of CLB and IOB logic blocks.

Design Summary

The design-summary section, as indicated by D in the following figure, shows the part utilization of the partitioned design.

CLB Cross-reference

The CLB cross-reference section, as indicated by E in the following figure, shows each CLB that XNFMAP generates and lists the logic elements of that CLB.

IOB Cross-reference

The IOB cross-reference section, as indicated by F in the following figure, shows each IOB that XNFMAP generates and lists the I/O elements of that IOB.

```
A  Design: SAMPLE, XNFMAP Ver 5.00 run at Thu Dec 16 09:12:00
1993
   Options selected:   Guide
   CHECKING NETWORK FOR LOGICAL ERRORS:
   CHECKING Guide FILE FOR ERRORS:
   PREPARING DESIGN:

B  Guide SYMBOL MAPPING CONTROL:

   Guide SYMBOL SUMMARY:
   2 of 2 Guide file guide symbols successfully used

   ASSIGNMENT OF LOGIC TO LCA ELEMENTS:

C  CHECKING MAPPED LOGIC BLOCKS:

D  Design SUMMARY:
   Part type=3020PC68-100
   2 of 64 CLBs used
   6 of 58 I/O pins used
   0 of 6 internal IOBs used
   0 of 16 internal 3-state signals used (0 TBUFS used)

   2   CLB flipflops used

E  CLB CROSS-REFERENCE REPORT:

   CLB #1  Name = Q1'

   QX = signal Q1',
       latch = symbol Q1'
   QY = signal Q0',
       latch = symbol Q0'
   F = signal Q0'
       function contains:
       <no symbols - direct connection>
   G = signal CMP'
       function contains:
       symbol CMP (type=AND),
           output signal = CMP'
       symbol CMP1 (type=XNOR),
           output signal = CMP1'

   CLB #2  Name = CMP0'

   F = signal CMP0'
       function contains:
       symbol CMP0 (type=XNOR),
           output signal = CMP0'

F  IOB CROSS-REFERENCE REPORT:

   IOB #1: Name = QOUT'

   IOB #1 locked on block P17'
```

```
Defined by signal QOUT'
O  = signal Q1'

IOB #2: Name = CLKIN'

IOB #2 locked on block P16'
Defined by signal CLKIN'
I  = signal CLK'
.
.
.
```

Figure 3-6 Sample.crf File

MAP File for MAP2LCA and APR

The partitioned design is written to the sample.map file. The sample design is targeted for a 3020 non-A part, which uses MAP2LCA and APR for implementation.

This file defines each CLB, IOB, and TBUF used in the design. The following figure illustrates a portion of the MAP file. Each section of the MAP file corresponds to a letter designation on the figure, as follows.

File Header

The file header, as indicated by A in the following figure, records the LCA netlist version number, the LCA part type, and information about programs used to create this design file.

IOB Symbol and Model Records

A symbol (SYM) record of type IOB, as indicated by B in the following figure, defines the configuration of an I/O block. The corresponding MODEL record describes the I/O elements partitioned into that IOB.

CLB Symbol and Model Records

A symbol (SYM) record of type CLB, as indicated by C in the following figure, defines the configuration of a logic block. The corresponding MODEL record describes the logic elements partitioned into that CLB.

```
A  LCANet, 5
   PROG, XNFMAP, Ver. 5.00, Created from sample3.xtf: Tue Dec
21 09:12:00 1993: Options:
   PROG, WIR2XNF, 5.00, Tue Dec 21 09:11:02 1993,
   PART, 3020pc68-125

B  SYM, CMP0EXT, IOB
   CFG, Base IO
   CFG, Config IN:I OUT: TRI:
   PIN, I, O, CMP0,
   MODEL
   SYM, CMP0IN, IBUF
   PIN, O, O, CMP0,
   PIN, I, I, CMP0EXT,
   END
   ENDMOD
   END
   EXT, CMP0EXT, I,
   .
   .
   .

C  SYM, Q1, CLB
   CFG, Base FG
   CFG, Config X:QX Y: RSTDIR: ENCLK: DX:F DY:G CLK:K:NOT
F:QY G:A:B:C
   CFG, Equate F=QY
   CFG, Equate G=(~(B@C)*A)
   PIN, A, I, CMP0,
   PIN, B, I, B1,
   PIN, C, I, A1,
   PIN, K, I, CLK,
   PIN, X, O, Q1,
   MODEL
   SYM, CMP1, XNOR
   PIN, O, O, CMP1,
   PIN, 2, I, B1,
   PIN, 1, I, A1,
   END
   SYM, CMP, AND
   PIN, O, O, CMP,
   PIN, 2, I, CMP1,
   PIN, 1, I, CMP0,
   END
   SYM, Q0, DFF
   PIN, Q, O, Q0,
   PIN, D, I, CMP,
   PIN, C, I, CLK, , INV
   END
   SYM, Q1, DFF
```

```
PIN, Q, O, Q1,
PIN, D, I, Q0,
PIN, C, I, CLK, , INV
END
ENDMOD
END
EOF
.
.
```

Figure 3-7 Sample.map

MAP File for PPR

You can target the same design for a 3020A part for implementation through PPR by changing the part type to a 3020APC68, as illustrated in the introductory section, "Output Files." The MAP file generated for PPR does not contain MODEL records. Instead, this MAP file uses ASSIGN parameters to define the mapping of symbols into CLB function generators, and CLBNM parameters to define the mapping of function generators into CLBs. The PPR MAP file contains the following sections, each designated by a letter on the figure.

File Header

The file header, as indicated by the letter A in the following figure, records the LCA netlist version number, the LCA part type, and information about programs used to create this design file.

IO Symbols

The IO symbols, as indicated by the letter B in the following figure, are left unchanged. PPR maps them into IOBs.

Combinatorial Symbols

XNFMAP annotates the original combinatorial symbols, as indicated by the letter C in the following figure, with an ASSIGN parameter to link them to the function generator that implements the logic.

DFF Symbols

XNFMAP annotates the original DFF symbols, as indicated by the letter D in the following figure, with a CLBNM parameter to link

them to the CLBs to which they are mapped, and a BEL parameter to define the initial placement of the DFF in the CLB block.

Function Generator Symbols

The FG symbols XNFMAP creates, as indicated by the letter E in the following figure, are annotated with the equation they implement, the name of the associated CLB, and the initial placement of the function generator in the CLB block.

```
A  LCANet, 5
   PROG, XNFMAP, Ver. 5.00, Created from sample3.xtf: Tue Dec
   21 09:12:00 1993: Options:
   PROG, WIR2XNF, 5.00, Tue Dec 21 09:11:02 1993,
   PART, 3020apc68-125

B  SYM, CMP0IN, IBUF
   PIN, O, O, CMP0,
   PIN, I, I, CMP0EXT,
   END
   EXT, CMP0EXT, I,
   .
   .
   .

C  SYM, CMP1, XNOR, ASSIGN=FG_CMP
   PIN, O, O, CMP1,
   PIN, 2, I, B1,
   PIN, 1, I, A1,
   END
   SYM, CMP, AND, ASSIGN=FG_CMP
   PIN, O, O, CMP,
   PIN, 2, I, CMP1,
   PIN, 1, I, CMP0,
   END

D  SYM, Q0, DFF, CLBNM=Q1, BEL=FFY
   PIN, Q, O, Q0,
   PIN, D, I, CMP,
   PIN, C, I, CLK, , INV
   END
   SYM, Q1, DFF, CLBNM=Q1, BEL=FFX
   PIN, Q, O, Q1, , CLBPIN=X
   PIN, D, I, Q0,
   PIN, C, I, CLK, , INV
   END

E  SYM, FG_CMP, FG, EQN=(~(I1@I2)*I0), CLBNM=Q1, BEL=G
   PIN, I0, I, CMP0, , CLBPIN=A
   PIN, I1, I, B1, , CLBPIN=B
```

```
PIN, I2, I, A1, , CLBPIN=C
PIN, O, O, CMP,
END
EOF
```

Figure 3-8 Example MAP File for PPR

Warning Messages and Recovery Techniques

XNFMAP might display the following warnings messages during processing.

`Warning 222.` *number* `too many CLBs used.`

The design requires more than the available number of CLBs in the target part type. Change to a larger part type or eliminate some of the logic.

`Warning 223.` *number* `too many IOBs used.`

The design requires more than the available number of IOBs in the target part type. Change to a part type with more pins or eliminate some of the I/Os.

`Warning 224.` *number* `I/O pins used but only` *number* `available.`

The design requires more I/O pins than are available in the package type. Either change to a package with more pins or reduce the number of I/O pins. The number of I/O pins and the number of IOBs does not always match because some packages contain unbonded IOBs.

`Warning 241.` *TS* `attributes on` *pinname* `pin of` *symboltype symbolname* `will be lost within the function generator that sources signal` *signalname*`.`

Move the timing specification parameter to another input pin in the design.

`Warning 246. TBUF does not have a T pin signal and will not be treated as a 3-state buffer.`

The indicated TBUF does not have a signal attached to the T pin; therefore, XNFMAP removes it. Check if the source logic has been removed, and check the schematic for misconnected wires.

```
Warning 254. No CLBs used.
```

No CLBs were created when the design was partitioned. This typically happens when XNFPrep disables or removes all logic. If there are signals without sources or loads in the design that cannot be removed, use an S (save) attribute to preserve them or specify the XNFPrep option “savesig=true.”

```
Warning 255. No IOBs used.
```

No IOBs were created when the design was partitioned. If only a partial design is being processed and no I/O pins are used, you can ignore this message. You can preserve logic not sourced or loaded by IOBs by using S (save) attributes, using the savesig=TRUE option in XNFPrep, or specifying the -e option to inhibit logic removal.

```
Warning 256. No I/O pins used (only internal IOBs
used).
```

Only internal (unbonded) IOBs were used when the design was partitioned. If you indicated any bonded I/Os in the input design, check if any IOBs without sources or loads have been removed.

```
Warning 259. No connection to pin on symbol.
```

No signal is attached to the indicated pin on the specified symbol. This condition can result in logic removal.

Check your design for sourceless/loadless signals. Check the logic removal section in the report file from XNFPrep.

```
Warning 260. No connection to clock, set or reset
pins on symbol.
```

No signals are attached to the indicated pins on the specified symbol. This condition can result in the symbol being removed.

Check your design for sourceless/loadless signals. Check the logic removal section in the report file from XNFPrep.

```
Warning 261. Connection made to [clock pin | data pin]
but none made to [clock pin | data pin] on symbol.
```

A flip-flop or latch is illegally configured. Connect both the data (D) and clock (C or L) signals.

`Warning 263. Signal has no effect and is ignored.`

Signals in a combinational logic loop do not source a latch, a flip-flop, or an IOB. Check your design for errors.

`Warning 277. No source for signal` *signalname*`.`
`Cannot include signal` *signalname* `in mapped CLB.`

The indicated signal has no source or load pins. This condition can occur if logic that sources or loads the signal has been removed. If a signal is intentionally left dangling, you can preserve it by attaching an S (save) attribute, using the savesig=TRUE option in XNFPrep, or using the -e option to inhibit logic removal. Signals without loads and sources are removed even if you specified the -e option; attach an S (save) attribute to preserve such signals.

`Warning 279. No signals connected to symbol`
symbolname `type` *symboltype*`.`

The indicated symbol has no signals connected to it. XNFMAP removes it unless you specify the -e option.

Check your design for sourceless/loadless signals. Check the logic removal section in the report file from XNFPrep.

`Warning 280. No` *input pins* | *output pins* `found on symbol`

The indicated symbol has no signals connected to it. XNFMAP removes it unless you specify the -e option.

Check your design for sourceless/loadless signals. Check the logic removal section in the report file from XNFPrep.

`Warning 281. Invalid signal name` *signalname*`.`

The indicated signal name is not a valid FPGA name and is ignored. FPGA names can contain only letters, digits, underscores (_), dashes (-), dollar signs ($), and angle brackets (< and >). Names must contain at least one non-digit character and no spaces.

`Warning 286. External signal has no I/O symbols`
`connected to it.`

An external signal — one that connects to an I/O pad — must be connected to an I/O symbol, such as IBUF or OBUF. The external signal has no source or load.

Check the design for errors in connecting the PAD and I/O symbols.

`Warning 292. Too many` *typename* `symbols connected to signal` *signalname*`.`

This message typically results from using more than the available number of TBUFs or pull-ups. The number of TBUFs available on a horizontal longline depends on the FPGA part type. Reduce the number of TBUFs or use a part with more resources. There are two pull-ups available for each horizontal longline. Specify no more than two pull-ups per signal.

`Warning 294.` *number* `too many internal three-state signals used.`

This message typically results from using more than the available number of horizontal longlines. The number of horizontal longlines available depends on the FPGA part type. Reduce the number of 3-state signals or use a part with more horizontal longlines.

`Warning 323.` *Signal* `on guide symbol is not used.`

The design logic does not use a signal specified on a CLBMAP. Check that you added the correct signals to the CLBMAP pins.

`Warning 324. [`*clock* | *clock enable* | *reset*`] signal on guide symbol doesn't match actual latch [`*clock* | *clock enable* | *reset*`] signal.`

A guide symbol {clock | clock enable | reset} signal is ignored. Check that the indicated signal on the CLBMAP matches the corresponding signal on the design logic.

`Warning 327. CLBMAP symbol` *symbolname* `includes symbols from different levels of the design hierarchy. Hierarchy information will be disregarded.`

Use the -a or -q options to prevent mixing logic from different levels of the hierarchy in a single CLB. However, these options are not recommended for the most efficient use of FPGA resources.

`Warning 335. Output signal is specified for guide symbol but output not needed. Output signal will not be generated.`

The indicated signal is specified as a CLBMAP output, but the signal had no load and was removed. Check the output signal for improper connection to loads. If the signal was intentionally left without a load,

attach an S (save) attribute to it to prevent XNFMAP from removing it.

```
Warning 346. [clock | clock enable | reset] signal not
specified for guide symbol.
```

XNFMAP uses the signal name for the CLB **{**clock | clock enable | reset**}**. Since the CLBMAP does not assign a signal to the indicated pin, XNFMAP chooses the appropriate signal and issues this warning.

```
Warning 348. symbol has a [GND|VCC] signal for the
[clock | clock enable | reset] signal.
```

Since this condition significantly changes the logic, XNFMAP ignores the guide symbol instructions. A CLBMAP can only accommodate power signals if they do not significantly change the logic to be mapped into a CLB. XNFMAP only accepts the following power connections.

RESET pin tied to GND (RESET disabled)

SET pin tied to GND (SET disabled)

EC pin tied to VCC (clock enabled)

In these three cases, the pin connected to a power signal is removed. However, if power signal connections cause logic to change, such as an EC pin tied to GND, reset pin tied to V_{CC}, or V_{CC} tied to the input of a logic gate, XNFMAP issues this warning and ignores the CLBMAP instructions.

```
Warning 352. CLB has invalid configuration
information. More information about the invalid
CLB will follow.
```

This warning might indicate that the configuration information on a CLB primitive is incorrect. The warning might also occur if unused signals have been removed from the CLB.

Check the configuration information specified for the CLB primitive.

```
Warning 358. DFF symbolname and DFF symbolname have
the same location attribute but different
blockname attributes.
```

Single-block location specification takes precedence, so the two DFFs are put into the same CLB. Although the two indicated flip-flops were

given different BLKNM parameters, their location specifications match. XNFMAP follows the location specification and pairs the two flip-flops even though they carry different BLKNM parameters. Correct either the BLKNM or LOC parameters.

```
Warning 359. DFF symbolname and DFF symbolname have
same blockname attribute but incompatible
location attributes.
```

XNFMAP does not pair the two DFFs. Although matching BLKNM parameters indicate that the two flip-flops should be paired, the location specifications require that they are separate. Correct either the BLKNM or LOC parameters.

```
Warning 363. MUX for CLB removed due to disabled
[E|B] signal.
```

XNFMAP might have removed the G function and invalidated the CLB. The select signal for the 2-to-1 multiplexer in a base FGM CLB must be on the B (XC2000) or E (XC3000) pin. If this signal is missing or disabled, XNFMAP removes the G function. Check the MUX-select signal.

```
Warning 365. Exceeded resources of part type. Do
not use the MAP file for translation to an LCA
file. Use the MAP file for analysis only. To
continue, alter the design to fit the part or
choose a larger part, and execute XNFMAP again.
Note that less than optimal partitioning may have
occurred as the program tried to force the design
into this part. Once it is correctly sized, it
may use more CLBs.
```

To continue, alter the design to fit the part or choose a larger part, and execute XNFMAP again.

```
Warning 366. DFF symbolname eventually sources only
itself.
```

If XNFMAP finds a flip-flop output that leads only to its own input, it removes that flip-flop. The flip-flop is only preserved if it sources logic that eventually leads to an output block.

Check the design for proper connections of signals to pins.

```
Warning 367. Since CLB MUXes the F and G
functions, removal of one function forces removal
of the other function.
```

If a CLB uses the base FGM configuration, both inputs to the 2-to-1 multiplexer must be preserved to create a valid function. If XNFMAP removes one of these functions, it always removes the other with it. Check both functions that source the MUX.

```
Warning 368. Cannot map guide symbol using the
MUX.
```

Since the MUX select signal (pin B or E) is locked, XNFMAP tried to program the MUX, but failed. Instead XNFMAP tries to follow the guide-symbol directions for a non-FGM mode CLB.

If the E (XC3000) or B (XC2000) pin on a CLBMAP is locked, XNFMAP tries to fit the indicated logic into a base FGM CLB, using the B or E pin as the 2-to-1 multiplexer select signal. If the function cannot be implemented as base FGM, XNFMAP issues this message and attempts to use base F and base FG before disregarding the CLBMAP. If the CLB is not supposed to be base FGM, remove the pin lock from the B or E pin.

Warning 369. Cannot use signal *signalname* for [*reset* | *clock* | *set* | *FF-Data-pin*] signal in CLB specified by GUIDE symbol *symbolname* if it is configured as an FGM CLB.

A CLBMAP indicates that an XC2000 CLB should use base FGM, but the specified flip-flop input is not specified on the appropriate input pin of the CLBMAP.

If the CLB uses the MUX, the flip-flop signal must be either the MUX signal *name* or one of the direct input signals to the CLB. If a CLB is base FGM, the appropriate CLB input pin must source the flip-flop inputs. Do not use the F function generator since it goes to the MUX. Correct the CLBMAP to specify the flip-flop input.

```
Warning 371. Changing TBUF symbol to type OBUFZ
symbol.
```

If an XNF interface does not support the 3-state output buffer (OBUFZ), XNFMAP recognizes an OBUF followed by a TBUF as being equivalent to an OBUFZ. XNFMAP makes this substitution.

```
Warning 400. [CLB|IOB] name. This message
indicates an error in the configuration for a CLB
or IOB primitive.
```

An explanation of the error follows this message.

```
Warning 401. [CLB | IOB] symbolname msg.
```

An error in the configuration for a CLB or IOB primitive. An explanation of the error follows this message.

```
Warning 424. Unable to open Guide file for
reading.
```

Check that the guide file exists in the current directory and that it carries the design name and the extension PGF. If XNFMAP cannot open the guide file, it cannot guide the mapping.

```
Warning 426. There are n guide symbols in the
Guide file that will be deleted because their
output signals do not exist in the network of the
Design file.
```

The following Guide file symbols are ignored. A list follows showing which guide file CLBMAPs are disregarded. See the "Using a Partitioning Guide File" section for more information about guide file rules.

```
Warning 427. There are n guide symbols in the
Guide file that have input signals that do not
exist in the design file.
```

XNFMAP cannot implement these guide symbols. A list follows showing which guide file CLBMAPs might not be obeyed. See the "Using a Partitioning Guide File" section for more information about guide file rules.

```
Warning 428. There are n open CLBMAP symbols in
the Design file that will be deleted because the
Guide file has closed guide symbols that specific
the same logic.
```

The following guide file symbols take precedence over the design file symbols. A list follows showing which design file CLBMAPs are being superseded. See the "Using a Partitioning Guide File" section for more information about guide file rules.

```
Warning 429. There are n guide symbols in the
Guide file that will be deleted because the
Design file has guide symbols that specify the
same logic.
```

The following design file guide symbols take precedence over the guide file symbols. A list follows showing which design files CLBMAPs take precedence. All design file CLBMAPs can be ignored by specifying the -m option. See the "Using a Partitioning Guide File" section for more information about guide file rules.

```
Warning 430. No Guide symbols found in Guide file.
```

Run LCA2XNF with no options to create the *design*.lca file and then rename it with a PGF extension.

```
Warning 431. No new CLBs were mapped (All
previously mapped?) or (All logic removed?).
```

XNFMAP does not generate a guide file because it did not partition any CLBs. This might occur if XNFPrep removed all logic in the design. Check the CRF and the XNFPrep output.

```
Warning 432. There are guide symbols in the Guide
file that will be deleted because the Design file
has mapped CLB symbols that specify the same
logic.
```

The following guide file symbols are deleted. A list follows showing the guide file CLBMAPs that are superseded. See the "Using a Partitioning Guide File" section for more information about guide file rules.

```
Warning 440. Underlying DFF location
specification will be used instead of location
specified in Guide file.
```

If a flip-flop carries a location specification that conflicts with the guide-file location, XNFMAP uses the flip-flop location. All design file CLB location constraints can be ignored by specifying the -n option. See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about location control.

```
Warning 450. Will not use filename with Guide file.
```

If you specified the -k option and XNFMAP finds alias names ($-number) in the guide file, it searches for an AKA file with the input

design name. Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file. If XNFMAP finds no AKA file, it might not be able to guide partitioning effectively.

```
Warning 451. Signal name signalname on guide symbol
will not be restored to original name.
```

XNFMAP finds a name in the guide file that it identifies as an alias name, a name that begins with $-*number*, but cannot find that prefix in the AKA file. Check that XNFMAP read the correct AKA file.

If any user net names contain dollar sign ($) characters, the unrestored names might be original user names; in this case the message can be ignored.

```
Warning 455. EQN symbol symbolname has missing or
invalid equation.
```

XNFMAP removes the symbol and continues processing.

```
Warning 456. Cannot match equation variable variable
with any pin on the EQN symbol symbolname.
Variable variable will be trimmed from the equation.
```

XNFMAP finds an inconsistency between the EQN symbol's equation string, and the pins and related signal names of the symbol. XNFMAP removes an unmatched equation variable from the equation.

Check the logic removal section of the report file from XNFPrep. Check the specifications of the EQN symbol in the design.

```
Warning 457. pinname pin on EQN symbol symbolname is
not used in the equation, and will be removed.
```

XNFMAP removed the unused EQN symbol pin from the symbol.

Check the logic removal section of the report file from XNFPrep. Check the specifications of the EQN symbol in the design.

```
Warning 458. Cannot match equation variable variable
for EQN symbol symbolname with any CLB pin signal.
Variable variable will be trimmed from the symbol.
```

Check the logic removal section of the report file from XNFPrep. Check the specifications of the EQN symbol in the design.

`Warning 459. Equation for EQN symbol` *symbolname* `reduces to a null equation. Symbol will be removed.`

Check the logic removal section of the report file from XNFPrep. Check the specifications of the EQN symbol in the design.

`Warning 460. Equation for EQN symbol` *name* `reduces to a [GND | VCC] value. Symbol will be removed, and` *signal names* `signal will be set to [GND | VCC].`

Check the logic removal section of the report file from XNFPrep. Check the specifications of the EQN symbol in the design.

`Warning 461. Removal of` *pinname* `pin from EQN symbol` *symbolname* `results in a null equation. Symbol will be removed.`

Check the logic removal section of the report file from XNFPrep. Check the specifications of the EQN symbol in the design.

Error Messages and Recovery Techniques

XNFMAP issues the following error messages during processing. Some error messages below refer to guide symbols. A guide symbol is either a user-created CLBMAP symbol or a CLBMAP from the PGF, if you specified the -k option. The PGF uses CLBMAPs to direct mapping.

`Error 1. Field too long.`

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

`Error 2. Unexpected LCANET record.`

An LCANET record is found on any line except the first.

Re-generate the XNF and reprocess the design.

`Error 3. ENDMOD with no matching MODEL record.`

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

`Error 4. Symbol sub-record outside of symbol group. Record ignored.`

XTF syntax is violated. Records that should be within other records, such as MODEL, ENDMOD, or PIN, are found outside the group. Consequently, the XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

```
Error 5. Illegal record inside symbol group.
Record ignored.
```

XTF syntax is violated. A record that is not allowed inside other symbol groups is found within the group, for example, a PROG record inside a SYM group. Consequently, the XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

```
Error 6. Premature EOF record in symbol group.
```

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

```
Error 7. Illegal record inside MODEL group.
Record ignored.
```

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

```
Error 8. Premature EOF record in MODEL group.
```

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

```
Error 9. Premature End-of-file. No EOF record
found.
```

An EOF record must conclude every XTF file. The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

Error 10. Unknown record type *type*.

XNFMAP encountered an invalid XNF record type in an input file. Check for the corrupted input file.

Error 11. There are characters *char* at the end of a record line, after all expected data received.

Check for the corrupted input file.

```
Error 12. Invalid LCA netlist file. Invalid or
missing LCANET record.
```

The XTF file might be invalid or corrupted. Every valid XTF file must start with the LCANET record that lists the XNF version of the file.

Re-generate the XTF file.

`Error 13. Unsupported XNF netlist version` *number.*

XNFMAP does not support the version number listed in the LCANET record. This program only supports XNF versions 1, 2, 4, and 5.

Re-generate the XTF file.

`Error 15. Valid part type must be specified before netlist symbols can be read.`

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

Re-generate the XTF file.

`Error 16. Invalid part record, missing part type.`

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

`Error 17. Missing name on SYM record. SYM record ignored.`

`At line` *number*`: Missing type on SYM record. SYM record ignored.`

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

`At line` *number*`: Unknown symbol` *symboltype*`. SYM record ignored.`

`Error 18. Invalid PIN record. Missing name field.`

`At line` *number*`: Invalid PIN record. Missing or invalid direction field.`

`At line` *number*`: Invalid PIN record. Invalid direction field.`

`At line` *number*`: Invalid PIN record. Non-numeric delay field.`

The XTF file might be invalid or corrupted. Rerun the program that

created this XTF file and try again.

`Error 19. Pin name` *pinname* `used multiple times on symbol` *symbolname*`.`

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

`Error 20. Missing command on CFG record.`

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

`Error 21. CFG records allowed only in CLB and IOB symbols.`

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

`Error 22. Invalid SIG record. Missing signal name.`

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

`Error 23. Invalid EXT record. Invalid direction` *dir*`.`

EXT records must have a direction field of I, O, T, B, or U.

`At line` *number*`: Invalid EXT record. Missing signal name. EXT record ignored.`

`At line` *number*`: Invalid EXT record. Bad or missing direction field. EXT record ignored.`

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

`Error 24. Invalid BUS record. Missing bus name.`

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

```
Error 25. Invalid PULSE record. Missing pin name
field.

At line number: Invalid PULSE record. Missing or
invalid polarity field.

Invalid PULSE record. Invalid polarity field.

At line number: Invalid PULSE record. Invalid or
missing minimum width field.
```

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

```
Error 26. Invalid PWR record. Bad or missing
polarity field.
```

PWR record must have a 1 or 0 in the polarity field. Re-generate the design from the schematic. Check for the corrupted XTF file.

```
Invalid PWR record. Polarity is polarity. Must be
0 or 1.
```

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

```
Error 27. Invalid SETUP record. Missing pin name
field.

At line number: Invalid SETUP record. Missing
clock pin name field.

At line number: Invalid SETUP record. Missing or
invalid clock edge field.

At line number: Invalid SETUP record. Invalid
clock edge field.

At line number: Invalid SETUP record. Missing or
invalid setup time.

At line number: Invalid SETUP record. Missing or
invalid hold time.
```

The XTF file might be invalid or corrupted. Rerun the program that created this XTF file and try again.

`Error 28. At line` *number*`: Missing id on HIERG record.`

`At line` *number*`: Missing type on HIERG record. HIERG record ignored.`

`At line` *number*`: Missing name on HIERG record. HIERG record ignored.`

`At line` *number*`: Missing filename on HIERG record. HIERG record ignored.`

`At line` *number*`: Missing parent id on HIERG record. HIERG record ignored.`

The XTF file might be invalid or corrupted. Return to the program that created this XTF file and try again.

`Error 30. For general use of invalid parameters`

`At line` *number*`: Unknown MAP symbol` *symboltype* `for SYM` *symbolname*`. Will use default MAP type` *maptype*`.`

A CLBMAP symbol has a MAP parameter value that is not PUC, PUO, PLC, or PLO. See the section "Partitioning Logic on a Schematic" for more information about MAP parameters. XNFMAP defaults to MAP=PUC.

typename `has invalid parameter value` *parameter*`.`

A floating point value of the THI and TLO parameters have trailing characters that are not "ns" (for nanoseconds). Only add the suffix "ns" to a THI or TLO value.

`At line` *number*`: Unknown SYM record parameter` *parameter*`.`

An unknown XNF symbol parameter is found. Check the parameter specification in the design file.

`On line` *number*`: Invalid parameter found on SYM` *symbolname*`, type` *symboltlype*`.`

A parameter is assigned to a symbol that does not support that parameter, such as a fast tag on a DFF symbol. Check the parameter for that symbol.

`On line` *number*`: Unknown PIN record parameter` *parameter*`.`

An invalid parameter is assigned to a PIN. Check the parameter for that symbol pin.

`On line` *number*`: PIN record parameter` *parameter* `found on MAP symbol.`

An invalid parameter is assigned to a PIN statement for a CLBMAP. Only a P (pin-lock) parameter is allowed on CLBMAP symbol pins.

Correct or remove the parameter from the pin on the symbol.

`On line` *number*`: Unknown SIG record parameter` *parameter*`.`

An invalid parameter is assigned to a SIG record. Check the parameter for that signal.

`At line` *number*`: Unknown EXT record parameter` *parameter*`.`

An invalid parameter is assigned to a EXT record. Check the parameter for that I/O pad.

`At line` *number*`: FILE parameter found on` *symbolname* `SYM, type` *symboltype*`. (Non-flattened design?)`

A FILE= parameter is found on a non-macro symbol. FILE= parameters should be added only to unflattened macro symbols. This warning might occur if a reserved name, such as AND or OR, is used as the name of a FILE= macro. Change the macro name and re-run XNFMAP.

`At line` *number*`: Extra LOC parameter` *parameter* `found on signal` *signalname*`.`

More than one LOC parameter is found on a record. If commas (,) are used to separate LOC parameters, XNFMAP issues this warning. Separate multiple-block LOC parameters by semicolon (;) characters.

`At line` *number*`: Invalid MAP symbol type [PLO, PUO or PLC] for HMAP symbol` *symbolname*`.` *PUC* `is the only valid MAP= parameter for an HMAP symbol.`

`At line` *number*`:` *parameter* `is not complete on PIN` *pinname*`.`

An invalid parameter exists on the PIN record. Correct the attribute on the symbol pin.

```
At line number: SYM record parameter TNM not
allowed on timespec symbols.
```

Use the TNM= parameter on a flip-flop, RAM, I/O latch, and pad symbol.

```
At line number: SYM record parameter parameter only
allowed on timespec symbols.
```

You can only place TS*id* parameters for timing specifications on TIMESPEC symbols.

```
Error 31. Invalid LOC parameter parameter on
symbol-type name.
```

Correct the use of the parameter in the design.

```
Error 33. More than one parameter is specified on
symbol symbolname. Only one is allowed per symbol.
```

There is only one each of the following parameters allowed on a single symbol: RLOC, RLOC_ORIGIN or RLOC_RANGE.

Correct the use of the parameter in the design.

```
Error 34. Invalid RLOC parameter parameter on
symbol-type name.
```

Correct the use of the parameter in the design.

```
Error 35. Both TTL and CMOS parameters are
specified on symbol symbolname. Only one of these
parameters is allowed per symbol.
```

Remove one of the parameters from the symbol.

```
Error 36. RLOC specified on symbol symbolname of
type symboltlype. RLOCs may not be used with
decoders, clocks, logic symbols or IO primitives.
```

Remove the RLOC parameter from the symbol.

```
Error 37. FAST/SLOW/MEDFAST/MEDSLOW and FAST/
SLOW/MEDFAST/MEDSLOW parameters have been found
on symbol/ext sym name/ext name. These parameters
should be mutually exclusive.
```

Remove all but one of these parameters from the symbol.

`Error 38. TNM parameter` *TNM = tnm value* `on symbol` *symbolname* `has illegal type. The only legal types are FFS, RAMS, PADS, or LATCHES. The syntax should be TNM=`*name_value* `or TNM=`*type*`:`*name*`.`

Change the specification of the TNM parameter.

`Error 40. At line` *number*`: Two different IOBs use the same signal` *signalname*`.`

Two IOBs cannot use the same pad signal. Every pad signal must have a unique name. This error might occur if you have a pad signal connecting an input I/O symbol and an output I/O symbol, and the two symbols are in separate modules of the design. If the two modules are mapped separately (map-then-merge design processing), then the two symbols are made into two separate IOBs that both use the same pad signal.

Although you might have intended to make the pad signal a bidirectional signal, the separation of the two symbols into different modules causes this error. It is advisable to keep all the I/O elements of a single IOB (pad signal, input and output symbols) in the same module/hierarchy of the design.

`Error 53. Out of memory. Needed` *number* `bytes.`

There is insufficient memory to complete the XTF file processing. Check the memory requirements for the FPGA part type in use.

`Error 220. CLB primitive symbols must be translated into common logic symbols by XNFPREP before XNFMAP is run.`

XNFMAP does not accept CLB primitive symbols directly. XNFPrep is responsible for translating these primitives into EQN symbols and flip flops that can be trimmed of unused logic if necessary. XNFPrep also generates a CLBMAP symbol to direct XNFMAP to re-create the original CLB for the newly generated logic.

Run XNFPrep before using XNFMAP.

`Error 221. IOB primitive symbols must be translated into common logic symbols by XNFPREP before XNFMAP is run.`

XNFMAP does not accept IOB primitive symbols directly. XNFPrep is responsible for translating these primitives into common IO gates

that can be trimmed of unused logic if necessary. The connectivity of the newly generated IO gates leads to the restoration of the IOB in XNFMAP.

Run XNFPrep before using XNFMAP.

```
Error 225. Symbol defines an oscillator, but none
is available.
```

More than one oscillator was specified. Only one crystal oscillator circuit is available on XC2000 or XC3000 devices.

```
Error 226. Symbol defines an unavailable clock
buffer.
```

Only the GCLK and ACLK buffers are available on XC2000 and XC3000 devices.

Correct the design to use only clock buffers available in the part.

```
Error 227. Symbol defines a [GCLK|ACLK] clock
buffer which is already assigned to symbol name It
is an error to have multiple [GCLK|ACLK]
symbols.
```

One of the global clock buffers was used twice. Only one GCLK buffer and one ACLK buffer are available on XC2000 or XC3000 devices. Remove one of the extra clock buffers.

```
Error 228. The presence of MODEL records in the
input design file indicates that the design has
been previously mapped. This version of XNFMAP
requires unmapped input design files. Regenerate
the input design files from your schematics or
other methods of design entry.
```

Re-run XNFMAP with the -a and -q options if you are interested in "map-then-merge" techniques to control the mapping of designs on hierarchy levels.

```
Error 229. IOB needed for Oscillator but already
used.
```

The crystal oscillator requires two dedicated pins, XTAL1 and XTAL2. If the oscillator is used, the IOBs associated with these pins cannot be used as a normal I/O. Edit the design so it does not use

these pins. Check the pin descriptions in *The Programmable Gate Array Data Book* to find the locations of XTAL1 and XTAL2.

`Error 230. Unable to find data file` *filename*`.`

XNFMAP could not find the specified data file. Make sure that the indicated file exists in the \XACT or \XACT\DATA directory. If it does not exist, re-install the implementation software.

If the file does exist, use the DOS SET command to verify that the XACT environment variable points to the \XACT directory. Any spaces in the SET XACT= statement can prevent the data files from being found.

On workstations, set the XACT environment variable to point to the directory that has a "data" subdirectory with the necessary data files.

`Error 231. Data file` *filename* `is invalid. Unknown family` *familyname*`.`

Check that you specified a valid part type. If the part type is valid, the indicated data file might be corrupted. Re-install the implementation software.

```
Error 232. Insufficient information in partlist
file.
```

The file partlist.xct is out-of-date and does not contain all of the information necessary for XNFMAP to process the design.

Install a new or corrected partlist file.

`Error 234. Part type` *parttype* `not found in data file` *filename*`.`

Check that you specified a valid part type. A table of valid part types is listed after this message. If the part type is valid, the indicated data file might be corrupted. Re-install the implementation software.

```
Error 237. Unable to open temporary work file.
```

Check for a disk-full condition or incorrect file permissions. On a PC-based system, the DOS files variable may be too low; use at least FILES=20 in the config.sys file.

```
Error 239. Unable to rename temp file.
```

XNFMAP failed when attempting to rename its work file to the actual output name. This might occur if the target file already exists and is

flagged read-only, or a disk-full condition exists. Check for these two conditions and run XNFMAP again.

```
Error 240. Re-setting classification for CLBMAP
name to MAP=PUO due to lack of function
generator input pins on the symbol.
```

If a CLBMAP symbol does not have any signals connected to the A-E function generator input pins, then XNFMAP resets the MAP parameter to be "open" to more logic.

Specify signal names on the CLBMAP A-E pins if you want to use a "closed" CLBMAP.

```
Error 242. Unable to open LCA netlist file for
reading.
```

XNFMAP did not find the specified input file in the current directory. Check the directory and file name and run XNFMAP again.

```
Error 243. Unable to open report file for writing.
```

Check for a disk-full condition or incorrect file permissions. On a PC-based system, the DOS files variable may be too low; use at least FILES=20 in the config.sys file.

```
Error 244. Error writing report file to disk.
```

Check for a disk-full condition.

```
Error 245. Error while writing information to
disk. Some information can be found in file
filename. Check for a disk-full condition.
```

Clear some disk space and run XNFMAP again.

```
Error 247. Multiple source pins on signal. (TBUFs
disabled or removed?).
```

Check the schematic for inadvertently connected signals. Check signal naming, since signals with the same name are considered connected. If error occurs on a line sourced by TBUFs, check if XNFMAP or XNFPrep disabled or removed any of those TBUFs.

```
Error 248. Power/ground signal has a source.
```

A signal flagged as V_{CC} or GND is sourced by logic. If the signal was intended to be a power or ground connection, remove the source

connection. If the signal was intended to be a normal signal, it cannot be flagged as V_{CC} or GND.

`Error 249. Invalid pin name` *var* `on symbol` *symbolname* `type` *symboltype*`.`

XNFMAP finds a symbol it recognizes but does not recognize the pin name. If the indicated symbol is an unflattened macro, it must not use a reserved symbol name, such as AND or OR. If the indicated symbol is a primitive, check with the company that supplied the XNF interface.

`Error 251. Pin on symbol` *symboltlype* `is not invertible.`

XNFMAP finds an inverted signal on a symbol pin that cannot be inverted. Inverted pins are legal only on certain primitive elements. If the indicated pin has been changed to inverted sense, restore it to normal sense and use a discrete inverter.

`Error 252. Logic symbol has` *n* `inputs (max=<m>).`

For a Boolean logic gate, the maximum number of input pins is four for the XC2000 family and five for the XC3000 family. This limit represents the widest function that can be implemented in a CLB. Break the gate into the properly-sized functions.

`Error 257. Invalid CLB location` *loc* `specified on symbol` *symbolname*`.`

XNFMAP ignores an invalid CLB location. See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about specifying locations.

`Error 258. Symbol` *symbolname* `has CLB location` *location*`, which is already used.`

The CLB location specified is used by another block. See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about specifying locations.

`Error 264. External signal` *signalname* `defines IO block` *blockname* `which is already used.`

The IOB location specified is used by another pin. See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about specifying locations.

`Error 265. Unknown IO block name` *name* `on external signal` *signalname*`.`

The IOB location specified for the indicated pin is not valid. IOB locations are expressed with pin numbers (such as P13 or P23) or with PGA package pin designations (such as A5 or D4). Consult the pin descriptions in *The Programmable Gate Array Data Book* for the valid pin designations.

`Error 269. Missing part type for -p option.`

The -p option flag must be followed by a valid FPGA part type. Consult *The Programmable Gate Array Data Book* for valid part types.

`Error 270. Unknown flag ignored.`

Check use of command-line options. See the "Options" section of this chapter for the legal option flags.

`Error 271. Extra command-line argument.`

Check use of command-line options. See the "Options" section in this chapter for the legal option flags.

`Error 272. Option` *-name* `cannot be used with another hierarchy mapping option. Choose only one mapping option between -a, -q, and -u.`

The -a option separately maps the logic in each user-defined macro. Use of this option prevents logic from different levels of hierarchy being combined in one CLB.

The -q option separately maps the logic in each macro that was flagged with a "FILE=" parameter.

The -u option ignores the hierarchy defined in the input file and to use the hierarchy defined in the guide file.

Re-run XNFMAP again with one of the options.

`Error 282. External signal flagged as power/gnd`

A signal that connects to an I/O pad has been flagged as power or ground. An I/O pad cannot be connected directly to a power signal. If an output pin is to drive V_{CC} or GND, flag the input of the OBUF appropriately.

```
Error 283. Non-I/O symbol connected to external
signal.
```

An external signal, one that connects to an I/O pad, is connected to a non-I/O symbol (an I/O buffer or register). Only I/O symbols, such as IBUF, OBUF or INFF, can be connected to I/O pads. Correct the connection of symbols/signals in the design.

```
Error 284. Non-external pin of I/O symbol
connected to external signal.
```

An external signal, one that connects to an I/O pad, is connected improperly to an I/O symbol (an I/O buffer or register). Each I/O symbol has one external pin that may be connected to an I/O pad. Examples are the O pin of an OBUF, the I pin of an IBUF, the Q pin of an OUTFF, or the D pin of an INFF. If a non-external pin is connected to an I/O pad, XNFMAP issues this warning.

Correct the connection of symbols/signals in the design.

```
Error 285. Multiple load or source pins on
external signal.
```

An external signal, one that connects to an I/O pad, might be connected to at most one input element and one output element. A single I/O pad cannot be sourced by both an OBUF and an OUTFF.

Correct the connection of symbols/signals in the design.

```
Error 287. Signal flagged as both VCC and GND.
```

The indicated signal has been flagged as both V_{CC} and GND. Delete one of the power attributes.

Error 291. Unknown location specified on *symbolname*.

The location specified for the indicated symbol is not valid. See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about specifying locations.

Error 293. *typename* symbol *symbolname* specifies location *loc,* which is already used by symbol *symbolname*.

Two symbols specify the same location. See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about specifying locations.

`Error 295.` *typename* `symbol` *symbolname* `and` *typename* `symbol` *symbolname* `drive signal` *signalname*`, but their locations are on different longlines.`

Correct the LOC constraints on your schematic.

`Error 296.` *typename* `symbol` *symbolname* `on signal` *signalname* `specifies location` *loc* `which is already used for another three-state signal.`

A single horizontal longline can carry only one 3-state signal. You cannot specify symbols, normally TBUFs and pull-ups, driving two different 3-state signals on the same longline. Change the location specifications to use different longlines.

`Error 297. Illegal connection to global reset signal` *signalname*`. Illegal symbol names listed.`

The dedicated global reset signal is controlled by the Reset pin on the FPGA device. This fixed net connects to the asynchronous reset of every CLB flip-flop and IOB flip-flop or latch. A signal that is connected to the global reset pin of a flip-flop or latch is defined to be the global reset signal, and cannot be connected to a non-global reset pin. Remove the global reset signal from the indicated symbol pins.

`Error 298. Multiple global reset signals` *signalname* `and` *signalname*`.`

The reset pin on the FPGA device controls the dedicated global reset signal. This fixed net connects to the asynchronous reset of every CLB flip-flop and IOB flip-flop or latch. A signal connected to the global reset pin of a flip-flop or latch is considered the global reset signal. Because there is only one global reset net on the device, only one net can be connected to global reset pins. Remove one of the indicated signals.

`Error 299. Illegal location` *loc* `specified on direct clock input signal` *signalname*`. Using correct location` *loc* `for the direct clock input.`

Correct the LOC constraints on your schematic.

Error 300. Pull-ups and [*OBUF*|*OBUFZ*|*OUTFF*] both used on external signal.

The pull-up resistors in XC3000 IOBs can only be activated if the IOB is used as input only. If an output buffer is used, the pull-ups in that IOB cannot be used. Remove the pull-ups from the I/O pin.

Error 305. Cannot put DFF *symbolname* and DFF *symbolname* into same CLB.

XNFMAP finds that two flip-flops cannot be combined into the same CLB, but location specifications or a CLBMAP (or guide file) is attempting to force them. Correct the LOC parameters or CLBMAP.

Error 306. Both latches/DFFs in block must use Q pins.

Check that incompatible symbols are not assigned to the same CLB by a CLBMAP or location specifications.

Error 307. DFF *symbolname* and DFF *symbolname* that are assigned to the same block, must use identical clock, reset, and enable signals.

The specified symbols are not paired in the same CLB. Change the CLBMAP, LOC, BLKNM, or HBLKNM parameters that are trying to force the flip-flops together.

Error 308. Too many output pins required by logic related to DFF *symbolname* and DFF *symbolname* than can fit in one block.

The logic specified for one CLB by a location or CLBMAP symbol requires more outputs than are available on one CLB. Check the indicated logic.

Error 309. Mapped output signal *signalname* has no source symbol.

A CLB with missing input signals cannot be mapped properly. Check any CLBMAPs relating to the indicated signal.

Error 310. Location *loc* for mapped CLB *symbolname* already used.

The specified location is ignored. Check the LOC parameters for that CLB.

```
Error 311. Location loc for mapped CLB symbolname
not found.
```

A location specification is attempting to place a CLB in an invalid location. Check the LOC parameters for that CLB against the legal locations for the part type in use.

```
Error 312. Cannot map guide symbol as directed.
```

Refer to error messages listed above. XNFMAP issues one or more error messages before this one to indicate a problem with a CLBMAP or guide file symbol. Check the preceding messages to determine the source of the error. XNFMAP ignores the CLBMAP or guide file symbol and partitions the related logic using the normal algorithms.

```
Error 313. DFF has no D pin.
```

You cannot include the symbol in the mapped CLB. Check that the CLBMAP for the indicated flip-flop has been defined properly.

```
Error 314. Specified two latches/DFFs for guide
symbol symbolname. Cannot map these two symbols
into one CLB. Check specification of guide
symbol.
```

A CLBMAP places more than two latches in an XC3000 CLB or more than one latch in an XC2000 CLB. XNFMAP cannot map these symbols into one CLB. Check that the data pin for these latches does not require another latch to be included in the CLB. This error message is preceded by another error specifying the symbols that cannot be mapped.

```
Error 315. While searching logic path that
sources symbol, encountered a symbol that cannot
be mapped into a CLB.
```

Check specification of CLBMAP for logic that cannot be mapped into a CLB. All logic symbols to be partitioned into a CLB must be combinatorial logic, latches, or flip-flops.

```
Error 316. Too many logic levels to complete
mapping control process for CLB with [signal |
symbol].
```

Check that closed CLBMAP inputs are completely defined. Attempts to partition too many combinatorial gates into one CLB can also

generate this error. Check that the indicated logic can be legally partitioned into a CLB.

```
Error 317. DIN signal not being used correctly as
a direct input to a latch/DFF.
```

XNFMAP finds that the DI signal sources other than data pins. A signal on the DI pin of a CLBMAP must go directly to the D input of a CLB flip-flop or XC2000 latch.

```
Error 318. Tried to map logic associated with
signal into a CLB that already has 2 functions
associated with signalname and signalname.
```

A CLBMAP symbol attempts to partition more logic into a CLB than fits. Check the CLBMAP specification.

```
Error 319. Too many inputs required for CLB
function that includes signal signalname.
```

A CLBMAP symbol attempts to use too many inputs to a function generator, or an illegal combination of inputs. Check the legal input configurations for the CLB function generator and correct the CLBMAP accordingly.

```
Error 320. Number of inputs require more
functions than available.
```

The logic defined by a CLBMAP symbol cannot be implemented in two function generators (that is, in one CLB). Correct the CLBMAP to define a legal CLB.

```
Error 321. Input signal not specified on mapped
CLB.
```

A signal internal to an F or G function has been flagged with an X (explicit) attribute. Check that you correctly specified all inputs to the CLBMAP.

```
Error 322. Location specified for symbol already
assigned to a block.
```

Another symbol already uses a location specified on a flip-flop, CLBMAP, or guide file. Check for conflicts between CLBMAP, guide file, and flip-flop locations.

`Error 325. Tried to map [`*F*`|`*G*`] function with signal into a CLB that needs the [`*F*`|`*G*`] function for` *symbol.*

The F or G function generator is needed to generate more than one internal signal. Check the CLBMAP specification and remove one of these functions.

`Error 326. Cannot map [`*GUIDE file*`|`*guide*`] symbol` *symname* `as directed. Conflicting hierarchy information in guide and design files.`

Do not use the -a or -q options, or use a the -u option to allow the guide file hierarchy to override the design file hierarchy.

`Error 329. Can't use Q and D pin signals in same function.`

An XC2000 function generator can have either D or Q as an input but not both. Correct the CLBMAP to use one or the other.

`Error 331. Cannot use [`*C pin* `|` *G function* `|` *K pin*`] for clock signal in guide symbol` *symname*`. Ignoring directive to use [`*C pin* `|` *G function* `|` *K pin*`]..`

Check the specification of the guide symbol.

`Error 340.` *symbol* `uses GCLK buffer output signal improperly.`

On an XC2000 device, the GCLK output can only connect to the B or K pins on the CLB. On an XC3000 device, the GCLK output can only connect to CLB and IOB clock pins (K, IK, and OK).

Correct the use of the clock output signal.

`Error 341. Cannot fit function group(s) in CLB.`

Check that the CLBMAP does not attempt to use an invalid CLB configuration. Check the legal CLB function generator options and correct the CLBMAP.

`Error 342. Clock signal` *signalname* `has been directed to use the G function on the CLB. However, the G function is reserved for the reset signal. The clock signal should be assigned to the K or C pin instead.`

A CLBMAP attempts to use the G function for both. In an XC2000 CLB, if the G function generator is used as the reset signal; it cannot be used as the clock signal.

Flag the clock signal to use the K or C pins instead with the K or C signal parameters.

`Error 343. Clock signal` *signalname* `has been directed to use the C pin on the CLB. But the signal is from the GCLK buffer, which can only be routed to the K or B pins. The signal should be assigned to the K pin instead of the C pin.`

An XC2000 CLBMAP indicates that the C pin should be used for the clock signal. The output of the GCLK buffer can only be routed to the CLB K or B pins. Correct the CLBMAP accordingly.

`Error 344. Equation building buffer for CLB` *names* `overflowed. Cannot complete process.`

Use the -g option to limit the number of gates that can be included in an equation. See "Options" for more information.

`Error 345. Clock signal on guide symbol cannot be combined with clock signal on DFF.`

The signal specified on the CLBMAP K pin does not match the clock signal on the corresponding flip-flop or latch. If no signal is indicated on the CLBMAP K pin, XNFMAP chooses the clock signal.

Remove the incorrect signal from the CLBMAP clock pin.

`Error 347.` *symbol* `has been mapped into a CLB and its output signal is needed by another CLB or IOB.`

However, the guide symbol that guided the mapping of this CLB did not specify this signal as an output. The indicated signal must be assigned to an output pin of the CLBMAP or it is not available outside the CLB. Change the CLBMAP so that the signal is attached to the X or Y pins.

`Error 349. Too many input pins required by logic that sources DFF` *symbolname* `and DFF` *symbolname*`.`

Two flip-flops or latches were assigned the same BLKNM parameter but cannot be combined into one CLB because the logic gates that

source them require too many inputs. If flip-flops must be combined in one CLB, use a CLBMAP to carefully group the logic.

`Error 350. Too many variables per function or too many functions specified for guide symbol.`

The logic defined by the CLBMAP does not fit into one CLB. Change the CLBMAP definition so that the logic defined by the input and output signals can be placed into one CLB.

`Error 351. Guide symbol tried to map` *name* `into a new CLB, but it has already been assigned.`

Two CLBMAPs attempt to map the same symbol. Change one of the CLBMAPs to exclude that function.

`Error 353. Reserved system name - can't use name` *name*`.`

The indicated name is reserved by XNFMAP and cannot be used as a user name. Change the name to a non-reserved one.

`Error 354. Name` *name* `specified more than once, but will be used only once.`

Symbol names in a design must be unique. Correct naming errors in design.

`Error 355. Illegal characters in name` *name*`. Will not use name` *name*`.`

The indicated name is not a valid FPGA name, and is ignored. FPGA names can contain only letters, digits, underscores (_), dashes (-), dollar signs ($), and angle brackets (< and >). Names must contain at least one non-digit character and no spaces.

`Error 360. Invalid equation for Block:` *symbolname: equation string*`.`

Correct the indicated CLB primitive symbol.

`Error 361. Unmatched parenthesis in equation for block` *blockname*`.`

The number of "(" characters does not match the number of ")" characters. Correct the equation for the indicated CLB primitive symbol.

```
Error 362. Illegal pin name in equation for CLB.
```

An equation for the CLB primitive must be expressed in terms of CLB pins only. Correct the equation for the specified CLB primitive symbol.

```
Error 364. Guide symbol output signal is sourced
by symbol that cannot be mapped into a CLB.
```

The source of a CLBMAP output signal does not fit into the CLB. Correct the CLBMAP assignments.

```
Error 370. Multiple [GCLK|ACLK|OSC] symbols in
design.
```

Each XC2000 or XC3000 device has only one GCLK buffer, one ACLK buffer, and one on-chip oscillator. Remove any duplicate symbols from the design.

```
Error 372. Could not fit in a CLB the logic
indicated by guide symbol.
```

Use the X signal flag or a fully specified (closed) guide symbol to better define the logic. An illegal feedback path is indicated by assigning a function-generator output to a CLBMAP input pin for the same CLB. Only the output of the flip-flops can be fed back to the function generator internally. If you want to feed back the function-generator output, it must be done external to the CLB; therefore, assign the function to a CLBMAP output pin as well.

```
Error 373. Cannot lock signals on input pins to a
CLBMAP symbol; with a 'PLO' or 'PUO' type.
Ignoring input pin locking for CLBMAP name.
```

If a CLBMAP is indicated as open (MAP=PLO or MAP=PUO), you can only place P (pin-lock) attributes on the output pins. Remove P attributes from the CLBMAP inputs.

```
Error 374. Symbol symboltype symbolname has both HBLKNM
and BLKNM parameters. Only one of the two
parameters can be specified per symbol.
```

Remove one of the parameters from the symbol.

`Error 375. Two symbols,` *symboltype symbolname* `and` *symboltype symbolname* `are assigned to the same IOB and cannot have conflicting BLKNM values. Remove BLKNM parameter from one of the symbols.`

See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about location control.

`Error 376. Two symbols,` *symboltype symbolname* `and` *symboltype symbolname* `are assigned to the same IOB and cannot have both BLKNM and HBLKNM values. Remove the parameter from one of the symbols.`

See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about location control.

`Error 378. Illegal pin name` *pinname* `in equation for function generator` *name*`.`

Run XNFMAP again.

`Error 420. Unable to open temporary work file.`

Check for a disk-full condition or incorrect file permissions. On a PC-based system, the DOS-files variable might be too low; use at least FILES=20 in the config.sys file.

`Error 421. Unable to move existing PGF file` *filename1* `to backup file` *filename2*`.`

XNFMAP uses the existing guide file *filename1* and writes the new guide information to the file *filename2*.

Check for file write permissions on the named files.

`Error 422. Unable to rename temp file` *filename* `to` *filename*`.`

XNFMAP failed when attempting to rename its work file to the actual output file name. This might occur if the target file already exists and is flagged read-only, or a disk-full condition exists. Check for these two conditions and run XNFMAP again.

`Error 423. Error while writing Guide information to disk.`

Some information can be found in file *filename*. Check for a disk-full condition or incorrect file permissions.

```
Error 433. Missing source logic for output
signals of guide symbol.
```

The output signal indicated on a CLBMAP or guide file symbol has no source. Check that the outputs have been correctly specified on the CLBMAP.

`Error 434. Invalid LOC parameter` *loc* `on` *symboltype symbolname*`.`

An explanatory message might follow. The indicated symbol carries a location specification that is not valid. See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about location control.

`Error 435. Location parameters on the EXT signal` *signalname* `will take precedence over parameters of` *symboltype symbolname*`.`

If a location specification is attached to both an I/O pad and its corresponding I/O buffers, the location on the pad takes precedence. See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about location control.

`Error 436. LOC parameter on (`*signal or symbol*`) will be ignored - unsupported location-naming syntax.`

An invalid location was specified. See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about location control.

`Error 437. Merging signals` *signalname* `and` *signalname* `leads to conflict on LOC parameters.`

XNFMAP finds a conflict in location specifications when it merges two signals. Remove one of the location specifications. See the section "Using the BLKNM, HBLKNM, and LOC Parameters" for more information about location control.

`Error 438. The` *part* `device is not supported by this product The design cannot be mapped into the specified LCA part type because the part is not supported by this version of software.`

Contact a Xilinx sales representative for different development system options.

Error 439. Guide symbol in design file *symbolname* **location specification will override any location specified on underlying logic.**

If a location is specified on a CLBMAP symbol, the mapped logic cannot use individual location constraints. XNFMAP uses the CLBMAP location.

Error 441. Conflicting location specifications for DFF *symbolname* **and DFF** *symbolname*.

A CLBMAP indicates that two flip-flops should be paired, but those flip-flops have conflicting location specifications. XNFMAP ignores both location specifications. The CLBMAP takes precedence and the flip-flops are paired.

Error 442. Invalid package file *filename*.

The specified package file might be invalid or corrupted. Re-install the implementation software.

Error 443. Unable to open file partlist.xct.

The partlist.xct file could not be found. Make sure that this file exists in the \XACT or \XACT\DATA directory. If it does not exist, re-install the implementation software.

If the file does exist, use the DOS SET command to verify that the XACT environment variable points to the \XACT directory. Any spaces in the SET XACT= statement can prevent the data files from being found.

Error 444. Missing or invalid token *tokenname* **in file** *filename*.

An invalid or corrupted partlist.xct or speeds.xct file exists. Re-install the implementation software.

Error 445. Can't change *name* **to .dev suffix. This is an indication of corrupted or invalid data files.**

Re-install the implementation software.

Error 446. Unknown alias-to-part *parttype*.

Corrupted or invalid data files exist. Re-install the implementation software.

Error 447. Failed to find *parttype* in part list.

Corrupted or invalid data files exist. Re-install the implementation software.

Error 455. EQN symbol *symbolname* has missing or invalid equation.

A non-recoverable error (invalid equation syntax) occurred; therefore, XNFMAP stops the program.

Regenerate the input file.

Error 470. The input signal *signalname* on EQN symbol *symbolname* must either be a latch feedback signal or specified on an input pin to the GUIDE symbol.

The EQN symbol, when used in conjunction with a "closed" CLBMAP symbol, must fully specify a CLB equation. This means that all of the input signals to the EQN symbol must also be on the CLBMAP input pins or be a feedback signal from a latch in the proposed CLB.

Correct the CLBMAP symbol or EQN value.

Error 2001. Unable to open AKA file for reading.

Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file.

Error 2002. Illegal prefix in line *number*.

Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file.

Error 2003. Unable to add symbol to table.

Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file.

Error 2007. No symbol for index *number*.

Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file.

```
Error 2008. PREFIX table overflow on symbol.
```

Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file.

```
Error 2010. Unable to retrieve error message for
UI error code number.
```

Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file.

The MAP2LCA Program

This program is compatible with the following families.

- XC2000
- XC2000L
- XC3000
- XC3100

MAP2LCA translates a MAP file into a Logic Cell Array (LCA) file for use with the XACT*step* Development System software. MAP2LCA is the last step in the XNF-to-LCA translation process.

MAP2LCA outputs an LCA file that describes how the design is partitioned into a particular FPGA device. This file format is used by the XACT Design Editor (XDE) and the Automatic Place and Route program (APR). MAP2LCA produces a constraints file (SCP) that contains the placement and routing constraints specified in the schematic.

Syntax

The following syntax creates an LCA file from your MAP file.

`map2lca` [*options*] *design*[**`.map`**] [*output*[**`.lca`**]]

Files

This section describes the input file that MAP2LCA requires to generate an LCA file and the subsequent output files.

Input Files

design.map

This file contains logic-partitioning information and is the output of the XNFMAP program.

Output Files

design.lca

This file contains the design partitioned into the FPGA architecture and is used with the XACT*step* Development System Software (for example, XDE and APR).

design.scp

This file contains the placement and routing constraints specified in the schematic, which APR automatically reads.

Specifically, the SCP file includes all the constraints for APR contained in the schematic drawing. It does not include those additional constraints in the separate APR constraints file, the *design*.cst file. See The "APR" chapter in this reference guide for more information.

design.aka

This file contains the hierarchical path names for symbols and signals and their associated abbreviations.

Note: The -s option is not recommended. See the "Options" section for more information.

Options

MAP2LCA has the following options that you can use when creating the *design*.lca file.

–i Ignore Placement Constraints

This option directs MAP2LCA to ignore user placement constraints found in the MAP file.

–p Specifies the LCA Package and Part Type

This option allows you to override the part and package specification for the design. However, Xilinx does not recommend the use of this option at this stage in the design flow.

Note: If you want to modify the part type, change it in the schematic or synthesized input design, or use the -p option in XNFMerge. It is important that XNFPrep is run on the design with the correct part type prior to running MAP2LCA.

This option is only available from the command line.

–s Use AKA file to Shorten Names

Note: The current design flow does not include this option. Xilinx does not recommend the -s option for the reasons provided at the end of this section.

The -s option directs MAP2LCA to shorten the path names of signals and symbols and create an AKA file that lists each path name and its abbreviation. For example, a symbol named top/sub1/AND might be changed to '$1' to substitute for the path name top/sub1/. By default MAP2LCA creates abbreviations using the dollar sign '$' followed by one or more digits.

MAP2LCA reads and uses an existing AKA file before it generates a new file with default names. Therefore, it is possible for you to provide meaningful abbreviations for the path names. If an existing AKA file does not exist, run MAP2LCA with the -s option to create the file. You must edit the abbreviations in the right-most column in the file to be short, meaningful names for the identified paths in the left-most column. Be careful not to use the same abbreviations for different paths in the design. Then, run MAP2LCA again with the -s option to use the new abbreviations for the output LCA file. MAP2LCA retains your abbreviations when it creates the new AKA file.

The -s option is not recommended for the following two reasons.

- The back-annotation process that prepares a design for timing simulation depends upon maintaining original signal names throughout the design implementation process. If you change the signal names, the back annotation program (XNFBA) cannot successfully restore timing information to the original design.

- If you intend to derive a guide file from an LCA file for use with the XNFMAP -k option (guided design), then you must not edit the AKA file. You must keep the AKA file and make it available to XNFMAP so that it can restore the original signal names. Note, that XNFMAP does not recognize abbreviations that do not begin with a '$number'. Therefore, if you edited an AKA file to substitute the default abbreviations, XNFMAP cannot use that AKA file to help restore the original names in the guide file. See the "Guide by LCA File" section in the "XNFMAP" chapter.

MAP2LCA Example

Figure 4-1 shows a sample circuit and the partitioned result that is generated by MAP2LCA.

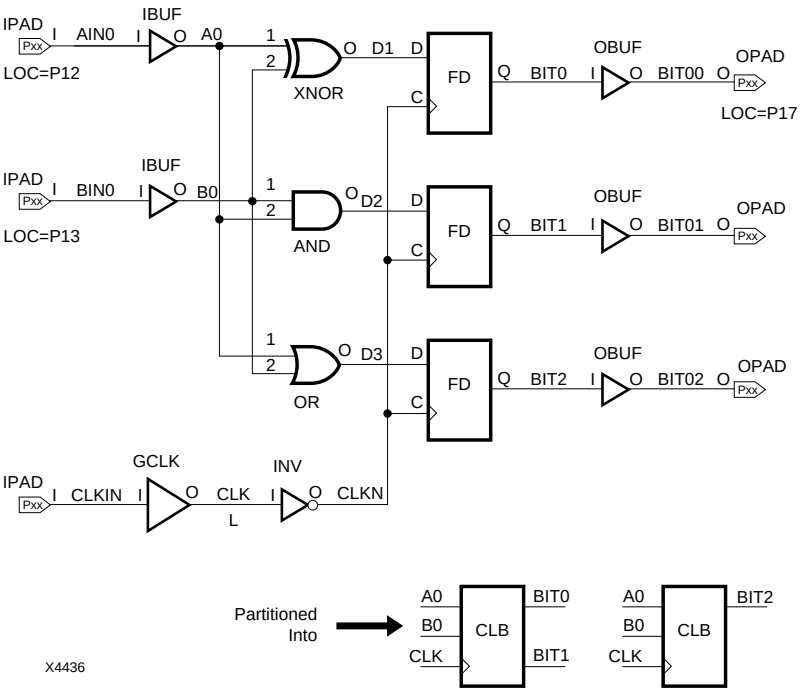


Figure 4-1 Sample Circuit

MAP2LCA translated the ADI1.MAP file into SAMPLE.LCA. Figure 4-2 shows the screen output of MAP2LCA using the following command line:

```
map2lca adi1 sample
```

```
A —  MAP2LCA Ver. 5.0
     (c) Copyright 1988 - 1994 Xilinx Inc. All rights reserved

     Checking network for logical errors...
B —  Assigning logic to LCA elements

     SUMMARY: Part type=3020pc68-100
              2 of 64 CLBs used
              6 of 58 I/O pins used
              0 of 6 internal IOBs used
C —           0 of 16 internal 3-state signals used
     Design written to file sample.lca (APR constraints in sample.scp
```

Figure 4-2 The Screen Output of MAP2LCA

The screen output lists any status messages and errors. A list of error messages, warnings, and recovery procedures is provided at the end of this chapter. If any error occurs, a message appears on the screen only.

If the translation aborts, perform the following steps.

1. Determine the nature of the error.
2. Correct your schematic design appropriately.
3. Translate the design into the MAP format.
4. Run MAP2LCA again.

The following letters correspond to the sections of the screen output shown in Figure 4-2.

A — Headers

This identifies the software version number, the date and time of the translation, and the copyright statement.

B — Status messages

Status messages about the translation display in this area.

C — Summary

Summary regarding the CLB-base implementation and the output files written.

The SCP constraints file generated for the adi1.map file is shown in Figure 4-3. The SCP file name is derived from the output file name, sample.lca.

Note: The letters on the left-hand side of Figure 4-3 correspond to the sections of the screen output listed below.

A — File header

Includes the version number and the date and time the file was produced.

B — Longline and critical net information

C — Block placement information

```
A —; Design: sample.lca, Created by MAP2LCA Ver 5.0 at Wed Jan 05 18:02:03 199
   ;(NOTE: Don't edit this file. It is rewritten each time MAP2LCA is run)
   flag net longline CLK ;
B —place block BIT00 P17 ;
   place block CLKIN P16
   place block BIN0 P13 ;
   place block AIN0 P12 ;
C —
```

Figure 4-3 Sample SCP File

design.aka

The AKA file gives the full hierarchical path names of all symbols and signals in the design and lists the abbreviations used in the LCA file. The circuit shown in Figure 4-1 has no hierarchy; therefore, the AKA file contains the usual header information, but no net names. An example circuit with hierarchical elements is shown in Figure 4-4. The associated AKA file is shown in Figure 4-5.

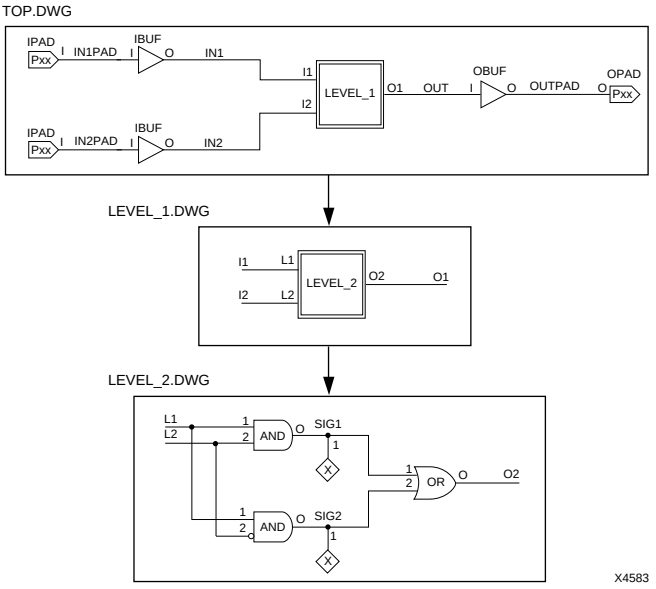


Figure 4-4 Sample Circuit with Hierarchy, top.dwg

```
A — # top.aka alias file created by MAP2LCA on wed Jan 06 12:55:02 1
B — $1    /CLKMUX/LEVEL_2
```

Figure 4-5 Sample top.aka File

Note: The AKA file is not read or produced unless you use the –s option. The letters on the left-hand side of Figure 4-5 correspond to the sections of the screen output listed below.

A — File header

Includes the file name, date, and time the file was generated.

B — Hierarchical path names and associated abbreviations

The AKA file for the top.dwg schematic is shown in Figure 4-5. The symbol and signals for level_2.dwg have hierarchical path names that begin with /CLKMUX/LEVEL_2. In the LCA file, this prefix is

shortened to $1. Therefore, the signal named SIG1 in the LEVEL_2.dwg file will be named $1/SIG1 in the LCA file.

Warning Messages and Recovery Techniques

This section describes the warning messages that MAP2LCA2 can generate. An explanation and workaround solution follows each warning message.

```
Warning 254. No CLBs used.
```

This message is issued if no CLBs were used when XNFMAP partitioned the design. This warning typically happens when all logic has been disabled or removed. If there are sourceless or loadless signals in the design that should not be removed, use an "S" (save) attribute to preserve them, or specify the estimate options in XNFPrep and XNFMAP.

```
Warning 255. No IOBs used.
```

This message is issued if zero I/O pins were used when the design was partitioned. If only a partial design is being processed and no I/O pins are used, you can ignore this message. Logic not sourced or loaded by IOBs can be preserved by using "S" (save) attributes, or by specifying the estimate options in XNFPrep and XNFMAP.

```
Warning 256. No I/O pins used (only internal IOBs
used).
```

This message is issued if only internal (unbonded) IOBs were used when the design was partitioned. If any bonded I/Os are indicated in the input design, check if any sourceless or loadless IOBs have been removed.

The INTERNAL parameter is used to specify that an I/O symbol does not need to be on a bonded pad. Check the design to ensure that all the I/O symbols do not have the INTERNAL attribute, or else none of the I/O symbols are placed on bonded pads and the FPGA will have no external inputs or outputs.

```
Warning 265. Configuration tag BASE, CONFIG on
IOB symbol symbolname ignored. Configuration tags
must begin with BASE or CONFIG.
```

Invalid IOB specification in MAP file. Regenerate the input MAP file using XNFMAP. Check for a disk-full condition that might have

caused an incomplete MAP file. The MAP file should always have an EOF on the last line.

`Warning 265. Configuration tag` *BASE, CONFIG, or EQUATE* `on CLB symbol` *symbolname* `ignored. Configuration tags must begin with` *BASE*, *CONFIG*, `or` *EQUATE*`.`

There is an invalid CLB specification in the MAP file. Regenerate the input MAP file using XNFMAP. Check for a disk-full condition that might have caused an incomplete MAP file. The MAP file should always have an EOF on the last line.

`Warning 266. No` *BASE, CONFIG,* `or` *EQUATE* `configuration command on CLB symbol` *symbolname*`.`

There is an invalid CLB specification in the MAP file. Regenerate the input MAP file using XNFMAP. Check for a disk-full condition that might have caused an incomplete MAP file. The MAP file should always have an EOF on the last line.

`Warning 267. Too many` *BASE, CONFIG,* `or` *EQUATE* `configuration commands on CLB symbol` *symbolname*`.`

There is an invalid CLB specification in the MAP file. Regenerate the input MAP file using XNFMAP. Check for a disk-full condition that might have caused an incomplete MAP file. The MAP file should always have an EOF on the last line.

`Warning 299. Illegal location` *location* `specified on direct clock input signal` *signalname*.

If the GCLK or ACLK buffer is connected directly to an IPAD symbol, the corresponding direct clock input is used. This message is issued if a location other than the direct clock input pin is specified for that IPAD. Consult the pin descriptions in *The Programmable Logic Data Book* to determine which pins are the dedicated clock inputs (TCLKIN and BCLKIN) for a particular part type. If no location is specified, the appropriate pin is chosen.

`Warning 448. Reserved name` *name* `will be ignored.`

Some names are reserved by the XACT*step* Development System and cannot be used as a user-specified block name.

`Warning 449. Illegal name` *name* `found. Name will be ignored.`

User-specified block name contains illegal characters or does not contain at least one non-numeric character. Change the name in the input design so that it is composed of alpha-numeric characters, and includes at least one alphabetic character.

`Warning 451.` *name* `is a reserved symbol name for an unbonded IOB in this package. Changing` *name* `to` *new_name*`.`

The names of unbonded package pins are reserved in the system. Typically, these names have the syntax of the letter 'U' followed by the unbonded pad number. MAP2LCA generates a new name to replace the reserved name.

`Warning 452.` *AKA_file* `found but will not be used. Specify the -s option if you want to use the` *AKA_file* `file to shorten signal and symbol names. However, this option is not recommended if you intend to run post-layout simulation because it changes the original signal names.`

In previous releases, MAP2LCA would read and use the AKA file if it existed in the design directory, even if it was not directed to do so. This is changed in release 5.0. In this version, the AKA file is used only if you specify the -s option.

`Warning 453. Due to the specification of the –s option, the design signal and symbol names will be shortened. However, this option is not recommended if you intend to run post-layout simulation because it changes the original signal names.`

Specifying the -s option forces MAP2LCA to produce an AKA file.

Error Messages and Recovery Techniques

This section describes the error messages that MAP2LCA2 can generate. An explanation and workaround solution follows each error message.

```
Error 1. At line number: Field too long.
```

This message indicates an invalid or corrupted MAP file. Rerun the program that created this file and try again. If the error persists, check with the company that supplied the program that created the XNF file.

```
Error 2. At line number: Unexpected LCANET record.
```

This error is issued when an LCANET record is found on a different line than the first.

```
Error 3. At line number: ENDMOD with no matching
MODEL record.
```

This message indicates an invalid or corrupted netlist file. Rerun the program that created this file and try again. If the error persists, check with the company that supplied the program that created the XNF file.

```
Error 4. At line number: Symbol sub-record outside
of symbol group. Record ignored.
```

This error is issued when the XNF syntax is violated, and records that should be within other records, such as MODEL, ENDMOD, or PIN, are found outside the group.

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 5. At line number: Illegal record inside
symbol group. Record ignored.
```

This error is issued when the XNF syntax is violated, and a record that is not allowed inside other symbol groups is found within the group, for example a PROG record inside a SYM group.

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 6. At line number: Premature EOF record in
symbol group.
```

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

`Error 7. At line` *number*`: Illegal record inside MODEL group.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

`Error 8. At line` *number*`: Premature EOF record in MODEL group.`

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

`Error 9. At line` *number*`: Premature End-of-file. No EOF record found.`

An EOF record must conclude every XNF file. This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

`Error 10. At line` *number*`: Unknown record type` *type*.

An illegal XNF record type was found in the file. This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

`Error 11. At line` *number*`: There are characters 'chars' on the end of a record line, after all expected data has been received.`

This message indicates an invalid input file. Re-generate the file

`Error 12. At line` *number*`: Invalid LCA netlist file. Invalid or missing LCANET record.`

This message indicates an invalid or corrupted XNF file. Every valid XNF file must start with the LCANET record, which lists the XNF version of the file.

`Error 13. At line` *number*`: Unsupported XNF netlist version` *version number.*

The version number listed in the LCANET record is not supported by this program. Only XNF versions 1, 2, 4, and 5 are supported.

```
Error 15. At line number: Valid part type must be
specified before netlist symbols can be read.
```

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 16. At line number: Invalid part record,
missing part type.
```

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 17. At line number: Missing name on SYM
record.
At line number: Missing type on SYM record.
At line number: Unknown symbol type.
```

These messages indicate an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 18. At line number: Invalid PIN record.
Missing name field.
At line number: Invalid PIN record. Missing or
invalid direction field.
At line number: Invalid PIN record. Invalid
direction field.
At line number: Invalid PIN record. Non-numeric
delay field.
At line number: A bidirectional field (B) must
only be used on a macro.
```

These messages indicate an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 19. At line number: Pin name name used
multiple times on symbol symbolname.

Error 20. At line number: Missing command on CFG
record.
```

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 21. At line number: CFG records allowed only
in CLB and IOB symbols.
```

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 22. At line number: Invalid SIG record.
Missing signal name.
```

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 23. At line number: Invalid EXT record.
Invalid direction dir.
At line number: Invalid EXT record. Missing signal
name.
At line number: Invalid EXT record. Bad or missing
direction field.
```

EXT records must have a direction field of I, O, T, B, or U.

These messages indicate an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 24. At line number: Invalid BUS record.
Missing bus name.
```

This message indicates an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 25. At line number: Invalid PULSE record.
Missing pin name field.
At line number: Invalid PULSE record. Missing or
invalid polarity field.
At line number: Invalid PULSE record. Invalid
polarity field.
At line number: Invalid PULSE record. Invalid or
missing minimum width field.
```

These messages indicate an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 26. At line number: Invalid PWR record. Bad
or missing polarity field.
At line number: Invalid PWR record. Polarity is
‘polarity’. Must be 0 or 1.
```

PWR record must have a 1 or a 0 in the polarity field.

These messages indicate an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 27. At line number: Invalid SETUP record.
Missing pin name field.
At line number: Invalid SETUP record. Missing
clock pin name field.
At line number: Invalid SETUP record. Missing or
invalid clock edge field.
At line number: Invalid SETUP record. Invalid
clock edge field.
At line number: Invalid SETUP record. Missing or
invalid setup time.
At line number: Invalid SETUP record. Missing or
invalid hold time.
```

These messages indicate an invalid or corrupted XNF file. Rerun the program that created this XNF file and try again. If the error persists, check with the company that supplied the program.

```
Error 28. At line number: Missing id on Hierg
record.
At line number: Missing type on HIERG record.
At line number: Missing name on HIERG record.
At line number: Missing filename on HIERG record.
At line number: Missing parent id on HIERG record.
```

These message indicate an invalid or corrupted MAP file. Rerun the design implementation process from the input XNF files. If the error persists, check with the company that supplied the program.

`Error 30. For general use of invalid parameters.`

`At line` *number*`: Unknown MAP symbol` *symboltype* `for SYM` *symbolname*`.`

This message is issued if a CLBMAP symbol has a MAP= parameter value that is not PUC, PUO, PLC, or PLO. See "Using CLBMAPs" in the "XNFMAP" chapter for more information about MAP= parameters. If there is no MAP= parameter on the CLBMAP symbol, then XNFMAP will default to MAP=PUC or MAP=PUO (if CLBMAP symbol has no A – E pins).

`At line` *number*`:` *type name* `has invalid parameter value` *parameter*`.`

This message is issued if a floating point value of the THI and TLO parameters have trailing characters that are not "ns" (for nanoseconds). Only the suffix "ns" can be added to a THI or TLO value.

`At line` *number*`: Unknown SYM record parameter` *parameter*

This message is issued if an unknown XNF symbol parameter is found. Check the parameter specification in the design file.

`At line` *number*: *parm* `parameter found on SYM` *symbolname*`, type` *symboltype*`.`

This message is issued if a parameter is assigned to a symbol that does not support that parameter, such as a FAST tag on a DFF symbol. Check the parameter for that symbol.

`At line` *number*`: Unknown PIN record parameter` *parameter*

This message is issued if an invalid parameter is assigned to a PIN. Check the parameter for that symbol pin.

`At line` *number*`: PIN record parameter` *parameter* `found on MAP symbol.`

This message is issued if an invalid parameter is assigned to a PIN statement for a CLBMAP. Only a P (pin-lock) parameter is allowed on CLBMAP symbol pins.

`At line` *number*`: Unknown SIG record parameter` *parameter*

This message is issued if an invalid parameter is assigned to a SIG record. Check the parameter for that signal.

`At line` *number*`: Unknown EXT record parameter` *parameter* `ignored.`

This message is issued if an invalid parameter is assigned to a EXT record. Check the parameter for that I/O pad.

`At line` *number*`: FILE parameter found on` *name* `SYM, type` *symboltlype*`. (Non-flattened design?)`

This message is issued if a FILE= parameter is found on a non-macro symbol. FILE= parameters should be added only to unflattened macro symbols. This warning can occur if a reserved name — such as AND or OR — is used as the name of a macro with a FILE=.

`At line` *number*`: Extra LOC parameter` *parm* `found on signal` *signalname*

This message is issued if more than one LOC parameter is found on a record. Multiple-block LOC parameters should be separated by semicolon (;) characters. If commas (,) are used to separate LOC parameters, this error is issued.

`At line` *number*`: Invalid MAP symbol type [PLO, PUO, or PLC] for HMAP symbol` *symbolname*`. PUC is the only valid MAP= parameter for an HMAP symbol.`

`At line` *number*`:` *parameter* `is not complete on PIN` *pinname*`.`

This message is issued if there is an invalid parameter on the PIN record.

`At line` *number*`: SYM record parameter TNM not allowed on timespec symbols.`

Only TSid parameters are allowed on TIMESPEC symbols.

`At line` *number*`: SYM record parameter` *parameter* `only allowed on symbols with TIMESPEC type.`

The only symbols that can have this parameter are TIMESPEC symbols.

`Error 31. At line` *number*`: Invalid LOC parameter` *name* `on symbol-type` *name*`.`

An invalid location specification was found on the indicated symbol. Check the legal LOC constraints for the FPGA family in use.

`Error 33. At line` *number*`: More than one` *parameter* `is specified on symbol` *symbolname*`. Only one is allowed per symbol.`

Only one RLOC, RLOC_ORIGIN, or RLOC_RANGE parameter is allowed per symbol in the input design.

`Error 34. At line` *number*`: Invalid RLOC parameter` *parameter* `on symbol` *symbolname*`.`

Refer to the *Libraries Guide* for the correct use of RLOC parameters.

`Error 35. At line` *number*`: Both TTL and CMOS parameters are specified on symbol` *symbolname*`. Only one of these parameters is allowed per symbol.`

`Error 36. At line` *number*`:` *parameter* `specified on symbol` *symbolname* `of type` *type* `RLOC-type parameters may not be used with decoders, clocks, logic symbols or IO primitives.`

`Error 37. At line` *number*`: FAST/SLOW/MEDFAST/MEDSLOW and FAST/SLOW/MEDFAST/MEDSLOW parameters have been found on symbol` *symbolname*`. These parameters should be mutually exclusive.`

`Error 37. At line` *number*`: FAST/SLOW/MEDFAST/MEDSLOW and FAST/SLOW/MEDFAST/MEDSLOW parameters have been found on signal` *signalname*`. These parameters should be mutually exclusive.`

`Error 38. At line` *number*`: TNM parameter` *TNM=value* `on symbol` *symbolname* `has illegal class type. The only legal class types are FFS, RAMS, PADS or LATCHES. The syntax should be TNM=name_value or TNM= class-type:name.`

`Error 40. At line` *number*`: Two different IOBs use the same external signal` *signalname*`.`

Two IOBs cannot use the same pad signal. Every pad signal must have a unique name. This error can occur if you have a pad signal connecting an input I/O symbol and an output I/O symbol, and the

two symbols are in separate modules of the design. If the two modules are mapped separately (map-then-merge design processing), then the two symbols will be made into two separate IOBs that both use the same pad signal. Although the intention might have been to make the pad signal a bidirectional signal, the separation of the two symbols into different modules caused this error. It is advisable to keep all the I/O elements of a single IOB (pad signal, input and output symbols) in the same module/hierarchy of the design.

`Error 53. Out of memory. Needed` *number* `bytes.`

There is insufficient memory to complete the XNF file processing. Check the memory requirements for the FPGA part type in use.

`Error 208. Un-mapped symbol found` *name*`.`

Run XNFMAP and MAP2LCA again.

`Error 222. Too many CLBs used.`

The design requires more than the available number of CLBs in the target part type. Either change to a larger part type or eliminate some of the logic.

`Error 223. Too many IOBs used.`

The design requires more than the available number of IOBs in the target part type. Either change to a part type with more pins or eliminate some of the I/Os.

`Error 224.` *number* `I/O pins used but only` *number* `available.`

The design requires more I/O pins than are available in the package type. Either change to a package with more pins or reduce the number of I/O pins. The number of I/O pins and the number of IOBs do not always match, as there are some packages with unbonded IOBs.

`Error 225. Symbol` *symbolname* `defines an oscillator, but none are available.`

More than one oscillator was specified. Only one crystal oscillator circuit is available on XC2000 or XC3000 devices.

`Error 226. Symbol` *symbolname* `defines an unavailable clock buffer.`

Only the GCLK and ACLK buffers are available on XC2000 and XC3000 devices.

`Error 227. Symbol` *symbolname* `defines a [GCLK/ACLK] clock buffer which is already assigned to symbol` *symbolname*`. It is an error to have multiple [GCLK/ACLK] symbols.`

One of the global clock buffers was used twice. Only one GCLK buffer and one ACLK buffer are available on XC2000 or XC3000 devices. Remove one of the extra clock buffers.

`Error 228. Invalid part type` *parttype*`.`

`Error 229. IOB` *name* `needed for oscillator but already used.`

The crystal oscillator requires two dedicated pins, XTAL1 and XTAL2. If the oscillator is used, the IOBs associated with these pins cannot be used as a normal I/O. This message is issued if one of these pins is specified as a normal IOB. Edit the design so it does not use these pins. Check the pin descriptions in *The Programmable Logic Data Book* to find the locations of XTAL1 and XTAL2.

`Error 230. Unable to find data file` *filename*`.`

The specified data file could not be found. Make sure that the indicated file exists in the \XACT or \XACT\DATA directory. If it does not exist, the implementation software should be re-installed. If the file does exist, use the DOS Set command to verify that the XACT environment variable points to the \XACT directory. Any spaces in the SET XACT= statement can prevent the data files from being found.

`Error 232. Insufficient information in partlist file.`

The partlist.xct file is out-of-date and does not contain all of the information necessary for MAP2LCA to process the design. Check that a valid part type is specified. If the part type is valid, the indicated data file might be corrupted. Re-install the implementation software.

`Error 234. Unable to open package file` *filename*`.`

Check that a valid package type was specified for the desired part. If the part and package type is valid, the indicated data file might be

corrupted. Re-install the implementation software.

`Error 237. Unable to open temporary work file.`

Check for a disk-full condition or incorrect file permissions. On a PC-based system, the DOS FILES variable may be set too low; use at least FILES=20 in the config.sys file.

`Error 239. Unable to rename temp file.`

MAP2LCA failed when attempting to rename its work file to the actual output name. This might occur if the target file already exists and is flagged read-only, or because of a disk-full condition. Check for these two conditions and run MAP2LCA again.

`Error 241. File name` *filename* `is too long.`

`Error 241. Illegal extension on name` *filename*`.`

`Error 242Unable to open LCA netlist file` *filename* `for reading.`

This typically means that the specified input file was not found in the current directory. Check the directory and file name and run MAP2LCA again.

`Error 257. Invalid CLB location` *loc* `specified on symbol` *symbolname*`.`

An invalid CLB location was specified. See the "XNFMAP" chapter for more information about specifying locations.

`Error 258. Symbol` *symbolname* `has CLB` *name* `location which is already used.`

The CLB location specified is used by another block. See the "XNFMAP" chapter for more information about specifying locations.

`Error 264. IOB symbol` *symbolname* `requires I/O block name. The GCLK and ACLK direct input signals (TCLKIN and BCLKIN) are only available from specific IOBs.`

Check the part's data sheet for the correct IOB pin name. However, you do not need to specify the direct clock input IOBs; MAP2LCA automatically assigns the correct IOBs.

`Error 270. Unknown command-line option` *option*`.`

An unknown option flag was specified on the MAP2LCA command line. See the "Options" section in this chapter for the legal option flags.

`Error 271. Extra command-line argument` *argument*`.`

An extra argument was found on the command line after the output file name. See the "Syntax" section in this chapter for the syntax of the command line.

```
Error 273. Missing part type -p option.
```

The -p option was specified at command line without the part type following.

```
Error 274. Option -N and -M are no longer
supported by MAP2LCA.
```

The mincut pre-placement step has been removed from MAP2LCA.

```
Error 282. External signal flagged as power/gnd.
```

A signal that connects to an I/O pad has been flagged as power or ground. An I/O pad cannot be connected directly to a power signal. If an output pin is to drive V_{CC} or GND, the input of the OBUF should be flagged appropriately.

`Error 291. Unknown location` *locstion* `specified on` *symbolname*`.`

Correct LOC parameter or use –i option to ignore all LOC parameters. The location specified for the indicated symbol is not valid. See the "XNFMAP" chapter for more information about specifying locations.

`Error 292. Too many` *type* `symbols connected to signal` *signalname*`.`

This message typically results from using more than the available number of TBUFs or pull-ups. The number of TBUFs available on a horizontal longline depends on the FPGA part type. Reduce the number of TBUFs or use a part with more resources. There are two pull-ups available for each horizontal longline. No more than two pull-ups should ever be specified per signal.

`Error 293. Location` *location* `cannot be used by TBUF` *name*`, which is already used by symbol` *symbolname*`.`

Two symbols specify the same location. See the "XNFMAP" chapter for more information about specifying locations.

`Error 294. Too many internal three-state signals used.`

This message typically results from using more than the available number of horizontal longlines. The number of horizontal longlines available depends on the FPGA part type. Reduce the number of 3-state signals or use a part with more horizontal longlines.

`Error 295. Symbol` *symbolname* `and symbol` *symbolname* `drive signal` *signalname*`, but their locations are on different longlines.`

All TBUFs and pull-ups that drive the same net must be placed on the same horizontal longline.

`Error 296. TBUF symbol` *symbolname* `on signal` *signalname* `specifies location which is already used for another three-state signal.`

A single horizontal longline can carry only one 3-state signal. This message is issued if symbols (normally TBUFs and pull-ups) driving two different 3-state signals are specified to be on the same longline. Change the location specifications to use different longlines.

`Error 300. Pullup and [`*OBUF*`/`*OBUFT*`/`*OUTFF*`] both used on external signal` *signalname*`.`

The pull-up resistors in XC3000 IOBs can only be activated if the IOB is used as input only. If an output buffer is used, the pull-up in that IOB might not be used. Remove the pull-up from the I/O pin.

`Error 310. Invalid LOC parameter` *location* `on symbol` *symbolname*`.`

Correct the LOC parameter for the symbol, or use the -i option to ignore all LOC parameters.

`Error 311. Conflicting LOC parameters on the EXT signal` *signalname* `and the IOB` *name*`.`

Correct the LOC parameter for the symbol, or use the -i option to ignore all LOC parameters.

`Error 325. Error while writing LCA information to disk. Some information may be found in file` *filename*`.`

Check for a disk-full condition or incorrect file permissions. On a PC-based system, the DOS files variable may be set too low; use at least FILES=20 in the config.sys file.

`Error 438. The` *part* `device is not supported by this product. The design cannot be mapped into the specified LCA part-type because the part is not supported by this version of software.`

`Error 442. Invalid package file` *filename*`.`

The package file has been corrupted. Re-install the implementation software.

`Error 443. Unable to open file partlist.xct.`

The specified data file could not be found. Make sure that the indicated file exists in the \XACT or \XACT\DATA directory. If it does not exist, re-install the implementation software. If the file does exist, use the DOS Set command to verify that the XACT environment variable points to the \XACT directory. Any spaces in the Set XACT= statement can prevent the data files from being found.

`Error 444. Missing or invalid data number` *data number* `in file` *filename*`.`

There is invalid data in the partlist.xct or speeds.spd files. Re-install the implementation software.

`Error 445. Can't change` *devicename* `to .dev suffix. This is an indication of corrupted or invalid data files.`

Re-install the implementation software.

`Error 446. Unknown alias-to part` *parttype*`.`

This is an indication of corrupted or invalid data files. Re-install the implementation software.

`Error 447. Failed to find` *parttype* `in part list.`

This is an indication of out-of-date, corrupted, or invalid data files. Re-install the implementation software.

`Error 460. No "bonded/unbonded" IO blocks are available for IOB symbol` *symbolname*`.`

PAD signals connected to OPAD, IPAD, and IOPAD symbols are placed only on IOBs that are bonded to package pins. Signals connected to UPAD symbols are placed only on IOBs that are not bonded to package pins. Neither the number of required bonded IOBs or unbonded IOBs can exceed the number that are available for the part/package. Change the package for the part to include more of the limited resource (bonded or unbonded IOBs) or change the symbols used for the pad signals in the input design.

`Error 2001. Unable to open AKA file for reading.`

Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file.

`Error 2002. Illegal prefix in line` *number*`.`

Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file.

`Error 2003. Unable to add symbol to table.`

Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file.

`Error 2007. No symbol for index` *number*`.`

Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file.

`Error 2008. PREFIX table overflow on symbol.`

Check that the AKA file created by the previous design iteration exists in the current directory and carries the name of the input design file.

`Error 2010. Unable to retrieve error message for UI error code` *number*`.`

Check that the AKA file that was created by the previous design iteration exists in the current directory and carries the name of the input design file.

APR

This program is compatible with the following families

- XC2000
- XC2000L
- XC3000
- XC3100

The Xilinx Automatic Place and Route (APR) program performs placement and routing of an FPGA design.

Before you can use APR, you must translate your XNF file into an LCA file by using the XNFMAP and MAP2LCAprograms.

The APR program reads an LCA design file, with or without routing information, generates block placement, routes a design's nets, and then writes the results to another LCA design file. APR writes a description of the final placement to a report file.

For information on the placement and routing of XC3000A, XC3000L, XC4000, or XC5200 designs, refer to the "PPR" chapter in the *Development System Reference Guide*.

Using XACT Design Manager

You can invoke APR through the XACT Design Manager (XDM), a menu-driven interface for executing all FPGA development operations. Refer to the "The XACT Design Manager" chapter in the *Development System Reference Guide* for a complete description.

Once you access XDM, select **`PlaceRoute`** → **`APR`** from the main menu. A cascading menu appears with the most commonly used APR commands. To display all available APR commands, select the

-x option. Refer to the "Options" section in this chapter for a detailed description of each menu selection.

Syntax

Use the syntax shown here to create a placed and routed design file from your LCA design file.

apr [*options*] *input*.**lca** *output*.**lca**

To use any command-line options, specify them before the input design name. You can also invoke any option through XDM. The input design file must be an LCA file.

To view a list of APR's commonly used options onscreen, enter **apr** and press ↵.

Options

Command-line options allow you to change the way the APR program operates. Some of the options tell APR which files to use. Others specify the course that the placement and routing algorithms take, which in turn may improve APR results or run time. In general, use command-line options to control global aspects of APR operation; you can achieve more detailed control using constraints files. Refer to the "APR Constraints" section in this chapter for more information.

APR ignores the difference between upper- and lower-case command-line options. For example, APR recognizes both -l and -L as the same option. File names can be case-sensitive depending on your operating system.

Positioning Options on the Command Line

To enter a command-line option, position that option on the command line before entering the input design name, as illustrated by the following example. A hyphen precedes the option letter.

apr -y *input output*

For more than one command-line option, enter options in any order. Command-line options that do not require arguments can be combined with other options that do not require arguments. For

example, you can specify j, p, and l options by entering either of the following command sequences.

apr -j -p -l *input output*

apr -jpl *input output*

Some command-line options require arguments, such as the -s and -c options. You can enter them as shown in the following example.

apr -s 5794 -c myfile.cst *input output*

The syntax and meaning of each command-line option are discussed in the following sections.

–a Specify the Number of Routing Attempts

The -a option enables you to specify the number of routing attempts. If you do not specify a *count*, the default is three attempts.

apr -a *count input output*

–c Use a Constraints File

The -c option reads constraints from the named user constraints file. APR attaches a .cst extension to the constraints file name and searches for the constraints file in two places: the current directory and the directory that contains the input design file. If the file name has a leading path, APR also searches the directory specified in that leading path.

APR reads constraints files that have file names with a .cst extension or without an extension. In the first two examples below, APR will read the file named *filename*.cst or *filename*. APR automatically supplies a .cst file name extension. In the third example, APR issues an error message and aborts because the constraints file name has an extension other than .cst.

apr -c *filename*.cst *input output*

apr -c *filename* *input output*

apr -c *filename*.xxx *input output*

See the "APR Constraints" section in this chapter for more information on constraints files.

–g Use an LCA File as a Guide

The -g option uses the specified LCA file as a guide for the input design. The term *guided design* refers to the process in which a previously implemented design — also known as a guide file — is used to guide placement and routing. This process allows you to modify or add logic to a design while preserving the layout and performance previously achieved.

APR appends a .lca extension on guide file names and searches for the guide file in two places: the current directory and the directory that contains the input design file. If the file name has a leading path, APR also searches the directory specified in that leading path.

You should specify the guide file name with a .lca extension or without an extension. Do not use an extension other than .lca. The first two examples below are correct. In the second example, APR automatically supplies the .lca file name extension. The third example is incorrect because the guide file name must have a .lca extension. In this case, APR issues an error message and aborts.

`apr -g` *filename*`.lca` *input output*

`apr -g` *filename input output*

`apr -g` *filename*`.xxx` *input output*

APR looks at the physical layout of the guide design, using it to implement any logic the two designs share. After using all available block, pin, and net information from the guide design, APR starts placing and routing the additional logic.

Incremental Design

If you wish to make changes to a design that was difficult to place and route, use the placement and routing from your old design to guide the placement and routing of the changed unrouted input design. The -g option is useful for designs that nearly fill the FPGA (80 percent utilization or above), designs having tight timing specifications, or designs containing any portion that was manually placed and routed.If the changes are minor, the run time for APR decreases considerably.

The extent to which the old design is preserved depends on several factors.

- The extent of the design changes
- The design creation method: XACT Design Editor (XDE), schematic capture, Boolean logic files, and so on
- The implementation method: hierarchical XNF files, flattened XNF files, nonstandard partitioning control, submodule mapping, and so on

Note: For the best results when using the guide-file option, use the map-then-merge flow in the XMake program, and then the guide-file option (-k) in XNFMAP.

Block, Pin, and Net Matching

The -g option causes APR to read the following information from the guide design.

- Block names and locations
- Net names and their pin locations on the blocks
- Net routing information

APR compares the two files and performs the following steps.

- It matches blocks based on finding blocks in the new design that are connected to the same nets as blocks in the guide design.
- It swaps the pins on matched blocks so the nets are connected to the same pins in the new design and the guide design.
- It preserves the routing of nets that are connected to the same pins on the same blocks in both the new design and the guide design.

Note: Two pins are considered equivalent for the purpose of matching nets if they can be swapped with each other.

–j Place the Design without Routing

The -j option places the design only; it does not route it. This option is useful when an initial placement is desired for a design that will be routed manually in XDE. It can be used with other options to produce special output files. See the “Useful Option Combinations” section in this chapter for more details.

–l Lock All Blocks in Place

The -l option locks all blocks in place, effectively skipping the placement process altogether. This option is useful when you want to preserve the placement information of the input design file and change the routing only. It is also useful when routing a design after making minor placement changes with XDE.

–o Redirect Message Output to a File

The -o option redirects all message output into the named file. This option prevents APR message output from being displayed on the screen. If you name the message file the same as the output design file, APR automatically appends a .out extension to the file name to differentiate it from other files. Suppose you entered any of the following examples.

```
apr -o filename.out input output
apr -o filename input output
apr -o filename.xxx input output
```

In the first two examples, APR redirects all messages to the file named *filename*.out. In the second example, APR automatically supplies the .out extension. In the third example, APR aborts because the message file requires a .out extension.

The APRLoop program intercepts the -o option and creates a single output file containing the messages for all APR runs. See the "Placing and Routing Larger Designs" section for more information on how APRLoop handles the -o option.

–p Preserve All Initial Routing Information

The -p option directs APR to preserve all initial routing information. By implication, all pins that connect to preserved interconnect are locked, as are the blocks they belong to.

–q Skip the Annealing Algorithm

The -q option skips the first phase of the placement algorithm. This option is most useful for sparse designs and designs with excellent initial placements. Because this option causes APR to skip its most

time-consuming task, the run time might decrease. However, this option might result in non-optimal placement.

–r Set the Router Type

The -r option sets the router type, such as 1, 2, 3, or 4. The default is -r4, the latest and most effective option. Router types 1-3 are previous versions of the router.

–s Set a Seed for Multiple APR Runs

The -s option specifies a seed for the APR pseudo-random number generator. A seed is a random number that determines the final placement of the design. By default, APR uses the operating system time to set the seed. Alternatively, you can specify any seed from 1 to 32767. In the following example, the seed is set to 5794.

```
apr -s 5794 input output
```

If you run APR successively without -s, results differ since invoking it at different times produces different seeds. Successive runs of APR using the same initial seed produces identical results, unless the input design is different or you used different constraints. The -s option only affects the results of the placement algorithm and not the routing algorithm.

–t Improve the Timing on Existing Routing

The -t option skips normal placement and routing phases and improves the existing nets by decreasing net delays. APR unroutes and reroutes existing routing in order to achieve this goal. During this process, APR might route previously unrouted pins. This option is recommended for designs with less than 10 unrouted pins. APR continues the timing improvement process until it detects no further improvement.

Note: You can preserve routing for individual nets by using a constraints file. See the “APR Constraints” section in this chapter for more information.

This option might take some time to complete. You can save the current state of the improved design by using Ctrl-Break on a PC or Ctrl-C on a workstation. From the displayed menu, choose “W” to write the design and its report file. You can then quit or choose to

continue the routing improvement process. Refer to the "Routing Phase" section at the beginning of this chapter for more information.

–w Suppress the Overwrite Warning

The -w option suppresses the output design file overwrite warning. If you use this option, APR does not issue a warning when the output design name on the command line corresponds to an existing file.

Note: Use this option cautiously.

–x Display All APR Functions

The -x option displays onscreen all current APR options. If you enter apr and press ↵ without specifying -x, APR only displays the most commonly used options.

```
apr -x
```

–y Use a Faster Placement

This option specifies the use of an alternate placement algorithm for the simulated annealing placer. This alternate algorithm can decrease program run time for designs larger than a 3042. However, this option might result in less optimal placement.

Command-Line Options Summary

Table 5-1 summarizes APR's command-line options. The table includes the command syntax, the system default (if applicable), the minimum value for the argument (if applicable), the maximum value for the argument (if applicable), and a one-line summary of the option.

Table 5-1 Summary of Command-Line Options

Syntax Summary	Default	Min	Max	Description
-a count	3	1	99	Sets number of routing attempts
-c file_name				Reads constraints file
-g file_name				Specifies guide design
-j				Skips routing, places only
-l				Skips placement, routes only
-o file_name				Redirects message output
-p				Preserves routing information
-q				Skips annealing, quenches only
-r router	4			Sets router
-s seed	Random	1	32767	Sets seed
-t				Improves timing on routing only
-w				Suppresses the output design file overwrite warning
-x				Displays all APR options
-y				Uses a faster placement schedule

Useful Option Combinations

You can combine certain command-line options to perform the following tasks.

- Complete the routing of your design
- Create a report file
- Add logic to existing FPGA designs

The following sections discuss each task in detail.

–pl Complete the Routing of Your Design

After manual routing, you can evaluate your routing's effectiveness by combining the -p and -l option. APR does not change the placement (-l) and attempts to complete manual routing (-p) for the remaining nets. Running APR with -l and -p is fast, depending on the number of prerouted nets.

If the number of unrouted pins and nets decreases and/or the performance goals are met, the manual routing is probably successful. If the design still has a high number of unconnected pins and unrouted nets, use the -p option only and allow the other logic to move, while preserving manual routing.

–jpl Create a Report File

Combining -j, -p, and -l quickly creates an output LCA file identical to the input file and produces a report file, usually in five minutes or less.

`apr -jpl` *input output*

These options are useful for creating report files on designs of unknown origin or designs created in XDE, or for obtaining the delays of a different speed grade. APR creates the report file and appends a .rpt extension to the end of the file name.

–ljg Add Logic to Existing LCA Design

Combining the -l, -j, and -g options produces an output LCA file that is affected by both the SCP constraints file and the guide LCA file.

APR performs the following steps.

1. Guides the design according to the -g option conditions specified in the "Incremental Design" section in this chapter.
2. Reads and executes the SCP file constraints.
3. Performs no new placement or routing.
4. Terminates the program and writes error messages to the report file if there are conflicts in the guide LCA or SCP files.

APR typically runs in less than five minutes when you use this option combination. This combination lets you create a snapshot of a new design based on an old design. You can examine the output file, either

in XDE or via the report file, to determine how much of the old design was preserved. If the results are unsatisfactory, you can change the input file, the guide file, the constraints file, or any combination of these to produce the desired result.

Using a Batch File for Multiple Runs

If you are using a Sun workstation or PC and want to perform several runs with different options or constraint files, you can create an executable batch or UNIX shell file to execute several statements serially.

Using a text editor, you can create a file that includes a series of APR command lines. Suppose you create a batch file that contains the following command lines.

```
apr -c file1 -o run1 input output1
apr -c file2 -o run2 input output2
apr -y -c file3 -g routed -o run3 input output3
```

In this example, the batch file enables you to use different constraint files, a guide file, and placement algorithms in a single executable file.

Running APR as a Background Process

On a Sun, execution on large designs can take a few hours. If your system administrator logs out users for terminal inactivity, you should make APR run as a background process by specifying the & (ampersand).

```
apr -o run1 input output &
```

Once the job is a background process, it cannot be terminated due to terminal inactivity. If you have questions regarding terminal inactivity logouts, consult your system administrator.

Interrupting APR During Processing

APR has three phases: annealing, quenching, and routing. The annealing and quenching phases are part of the simulated annealing placement algorithm. This algorithm evaluates the value of the current placement after moving blocks in a semi-random manner.

During the annealing phase, APR allows the blocks to be temporarily placed in a way that degrades the current quality of the placement. It does this to "shake up" the design in the effort to find the best solution.

During the quenching phase, blocks are only moved if the movement improves the quality of the design. Once APR completes the annealing and quenching placement phases, APR routes the design.

Annealing and Quenching Phases

To interrupt APR during the first two phases of processing, press Ctrl-Break on a PC or Ctrl-C on a workstation. A menu with the following options appears.

E — Switch to next phase
S — Suspend APR to check current results
C — Continue current phase of placement
Q — Quit this APR run entirely

Select one of these options.

Choosing E ends the current phase and begins the next phase. If APR has been annealing, it immediately starts quenching. If it has been quenching, it immediately starts routing.

Choosing S suspends APR and opens up a DOS or UNIX shell. Enter Exit when you are ready to return to this menu.

Choosing C continues where APR left off when it was interrupted.

Choosing Q completely ends the current APR run.

Routing Phase

If you press Ctrl-Break on a PC or Ctrl-C on a workstation during the routing phase, a menu with the following options appears.

S — Suspend APR to check current results
W — Write current design and report
C — Continue routing
Q — Quit this APR run entirely

Choosing S suspends APR and opens up a DOS or UNIX shell. When you are ready to return to this menu, enter Exit and press ↵.

Choosing W writes an output LCA file and adds information about the current state of the design to the report file. Once this information is written, the menu reappears so that you can choose to suspend APR and check the results, continue routing, or quit entirely. Every time you choose this option, the LCA file is rewritten, and the report file has new information added to it.

Choosing C continues where APR left off when it was interrupted.

Choosing Q ends the current APR run.

Note: Do not use Ctrl-C to break between phases of the APR program when running APRLoop because this action might cause APRLoop iterations to end. For more information about APRLoop, refer to the "Placing and Routing Larger Designs" section in this chapter.

Design File Names

The following sections discuss file naming conventions, file name extensions, and leading path specifiers on design names.

You must specify the name of the output design that APR creates. If the output design name corresponds to the name of an existing design file, APR issues a warning message before overwriting the existing design file. You are prompted to supply a different output design name. If you decline, APR overwrites the existing design file.

File Name Extensions

APR can only read LCA design files. If you specify a file name extension other than .lca, APR prints an error message and terminates. If you specify an input or output design name without the LCA file name extension, APR adds the correct extension before searching for or creating any files.

Leading Path Specifiers

Your specific platform determines how you indicate leading path specifiers on design names, as follows.

PC

On a PC, if you specify a design name with a complete path name (\xact\designs*input*.lca), APR uses the named file. If you specify a design name with a partial path name (designs*input*.lca) or no path name (*output*.lca), APR appends the specified name to the path name of the current directory, as follows.

Command line:	`apr` *designs\input output*
Current directory:	`\xact`
Resulting input file:	`\xact\`*designs\input*`.lca`
Resulting output file:	`\xact\`*output*`.lca`

Sun Workstation, HP700, and RS6000

On a Sun workstation, HP700, or RS6000, if you specify a design name with a complete path name (/user/xilinx/*input*.lca), APR uses the named file. If you specify a design name with a partial path name (xilinx/*input*.lca) or no path name (*output*.lca), APR appends the specified name to the path name of the current directory, as follows.

Command line:	`apr xilinx/`*input output*
Current directory:	*/user*
Resulting input file:	*/user/*`xilinx/`*input*`.lca`
Resulting output file:	*/user/output*`.lca`

Apollo

On an Apollo workstation, if you specify a design name with a complete path name (/local_user/xilinx/*input*.lca), APR uses the named file. If you specify a design name with a partial path name (xilinx/*input*.lca) or no path name (*output*.lca), APR appends the specified name to the path name of the current directory, as follows.

Command line:	`apr` *designs/input output*
Current directory:	`/local_`*user*
Resulting input file:	`/local_`*user*`/xilinx/`*input*`.lca`
Resulting output file:	`/local_`*user*`/`*output*`.lca`

Input Files

APR uses a series of files in order to place and route your design. APR can then generate an output design, message, or report file that include the results of the placement and routing process. Table 5-2

presents a summary of the files APR uses to place and route your design. The following sections discuss each in detail.

Table 5-2 Input File Summary

Input Files	Extension	Notes
Input design file	.lca	Required to exist
Schematic constraints file	.scp	Used only if it exists
User constraints file	.cst	With -c option
Guide design file	.lca	With -g option

Input Design File

APR receives most of the information it needs about a design from the input design file. The MAP2LCA program appends the .lca extension to your design file name. APR reads the LCA design information from this file. You specify the name of the input design file on the APR command line.

Schematic Constraints File

LCA design files automatically created from the netlist output of a schematic capture program are often accompanied by a schematic constraints file, *filename*.scp, that provides information so APR can route the design more effectively. The MAP2LCA program creates the *filename*.scp file for APR from the constraints information in the input design file.

Xilinx recommends against modifying the schematic constraints file, since those modifications are lost if you change your schematic and regenerate the LCA design file. APR automatically reads the schematic constraints file if it exists. See the “APR Constraints” section for more information.

User Constraints File

The user constraints file, *filename*.cst, is usually created using a text editor.

It contains additional information about the design or about the desired placement and routing of the design. You can use the -c

command-line option to submit a user constraints file to APR. See the "APR Constraints" section for more information.

Guide Design File

The guide design file, *filename*.lca, is an LCA file whose placement and routing acts as a guide to the input file.

Use the -g option to submit a guide design file to APR. See the "Incremental Design" section for more information.

Device, Package, and Speed Information Files

The various device, package, and speed information files contain essential information about each FPGA die and package type. Their file names end with one of the following extensions — .dev, .pkg, or .spd.

The speed information file contains information about FPGA speed grades needed by the net delay calculator in APR. These files are provided with the APR program and should not be modified or renamed.

APR Constraints

APR receives most of the information it needs about a design from the input design file.

When MAP2LCA creates an LCA design file, they also automatically create a companion constraints file. This file has the same name as the LCA design file, but with a .scp extension. This file contains the constraints found in your input design. APR always looks for this file and, if it exists, reads the constraints information from it. If you do not want APR to use the automatically generated constraints file, you should delete it or rename it. If you change your design and remake your LCA design file, MAP2LCA overwrites any existing SCP constraints file. For this reason, do not edit the SCP constraints file because your changes are lost when MAP2LCA regenerates the file.

When APR needs more information than the design file can provide, you can create an additional constraint files, which contains constraints in addition to those specified in the system-created SCP constraints file. This constraints file name must have a .cst extension.

Entering the following command causes APR to automatically search for the constraints file *input*.scp and use the constraints information in that file.

apr *input output*

APR automatically searches for the constraints file *input*.scp, then uses the constraints information in that file.

In the next example, APR searches for and uses the constraints information contained in the user-created constraints file *juke*.cst.

`apr -c juke` *input output*

Remember that APR attaches the .cst extension to all user-created constraints file names.

SCP Files

The MAP2LCA program automatically creates the SCP file. An example of an SCP constraints file is shown below. The first four lines are comments as indicated by the semicolon in the first column. It has the same name as the LCA design file, as specified on the first line, but the file name contains an SCP extension.

```
; Design: dramctrl.lca,
; Created by MAP2LCA Ver 5.00 at 15:47:02 Dec 1, 1993
; (NOTE: Don't edit this file.
; It is rewritten each time MAP2LCA is run)
flag net longline ADDRSTRB ;
flag net normal CNTRSET ;
flag IOB external ALE ;
place block BANKSEL P27 ;
weight net critical RAS ;
weight net 12 MUX ;
```

CST Files

If you create a constraints file using a text editor, you must give the file name a CST extension. This type of constraints file can help APR produce better results or run faster. APR reads your constraints file if you enter it on the command line with the -c option. APR always uses the constraints in the SCP file first, if it exists, and then uses the CST file.

Suppose you want to create some constraints in addition to the constraints contained in the dramctrl.scp file. You could create a file

called newcon.cst with a text editor and specify it on the APR command line as follows.

```
apr -c newcon dramctrl dramout
```

The following illustrates the contents of the constraints file newcon.

```
lock net RESET ;
lock pin BIT0.A BIT1.X BIT2.C ;
lock iobs ;
include newcon2.cst ;
lock block ADRBIT0 ADRBIT1 ;
```

The lock pin and lock block lines show how you can lock several pins or blocks using only one line. With the Include statement you can call other constraints files from within a constraints file.

Note: If a constraint in the SCP file conflicts with a constraint in the CST file, APR issues an error message if it cannot resolve the conflict. For example, if your design places a CLB at location AB, but the CST file prohibits the use of AB, APR issues an error message.

Case Sensitivity in Constraints Files

The operating system under which APR is running determines if case is ignored for constraint file names. Generally, most operating systems are not case sensitive in constraints files.

Definitions

The following definitions are useful when reading the descriptions of constraints.

- Block — Any block of the design, for example, CLB, IOB, and so on.
- Pin — A block's input or output pin.
- Net — Any net named in an Addnet statement in the input LCA design file. A net can contain zero or more pins.
- Net weight — The higher the net weight, the more important it is that the net be routed using a fast path.
- Location — A specific place where APR should assign a block as part of a placement. Each Xilinx FPGA contains a fixed number of CLB and IOB locations.

- External IOB — An IOB APR must place on an IOB location that is bonded to a package pin. Read "External and Internal IOBs" at the end of this section for more information.
- Area — An area of two or more blocks where APR can assign or prohibit a block as part of a placement, as illustrated in Figure 5-1. For example, CLBs can be confined to or prohibited from a rectangular region of the CLB array. IOBs can be assigned to or prohibited from an edge or half-edge of the part.

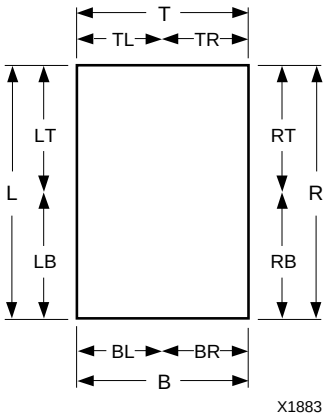


Figure 5-1 Edge Constraints for IOBs

The following list defines the letter designations in the previous figure.

T — top edge
R — right edge
B — bottom edge
L — left edge
TL — top-left
TR — top-right
RT — right-top
RB — right-bottom
BR — bottom-right
BL — bottom-left
LB — left-bottom
LT — left-top

- Internal IOB — Any IOB that is not bonded to a package pin.
- Locked block — A block whose placement has been locked with a Lock Block, Place Block, Lock IOBs, or Lock Net constraint, or with the -p command-line option.
- Net-locked block — A block whose placement has been locked because it has a pin routed to a net that was locked with a Lock Net constraint or with the -p command-line option. Every net-locked block is by definition also a locked block.

Constraints

A constraints file can be empty or contain multiple comments, statements, or both. This syntax applies to both SCP and CST files. Comments begin with a semicolon (;) in the first column and end with a carriage return. Break comments longer than one line into two or more comments, each beginning with a semicolon in the first column.

Statements begin with one or more keywords and end with a semicolon. Separate all words in a statement, both keywords and arguments, from each other by one or more spaces, tabs, or carriage returns. Since statements end with a semicolon instead of a carriage return, they can span any number of lines.

The following conventions are used in the sample syntax statements.

- Any word or phrase in italics represents a type of argument. For example, *block_name* should be replaced with the actual name of a block. All other words or phrases should be entered literally; they are keywords.
- An ellipsis (...) indicates when more than one of an item is acceptable. For example, *"file_name ..."* means one or more file names.

This section describes each constraint available in APR. Table 5-3 provides a summary of APR constraints.

Allow Block

The Allow Block constraint tells APR that the specified block — an IOB, CLB, TBUF or pull-up — must be placed in one of the specified locations (*loc1* through *locn*), as illustrated by the following syntax.

allow block *blockname loc1 loc2 ... locn* **;**

The *blockname* argument is the logical name of an IOB, CLB, TBUF, or pull-up in the input design. The *loc1* through *locn* arguments are physical locations on the part, as displayed in XDE.

For CLBs and TBUFs, you can specify the location as a specific physical location (AB, AC, TBUF.BE.1) or as a rectangular area (BB:HH, AC:DE, TBUF.CC.1:TBUF.ED.2) defined by the upper left and lower right CLB positions of the area. For IOBs, you can specify the location as a specific physical location (P9) or as an area on the edge or half-edge of the part. You can also use wildcards to specify a group of locations.

The following constraint forces APR to put the block named mux1 into either CLB location BC or anywhere in column D, as indicated by the wildcard character, * (asterisk).

```
allow block mux1 bc *d ;
```

The following constraint forces APR to put the block named Enable into any of the CLBs in the area bounded by CLB locations EB and GD.

```
allow block enable eb:gd ;
```

The following constraint forces APR to put the IOB block named INA into any of the IOBs located on the top edge of the part.

```
allow block ina t ;
```

For other IOB location options, refer to the previous figure, Edge Constraints for IOBs.

Flag IOB

The external form of this statement tells APR that the named IOBs are external and must be placed on a pin-bonded IOB. The internal form tells APR that the named blocks must be placed on an IOB that is not bonded to an external pin. The *blockname* arguments are logical name of IOBs in the input design. See the "External and Internal IOBs" section in this chapter for more information.

flag iob external *blockname ...* **;**
flag iob internal *blockname ..* **;**

Flag Net

The Flag Net constraint sets a flag for the named nets. These flags are as follows.

Critical — Weight net critical
Uncritical — Weight net uncritical
Normal — Weight net normal
Longline — Routes named nets using longlines

The syntax for each flag follows.

flag net critical *netname ...* **;**
flag net uncritical *netname ...* **;**
flag net normal *netname ...* **;**
flag net longline *netname ...* **;**

You can set the net weight with either this constraint or the Weight Net statement. Net weights are only significant in their relative magnitude, not in their absolute value. Weighting all nets critical produces the same results as weighting all nets normal.

Include

The Include constraint tells APR to process the constraints in each of the specified files before processing the remainder of the file containing the Include statement. Each file is treated as if it had been specified with -c; APR searches for a file with a .cst extension in the following directories.

- First, in the directory specified by the leading path on the file name.
- Second, in the current directory.
- Third, in the directory containing the input design file.

To use this constraint, enter the following syntax.

include *filename* **;**

A file can also contain Include statements including other files. The maximum nesting depth for constraints files is ten.

After processing all the constraints in an included file, APR closes the included file. Processing continues either with the contents of the

next file named in the Include statement or with the next statement in the current file if no more files were named.

Lock Block

The Lock Block constraint prevents the named blocks from being moved during the placement phase. To use this constraint, enter the following syntax.

lock block *blockname ...* **;**

Usually a block's location during constraint processing is the same as that occupied in the input design file. However, a block's location might have been changed with a Place Block statement.

APR automatically locks a block if one of its pins is on a locked net and that pin is connected to programmed interconnect, that is, if the block is net-locked.

Use this constraint with caution. You can over-constrain APR so that it cannot find a placement solution that satisfies all specified constraints. One way to do this is to lock enough internal IOBs onto bonded IOB locations so that all external IOBs cannot be placed. APR detects this condition and issues an error message.

Lock IOBs

The Lock IOBs constraint locks all IOBs in their current locations. You can lock individual IOBs with the Lock Block command. To use this constraint, enter the following syntax.

lock iobs ;

Lock Net

The Lock Net constraint enables APR to determine which pins are presently connected to programmed interconnect for each specified net. APR locks those pins and net-locks their blocks. To use this constraint, enter the following syntax.

lock net *netname ...* **;**

APR preserves all routing information for that net undisturbed by previous Place Block statements and protects routing information from further change.

Once you lock a block by using this command, you cannot use the Place Block constraint. The Place Block statement removes the routing information associated with the pins on the block being placed.

Pins that belong to a net not connected to programmed interconnect are not locked, nor are their blocks.

Lock Pin

The Lock Pin constraint tells APR not to move the named pins even if moving them might improve the final routing completion. To use this constraint, enter the following syntax.

`lock pin` *pinname ...* **`;`**

A pin is automatically locked if it is connected to programmed interconnect and is on a net that is locked with the Lock Net command or with the -p command-line option.

Place Block

The Place Block constraint assigns the named block to the named location. To use this constraint, enter the following syntax.

`place block` *blockname location* **`;`**

Block type and location must match; you cannot assign a CLB to an IOB location. APR automatically locks the block after it is assigned to its new location. APR assigns the block even if it was previously locked, except if the block is net-locked.

You cannot place net-locked blocks with this constraint. It removes the routing information associated with the pins on the block being placed.

If APR places a block on a location already occupied by another block, it displaces the other block, unless the other block is already locked. APR removes the routing from all pins on the placed block, unless the new location and old location match.

If a Prohibit Location statement prohibits the use of a location, a Place Block statement cannot use that location.

Place Net

The Place Net constraint places the blocks connected to the specified net along the longline if possible. This constraint is effective for placing TBUF output nets and TBUF enable nets. To use this constraint, enter the following syntax.

place net *netname location* **;**

Enter the location as a longline name in the form used by the XDE's Querygrid command. Valid names can be determined by loading the design into XDE, starting the Querygrid command, then using the mouse to click on any programmable interconnect point (pip) along the desired longline. XDE returns the longline name in the form *rownumber*.X.*long*.Y.

For example, the following constraint forces APR to place the blocks connected to the net named bus<0> on the first horizontal longline at the top of a 3020. Enter the longline name as it appears in XDE.

```
place net bus<0> row.A.long.3 ;
```

Prohibit Block

The Prohibit Block constraint prohibits APR from placing the specified block—IOB, CLB, or TBUF—in any of the specified locations. To use this constraint, enter the following syntax.

prohibit block *blockname location* **;**
prohibit block *blockname area* **;**

You can use wildcards when specifying locations. You can specify areas for CLBs by listing the CLBs at the upper-left and lower-right corners of the box. You can specify areas for IOBs by using the edge or half-edge constraint specifications.

In the following example, the constraint prohibits APR from placing mux1 in any of the CLBs inside the box defined by corner locations CC and EE.

```
prohibit block mux1 cc:ee ;
```

The following constraint forces APR to prohibit the placement of IOB in_a on the right edge of the part. For additional location options, refer to Figure 5-1 in the previous section.

```
prohibit block in_a r ;
```

Prohibit Location

The Prohibit Location constraint prohibits APR from placing any configured blocks—IOBs, CLBs, or TBUFs—in the specified location. Once this constraint prohibits the use of a location, a Place Block statement cannot use that location. To use this constraint, enter the following syntax.

prohibit location *location* **;**

In the following example, APR prohibits all CLBs from being placed in the third row from the top, as indicated by the letter "c."

```
prohibit location c* ;
```

The size of the part determines the number of rows and columns. All rows and columns are designated by a single alphabetic character.

Weight Net

The Weight Net constraint sets the net weight for the named nets. Three symbolic net weights are currently defined as follows.

Critical = 10
Normal = 3
Uncritical = 1

The last form of this statement allows you to explicitly set the net weight to any value between 1 and 99. Enter this constraint by using one of the following syntax examples.

weight net critical *netname ...* **;**
weight net normal *netname ...* **;**
weight net uncritical *netname ...* **;**
weight net *weightvalue netname ...* **;**

You can also set net weights using the Flag Net statement.

Net weights are only significant in their relative magnitude, not in their absolute value. Weighting all nets critical produces the same results as weighting all nets normal. Weighting a net 99 and the next 98, 97, 96, ...3, 2, 1 controls the individual net ranking and routing order. All nets with the same weight have the same rank.

Table 5-3 Summary of APR Constraints

Syntax Summary	Meaning
Allow Block blockname location ... ;	Places block in any of the specified locations.
Flag IOB Internal blockname ... ;	Flags IOB as internal.
Flag IOB External blockname ... ;	Flags IOB as external.
Flag Net Critical netname ... ;	Same as Weight Net Critical.
Flag Net Longline netname ... ;	Allows net to use long line.
Flag Net Normal netname ... ;	Same as Weight Net Normal.
Flag Net Uncritical netname ... ;	Same as Weight Net Uncritical.
Include filename ... ;	Reads more constraints from file.
Lock Block blockname ... ;	Does not move block.
Lock IOBs ;	Does not move any IOBs.
Lock Net netname ... ;	Preserves net routing information.
Lock Pin pinname ... ;	Does not swap pin.
Place Block blockname location ;	Places block, then locks it.
Place Net netname longline ... ;	Guides blocks on net along named longline
Prohibit Block blockname location ... ;	Prohibits use by the named block.
Prohibit Location location ... ;	Prohibits any use of the location.
Weight Net Critical netname ... ;	Raises net weight above normal.
Weight Net Normal netname ... ;	Sets net weight to normal.
Weight Net Uncritical netname ... ;	Lowers net weight below normal.
Weight Net weightvalue netname ... ;	Sets net weight as desired.

Improving APR Results

Use the Place Block statement to preplace those parts of your design for which you already know the best placement. You can preplace and preroute whole sections of your design and use the Lock Net statement to lock those sections. APR then only replaces and reroutes those parts of the design that are not locked. This incremental use of APR is useful for those designs containing timing-sensitive sections whose placement and routing must meet strict delay requirements.

The more blocks you preplace, the less time APR takes to place and route your design.

External and Internal IOBs

Using FPGAs in packages that have fewer pins than the number of IOBs places some restrictions on IOB placement. When an IOB is being used as an input and/or an output of the FPGA, the IOB must be placed on an IOB location bonded to a package pin.

You can configure some IOBs simply as registers for internal use without the need for bonding to a package pin. APR distinguishes between these two uses of IOBs. IOBs requiring a bonded IOB location are considered external IOBs. IOBs that do not require a bonded IOB location are considered internal IOBs. APR uses the initial placement and the constraints in the SCP file to determine whether an IOB is external or internal.

The Flag IOB statement in a constraints file explicitly flags an IOB as external or internal. Flagging an IOB overrides any designation that APR assigned to it during initial placement. The Flag IOB constraint can designate more IOBs as external than there are bonded IOB locations to accommodate them, resulting in an "over-constrained" condition. APR detects this over-constrained condition and issues an error message. The following options are available.

- Designate some of the IOBs as internal, if possible.
- Reduce the number of external IOBs needed in the design.
- Move the design to a package with a higher pin count.

Another over-constrained condition results from placing and locking IOBs designated as internal on bonded IOB locations. It effectively reduces the number of bonded IOB locations that are available for the external IOBs that need them. Over-constraining occurs when you either designate an IOB as internal and then lock it on a bonded IOB location, or use the Place Block statement to place an internal IOB on a bonded IOB location. APR detects this over-constrained condition if it occurs and issues an error message. You can then select one of the following options.

- Designate more IOBs as internal.
- Make sure internal IOBs are not locked on bonded IOB locations.

Output Files

APR creates output files during and after placing and routing your design, which can include an output design file, a report file, and a message file. All output files are discussed in the following sections.

During processing, APR displays informational messages on the screen. To redirect messages into a file, refer to the "Message File" section below for more information. Table 5-4 provides a summary of output files.

Output Design File

APR generates an LCA design file, *filename*.lca, that contains the new placement and routing information, which is called the output design file. Specify the output design file name on the APR command line.

If you specify an output design name that is the name of an existing LCA design file, APR prompts for permission to overwrite it. You then have the opportunity to specify a different output-design name.

Report File

APR generates a report file, *filename*.rpt, that describes several aspects of the placement and routing activity.

This file is always created, even if APR fails to produce an output-design file, either through detecting an error or because you interrupted the APR execution before it finished. If APR detects an error or issues a warning, the error or warning message appears in the report file. See the "APR Reports" section for more information.

Message File

You can redirect all informational messages that APR produces, which are normally displayed on your screen, to a message file. APR creates this file, *filename*.out, when you use the -o command-line option.

Table 5-4 Output File Summary

Output Files	Extension	Notes
Output design file	.lca	Always created
Report file	.rpt	Always created
Message file	.out	With -o option

During the placement phase, APR displays several columns of information showing the progress of the placement algorithm. This information appears for those who are curious about the workings of the placement algorithm. See "The APR Annealing Progress Messages" section for a complete description of this information.

APR Reports

APR always produces a report file that has the same name as the output design file but with an extension of .rpt. This report file contains useful information about the final state of your design when APR finishes its processing. The report file is organized in the following manner.

- Header information
- Final results summary
- Unrouted nets listing
- Net routing order
- Block placement and pin swapping table
- Net flags table
- Block flags table
- Location flags table
- Net delay table

Header Information

The header information tells what files APR used and what options were specified on the command line. For example, a header information section might appear as shown following.

```
AUTOMATED PLACE AND ROUTE PROGRAM -- Version 5.00
Copyright (c) 1986-1994, by Xilinx, Inc. All Rights Reserved.
Wed Jan 5 09:53:32 1994
 Input Design File: rob_dr3.lca
         Part Type: 3090pg175
 Guide Design File: (none)
    Schematic File: (none)
  Constraints File: rob_dr3.cst
           Options: -a1 -l -r3 -s1564
Output Design File: rob_k1.lca
       Report File: rob_k1.rpt
      Message File: rob_k1.out
       Speed Grade: -100
```

The header section of the report file provides information about the particular run of APR. In this case, APR was run on the input design file rob_dr3.lca without a schematic constraints file, but with a user constraints file rob_dr3.cst. APR generated three files: an output design file, rob_k1.lca; a report file, rob_k1.rpt; and a message file, rob_k1.out.

The first few lines identify the version of APR and the date and time when APR was run. The rest of the header information section lists the files used and the options specified on the command line.

APR always includes the seed used for the pseudo-random number generator, followed by the -s, in the Options line of the report file header. You can use this seed in a subsequent iteration to repeat the same results if there is no change to the design file or constraints files.

Final Results Summary

The final results summary might look like the following example.

```
FINAL RESULTS:
+-------------------------------------------------------------------+
|  Total:                                                 |  Unrouted:    |
|  Blks (CLBs IOBs TBUFs PLUPs CLKs OSCs)  Pins  Nets  |  Loads  Nets  |
| | 448   262   96    88     0    2    0   1860   444  |      2     2  |
+-------------------------------------------------------------------+
```

In this particular design, a total of 448 blocks and 1860 pins were used in routing 444 nets. In this APR run, two load pins on two nets were not successfully routed.

Unrouted Nets Listing

This list contains the nets that APR could not completely route. It is useful when going back to interactively route these unrouted nets. The following example shows the list format. There is an entry for each unrouted net. An entry in the example consists of the net name, d4m; the source pin of that net, d4.X; and the load pins the net failed to route to, MD4.O. The names of the source and load pins consist of the block names and the pins on those blocks the net was to connect to.

```
Nets with Unrouted Load Pins
+-------------------------------------------------------+
| Net d4m (source pin = d4.X)has 1 unrouted pin (out of 2)|
|      MD4.O                                            |
| Net s16q(source pin=s16.Y)has 1 unrouted pin(out of 4)--|
|     AS16.O                                            |
+-------------------------------------------------------+
```

Net Routing Order

This section lists the nets in the order APR routes them.

Block Placement and Pin Swapping Table

The block placement and pin swapping table has three columns. The first column contains the names of the blocks. The second column contains the names of the locations occupied by the blocks. The third column lists the pins for each block that were swapped to improve routability.

If two pins are swapped with each other, they are shown as a pair of pin names separated by a double arrow (↔). Otherwise, the pin names are separated with a single arrow (→), indicating that one pin was moved onto a different pin.

For example, in the following block placement table, the X and Y pins on block D0, location AH, were swapped with each other, while on block D3, location EH, pin A was moved to pin B, and pin C was moved to pin D.

```
Final Placement:
+--------------+------------+-----------------------------+
| Block Name   |  Location  |   Moved or Swapped Pins     |
+--------------+------------+-----------------------------+
|     DB0      |     P6     |                             |
|     DB1      |     P5     |                             |
|     DB2      |     P4     |                             |
|     DB3      |     P3     |                             |
|     D0       |     AH     |   X <-> Y                   |
|     D1       |     BH     |   X <-> Y, C <-> D          |
|     D2       |     CH     |                             |
|     D3       |     EH     |   A -> B, C -> D            |
+--------------+------------+-----------------------------+
```

Net, Block, and Location Flags Tables

The block, location, and net flag tables indicate which blocks, locations, and nets have been flagged or weighted. The following tables illustrate examples of all three.

```
Flagged Nets:
+--------------+----------------------------------------+
| Net Name     | Flags                                  |
+--------------+----------------------------------------+
| datena       | Critical Weight=10                     |
| ntreset      | Normal Weight=3                        |
| datbus       | Longline                               |
+--------------+----------------------------------------+

Flagged Blocks:
+--------------+----------------------------------------+
| Block Name   | Flags                                  |
+--------------+----------------------------------------+
| clkgen       | Locked                                 |
| phix         | Locked External                        |
| abupdate     | Locked                                 |
+--------------+----------------------------------------+

Flagged Locations:
+--------------+----------------------------------------+
| Location Name| Flags                                  |
+--------------+----------------------------------------+
|            P6| Prohibited                             |
|            P5| Prohibited                             |
|            P4| Prohibited                             |
+--------------+--- ------------------------------------+
```

Net Delay Table

The net delay report consists of five columns: net status, net name, source pin location, delay, and load pin location. An example of part of a net delay report follows.

```
Net Delays: (using -125 Speedgrade)
------------------------------------------------------------
  Net Name       Source Pins        Delay     Load Pins
------------------------------------------------------------
M--- BIT04 . . . .TBUF.HC.1.O . . . ~12       P48.O
                  TBUF.HD.1.O
                  TBUF.HF.1.O
-L-- BIT05 . . . U GE.X
S--- BIT06. . . . . . . . . . . . . ***       FG.A
                                    ***      FH.A
------------------------------------------------------------
```

Net Status

The Net Status column in the Net Delay Table has no column header and is four characters wide. The letters in Table 5-5 might appear in a net status column.

Table 5-5 Net Status

Net Status	Description
C	The net is flagged critical
L	The net has no load pin
M	The net has multiple source pins
S	The net has no source pin

Net Name

The Net Name column in the Net Delay Table contains the name of each net.

Source Pins

The Source Pins column in the Net Delay Table contains the physical locations of each source pin on the net followed by the logical block name associated with that location.

Delay

The Delay column in the Net Delay Table contains the net delays, in nanoseconds, to each of the net load pins. If instead of a delay value this column contains three asterisks (***), the corresponding load pin is not routed to the source pin, so there is no delay for that load pin.

Sometimes a delay value is preceded by a tilde, for example, ~12 on P48.0 on net BIT04 in the previous example. The tilde means that the delay is generated by so many concatenated-capacitor elements (resistor or pass-transistor), that the value might be less accurate. Xilinx does not guarantee it as an absolute worst-case value. The number following the tilde is still conservative; most likely the parameter in question is better than this value. If the tilde value is critical to a design, there are two choices.

- Change the layout or routing such that the long uncertain delay is broken up into two non-tilde values, either by passing the net through a bi-directional repowering buffer (BIDI) or through an unused CLB, or by dividing the net into two branches.
- Add 25 percent to the value and ignore the tilde, making the reasonable assumption that this addition compensates for the modeling uncertainty.

XC2064 and XC2018 ACLK delay values, though below 10 ns, are sometimes preceded by a tilde. The tilde can safely be ignored in these cases.

Load Pins

The Load Pins column in the Net Delay Table contains the physical locations of each load pin on the net followed by the logical block name, if any, associated with that location. This table is useful for checking how well APR routed critical nets.

APR Annealing Progress Messages

APR outputs progress messages to the screen while it is running unless you specified the -o option, which directs the output messages to an output file. The following is an example of an annealing progress message.

```
------------------------------------------------------------
               Best   % of   Avg
  Temp  %Chng  Score  Init  Score  %Chng  Stdev  CPU time
------------------------------------------------------------
 766.2  -23%   67603  221%  76794  -7.4%   5.1%  00:06:20
```

The following table provides a brief description of each message.

Table 5-6 Placement Progress Message Descriptions

Heading	Description
Temp	Current temperature of annealing process
%Chng	Percent change in temperature (cooling rate)
Best Score	Best placement score achieved so far
% of Init	Best placement score as a percent of initial placement
Avg Score	Average placement score at current temperature
%Chng	Percent change in average placement score
Stdev	Standard deviation of placement scores at current temperature as a percent of the average placement score
CPU Time	Cumulative execution time in CPU seconds

Placing and Routing Larger Designs

Some designs, in particular those using 80 percent or more of the resources of the larger FPGAs, can be difficult to completely place and route automatically. When your design has a high utilization of FPGA resources, there are two alternate courses of action to help APR produce better results.

First, create constraint files that give APR more information about the design placement and routing desired. This step can be challenging,

especially for designs that are partitioned automatically when translated from a schematic-capture program.

Second, you can obtain improved results by running APR on your design several times, using a different seed for the pseudo-random number generator each time. The random seeds are automatically generated in APRLoop. When APR is run with different seeds, it produces different placements with varying degrees of routability. Running APR successively, or iteratively, on an input design generates a set of output designs, some of which are more completely routed than others.

Running APR Iteratively on the Same Design

The APRLoop program that accompanies APR provides an easy way to run APR iteratively on the same design. Use the following syntax.

`aprloop` ***count options input output***

The iteration *count* is a value between 1 and 99 that specifies the number of times APR should run on the input design. APR runs until exhausting the specified number of iterations or until APR execution terminates abnormally due to multiple control-breaks or running out of disk space. You can terminate APR and APRLoop by typing the appropriate control character. Refer to "Interrupting APR During Processing" section in this chapter for more information.

The other arguments, *input* and *output*, represent the input design name and the output design name. The following is an example of a typical command line.

`aprloop 16 -c juke.cst` *input output*

This command causes APR to run 16 times. APR uses the user constraints file, juke.cst, for each run.

You can use any APR option when running APRLoop. Refer to the "Options" section in this chapter for more information.

Differentiating Between Iterations

APRLoop passes the output design name to APR after performing the following modifications.

- It truncates the output design name to six characters, if necessary.
- It appends a two-digit number to the resulting name.

The number that is appended to the output design name depends on the current iteration number. For example, if you run APRLoop with the following command line.

```
aprloop 3 input output
```

APR executes three times with the following three command lines.

```
apr input output00
apr input output01
apr input output02
```

The following is an APRLoop command line example for the PC.

```
aprloop 3 input mydesign
```

This results in the following three command lines.

```
apr input mydesi00
apr input mydesi01
apr input mydesi02
```

Typically, all command-line options you specify on the APRLoop command line are passed without modification to the APR command lines.

Redirecting the Output

If you specify the -o command-line option, APRLoop redirects all message output for all iterations of APR into the file named with the -o command-line option. APRLoop appends an .out file name extension on the message file specified with the -o command-line option. If you specify a different extension, APRLoop issues an error

message and aborts. The following two examples illustrate what you enter on the command line.

```
aprloop 3 -o output.out input output
```

```
aprloop 3 -o output input output
```

In both examples, APRLoop creates the file named *output*.out. In the second example, APRLoop automatically supplies the .out extension.

PPR

This program is compatible with the following families.

- XC3000A
- XC3000L
- XC3100A
- XC4000
- XC4000A
- XC4000H
- XC5200

The Partition, Place, and Route (PPR) program maps, places, and routes an XC3000A/L, XC3100A, XC4000 (including XC4000A/H), or XC5200 design. It gives you the option of retaining the timing for a design that has been mapped, placed, and routed, then subsequently changed. It re-establishes the same LCA block mapping, placement, and routing for logic elements that have not changed in the design. The LCA design implementation is modified only where necessary to accommodate the design change.

The input design is an XTF file generated by XNFPrep or a MAP file generated by XNFMAP. The output is an LCA design file that is used by the XACT bitstream generator. It can also act as a guide file when you reiterate block mapping, placement, and routing for a design to which minor changes have been made after the previous iteration.

For information on the placement and routing of XC2000 or XC3000 designs, refer to the "APR" chapter of this reference guide.

Design Flow

This section describes PPR's place in the Xilinx design flow. PPR is involved in four separate design flows. In addition, the default PPR flow is also discussed.

Default PPR Flow

This section describes how PPR implements a design by default. You can modify some of this behavior by changing PPR options.

1. For XC4000 and XC5200 designs, gates in the input XTF files are mapped into function generators.

   For XC3000A/L or XC3100A designs, XNFMAP maps gates into function generators and LCA blocks. PPR receives mapped logic in the input MAP file.

2. The function generators, flip-flops, and other blocks that make up the design are assigned to specific locations on the device. The XACT-Performance specifications and a user-specified degree of "effort" guide the placement process.

3. The design is routed with XACT-Performance.

4. PPR generates an LCA and a report (RPT) file for the design.

5. If the timing-insensitive route in step 3 yielded a completely routed design, the design is rerouted according to the XACT-Performance specifications. If the original route yielded any unrouted pins, however, PPR does not continue with this step.

6. PPR generates an LCA and a report (RPT) file for the design routed with XACT-Performance. The LCA file generated in step 4 is now renamed to an LCB file.

The flow chart in Figure 6-1 illustrates this process.

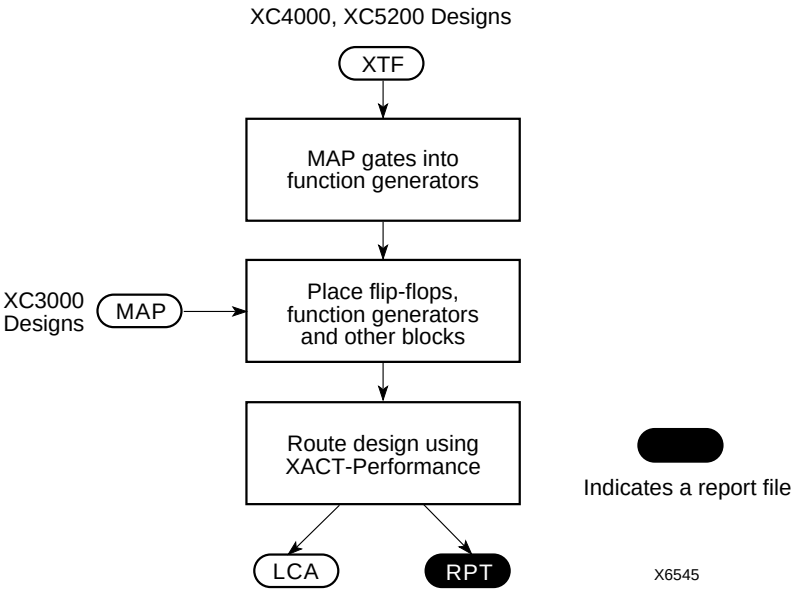


Figure 6-1 Default PPR Design Flow

XC4000 and XC5200 Designs

XC4000 and XC5200 designs are processed by the following programs.

1. A schematic-to-XNF program or synthesis program generates a netlist. This process outputs one or more XNF files.
2. XNFMerge merges all the input XNF files and flattens the design. It outputs an XFF file.
3. XNFPrep performs a design-rule check on the XFF file and trims unused and unneeded logic from the design. It outputs an XTF file.
4. PPR maps, places, and routes the design in the XTF file and outputs an LCA file. If you change any logic and select the Guide option, it maps, places, and routes just the changed portions of the design using the LCA file from the previous iteration as a guide file. You can also change some of the placement and routing of the

design in the XACT Design Editor before using the LCA file to guide PPR.

The following flow chart illustrates this design flow.

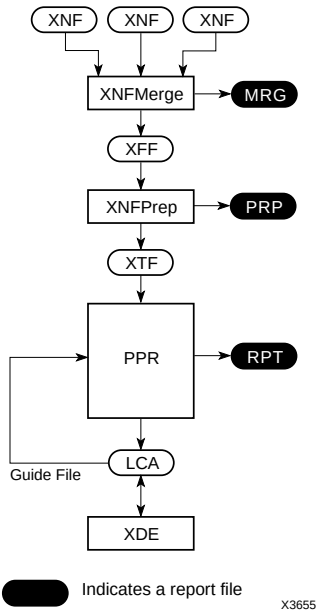


Figure 6-2 XC4000 Design Flow

XC3000A/L and XC3100A Designs

The design flow for XC3000A/L and XC3100A designs is similar to that for XC4000 designs.

1. A schematic-to-XNF program or synthesis program generates a netlist and outputs an XNF file for each hierarchical block, including Xilinx macros.
2. XNFMerge merges all the input XNF files and flattens the design. It outputs an XFF file.
3. XNFPrep performs a design-rule check on the XFF file and trims unused and unneeded logic from the design. It outputs an XTF file.
4. XNFMAP maps the design in the XTF file into CLBs and IOBs,

then outputs a MAP file. If any logic is changed and the Guide (-k) option is selected, XNFMAP uses the PGF file as a guide file to map only the changed portions of the design.

5. PPR places and routes the design in the MAP file and outputs an LCA file. If you change any logic and select the Guide option, it uses the LCA file as a guide file to place and route just the changed portions of the design. XNFMAP has a similar guide option using the PGF file. You can also change some of the placement and routing of the design in the XACT Design Editor before using the LCA file to guide PPR.

This flow is shown in the following figure.

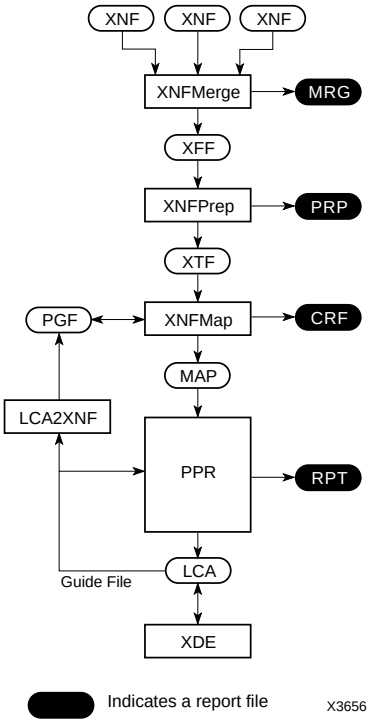


Figure 6-3 XC3000A/L and XC3100A Design Flow

XC4000 and XC5200 Designs with X-BLOX

For XC4000 designs with X-BLOX elements, the design flow is as follows.

1. A schematic-to-XNF program or synthesis program generates a netlist and outputs an XNF file for each hierarchical block, including Xilinx macros.
2. XNFMerge merges all the input XNF files and flattens the design. It outputs an XFF file.
3. XNFPrep performs a design-rule check on the XFF file and trims unused and unneeded logic from the design. It outputs an XTG file.
4. X-BLOX takes the higher-level design contained in the XTG file and generates a gate-level netlist. It instructs MemGen to create RAMs and ROMs as needed. X-BLOX outputs an XG file.
5. XNFPrep checks the design expanded by X-BLOX and trims any unused logic in the expanded design. It outputs a new XTF file.
6. PPR maps, places, and routes the design and outputs an LCA file. If you change any logic and select the Guide option, it maps, places, and routes just the changed portions of the design using the LCA file from the previous iteration as a guide file. You can also change some of the placement and routing of the design in the XACT Design Editor before using the LCA file to guide PPR.

Note: If you have made major structural changes in an X-BLOX design — for example, adding large modules or changing from an 8-bit datapath to a 16-bit datapath — the Guide option is not recommended.

This design flow is illustrated in the next flow chart.

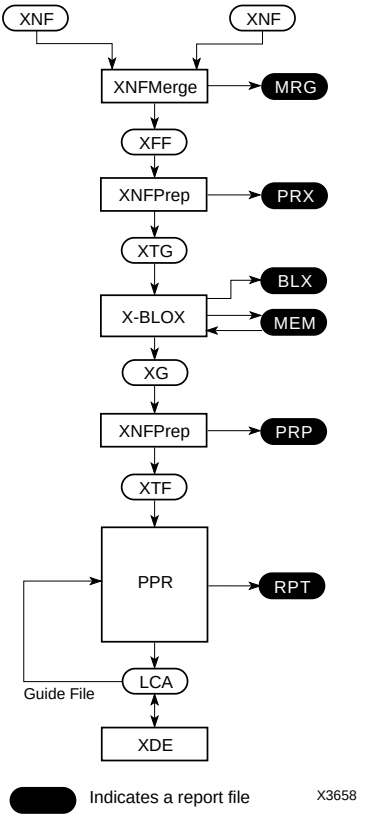


Figure 6-4 XC4000 Design Flow with X-BLOX

XC3000A/L and XC3100A Designs with X-BLOX

X-BLOX elements are also available with XC3000A/L and XC3100A designs, although the design flow is slightly different from that of XC4000 designs.

1. A schematic-to-XNF program or synthesis program generates a netlist and outputs an XNF file for each hierarchical block, including Xilinx macros.
2. XNFMerge merges all the input XNF files and flattens the design. It outputs an XFF file.

3. XNFPrep performs a design-rule check on the XFF file and trims unused and unneeded logic from the design. It outputs an XTG file.
4. X-BLOX takes the higher-level design contained in the XTG file and generates a gate-level netlist. It outputs an XG file.
5. XNFPrep checks the design expanded by X-BLOX and trims any unused logic in the expanded design. It outputs a new XTF file.
6. XNFMAP maps the design in the XTF file into CLBs and IOBs, then outputs a MAP file. If any logic is changed, it uses the PGF file as a guide file to map only the changed portions of the design.
7. PPR places and routes the design in the MAP file and outputs an LCA file. If you change any logic and select the Guide option, it uses an LCA file as a guide file to place and route just the changed portions of the design. XNFMAP has a similar guide option using the PGF file. You can also change some of the placement and routing of the design in the XACT Design Editor before using the LCA file to guide PPR.

Note: If you have made major structural changes in an X-BLOX design — for example, adding large modules or changing from an 8-bit datapath to a 16-bit datapath — the Guide option is not recommended.

The XC3000A/L and XC3100A design flow with X-BLOX is shown in Figure 6-5.

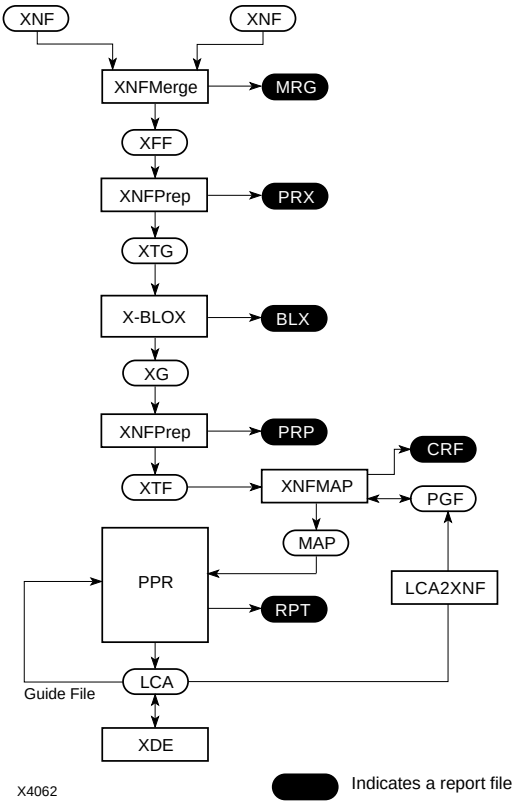


Figure 6-5 XC3000A/L and XC3100A Design Flow with X-BLOX

Files

Input Files

PPR accepts one of the following two files as the input design:

- *design*.xtf is an XTF file output by XNFPrep for an XC4000 or XC5200 design. You do not need to specify the .xtf extension, since an XTF file is the default for XC4000 and XC5200 designs.
- *design*.map is a MAP file output by XNFMAP for an XC3000A/L or XC3100A design. You must specify the .map extension.

PPR also accepts either or both of the following optional input files:

- *guide*.lca is a guide file output by a previous iteration of PPR that may have been edited in XDE. PPR uses this file to re-establish the same LCA block mapping, placement, and routing for logic elements that have not changed in the new iteration.
- *design*.cst is the optional constraints file that PPR automatically reads, if it is present in the design directory. Refer to the "Constraints File Syntax" section at the end of this chapter or to the *Libraries Guide* for information about the constraints file syntax.

Output Files

PPR outputs the following files:

- *design*.lca is the placed and routed LCA file that PPR generates. It is given the input design name with an .lca extension unless you use the Outfile option, described in the "PPR Options" section of this chapter, to specify a different output file name.
- *design*.lcb is a backup copy of the previous LCA file.
- ppr.log contains all information output to the screen during PPR execution. You can specify a log file name other than ppr.log by using the Logfile option described in the "PPR Options" section.
- *design*.rpt is a report file that includes design statistics and device utilization, program run information, the location of I/O pins, and other information. You can use this information to help analyze your design. This file is given the input design name with an .rpt extension unless you use the Outfile option described in the "PPR Options" section of this chapter to specify a different output file name.

Guided Design

The term *guided design* refers to the process in which a previously implemented design — also known as a guide file — is used to guide mapping, placement, and routing. Guided design allows logic to be modified or added to a design while preserving the layout and performance that have been previously achieved.

Types of Guided Design

There are three ways that you might use guided design:

- Iterative design — If a logic change is required in a design that has already been verified for timing, guided design can minimize the impact of that change on the new layout. This process is known as iterative design. It simplifies the mapping, placement, and routing process, as well as reverifies the design timing.

  In iterative design, the original design is specified as the guide file, and PPR attempts to copy as much of its mapping, placement, and routing as possible. It implements logic that has not been changed using exactly the same LCA resources as in the guide file, which ensures that the timing on those paths is identical. For logic that has been changed, it uses the normal mapping, placement, and routing process.

- Incremental design — You can implement and verify a design in stages using guided design. First PPR maps, places, and routes a single functional block and verifies the timing internal to that block. Then you add a second functional block to the design. PPR maps, places, and routes the design using the first result as the guide file. You then verify the timing of the new logic. By repeating this process, you can build and verify a complete design piece by piece. This type of design is known as incremental design.

- Placement and routing in the XACT Design Editor (XDE) — Guided design also allows you to place and route extremely critical portions of a design manually and then direct PPR to finish the placement and routing. This procedure is as follows.

  a) PPR places the design without performing any routing.

  b) You load the placed but unrouted design into XDE, moving blocks and pre-routing signals as needed. For example, you might align the CLBs associated with a counter in a single column and then route an enable input via a vertical longline.

  c) The pre-placed and pre-routed design is used as the guide file, with the same input design as that used for the previous iteration. Since the input design and the guide file are logically equivalent, PPR copies all of the placement and routing

specified in the guide file and then completes the place and route process.

Obtaining the Best Results from Guided Design

Guided design relies on signal names in order to match logic between the guide file and the input design netlist. For this reason, it is very important to minimize signal name changes when guided design is to be used. Following are some points to keep in mind.

- When the design is flattened by the XNFMerge program, a hierarchical path name is added to the beginning of every signal name. For example, the ENABLE signal inside a symbol named CNTR is resolved to CNTR/ENABLE after XNFMerge. If any new hierarchy is created at the top of the design, the names of every signal underneath are changed, and guided design would not operate on the associated logic. Continuing with the example, if the CNTR symbol is moved underneath a symbol called CONTROL, the signal that was called CNTR/ENABLE in the previous iteration would be called CONTROL/CNTR/ENABLE instead; in guided design, these signals would not be considered the same signal.
- IOBs are matched to the guide file by the name of the signal attached to the I/O pad symbol. To ensure that IOBs are guided properly, do not change the names of these signals.
- Schematic editors typically assign names to unnamed signals. If your schematic editor assigns names consistently, such assignment is no problem. However, if the names of unnamed signals are changed each time the design is processed, you should assign a name to every signal in the design.
- Synthesis software creates new nodes in a design and therefore new signal names. If changes are made to logic inside a synthesized module, names may no longer match those in the guide file. For best results with guided design, avoid changing synthesized logic as much as possible.
- XDE does not represent nodes internal to a CLB as named signals. For example, a signal going from a function generator to a flip-flop in the same CLB does not have a name in XDE. For this reason, XDE can be used to modify placement and routing but should never be used to add or change logic in the design. As a

general rule, any operation that is allowed in XDE's safe mode does not adversely affect guided design.

Guided Design Flow for XC4000 and XC5200 Designs

For an XC4000 and XC5200 design, the guided design process is completely controlled by the PPR program. As shown in Figure 6-2, the guide file is an LCA file generated by a previous run of PPR, which may or may not have been edited using XDE.

Consider an example in which an LCA guide file named original.lca is used to guide the mapping, placement, and routing of a modified version described in new.xtf. Specification of a guide file with the following command automatically causes PPR to use guided design:

```
PPR new guide=original
```

Several PPR options control the guided design process. In most cases, the defaults for these options are satisfactory, so the options need not be explicitly set. See the "PPR Options for Guided Design" section, following, for more information on these options.

Guided Design Flow for XC3000A/L and XC3100A Designs

For an XC3000A/L or XC3100A design, the guided design process is controlled by both the XNFMAP and PPR programs. XNFMAP guides the mapping of the design into CLBs, and PPR guides the placement and routing. As shown in Figure 6-3, the guide file is an LCA file generated by a previous run of PPR, which may or may not have been edited using XDE. The guided design procedure for XC3000A/L and XC3100A designs is as follows.

1. If the PGF file created by XNFMAP on the previous design iteration is available, use it to guide XNFMAP.

   If the PGF file is not available, run the LCA2XNF program on the LCA guide file using the -b option. You should specify the output file name to match the name of the new input design with an extension of .pgf. This output file is the partitioning guide file that XNFMAP uses.

2. Run XNFMAP on the new input design using the -k option, which tells XNFMAP to use the information in the PGF file to guide partitioning.
3. Run PPR on the MAP file generated by XNFMAP, using the Guide option to specify the LCA guide file. The specification of a guide file automatically causes PPR to use guided design.

Consider an example in which an LCA guide file named original.lca is used to guide the mapping, placement, and routing of a modified version described in new.xtf. The following commands implement the steps just given.

```
LCA2XNF -B original new.pgf
XNFMAP -K new
PPR new guide=original
```

Several PPR options control the guided design process. In most cases, the defaults for these options are satisfactory, so the options need not be explicitly set.

PPR Options for Guided Design

The options that control the guided design process are listed in the following table. The use of these options is explained in this section, and reference information on them can be found in the "PPR Options" section of this chapter.

Table 6-1 Guided Design Options

PPR Option	Valid Values	Default Value
guide_blks	all, routed_only	all
guide_routing	all, whole_sigs	all
guide_thru_routes	all, whole_sigs, none	whole_sigs
lock_routing	all, whole_sigs, none	whole_sigs
guide_only	true, false	false

In most cases, the default values of these options is sufficient, but this section explains when it may be helpful to use alternative settings.

Iterative Design

For iterative design, that is, changing logic in a design that has already been implemented and verified, the default settings of the guide options are usually appropriate. Using these defaults, PPR copies all of the routing in the guide file for each signal that it can match in the new input design. However, if a signal has changed in some way — that is, if one or more of the pins on that signal do not match between the guide file and the input design — PPR reroutes that signal if it can improve the timing. The Whole_sigs default setting of both the Lock_routing and Guide_thru_routes options locks only the routing of whole, or intact, signals.

Locking Partial Routes

To prevent PPR from rerouting any guided signals, even those which have changed in some way, set both Lock_routing and Guide_thru_routes to All. However, if any pins on a signal do not match between the guide file and input design, it may be more difficult for PPR to connect the changed pins to the guide file routing than it would be to reroute the entire signal. For this reason, using the All setting can result in worse performance than using Whole_sigs, which allows PPR to reroute signals where needed. The "Lock_routing and Guide_thru_routes Options" section later in this chapter describes when it might be desirable to set these two options to different values.

Incremental Design

The following flow chart illustrates the design flow for incremental design, in which you build up a design in stages by adding new functional blocks. This design flow also applies to iterative design, described in the preceding section.

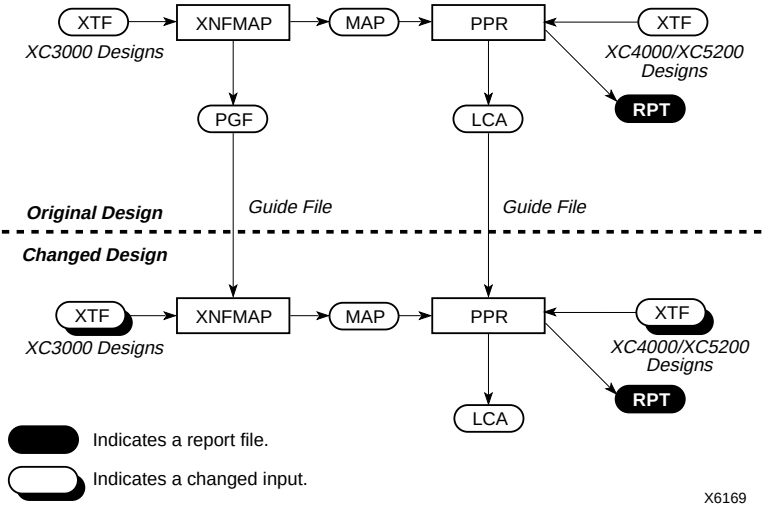


Figure 6-6 Incremental Design

For incremental design, all of the default settings of the guide options are appropriate. Using these defaults, PPR copies all of the routing in the guide file for each signal that it can match in the new input design. Furthermore, it does not change the routing found in the guide file on any signal when all pins in the guide file match those of the input design. With these defaults, all signals that are completely internal to a functional block are preserved, provided that the logic of that block is unchanged. The signals that connect the new logic to that in the guide file is routed or possibly rerouted by PPR.

In some situations, you may want to control which signals PPR is allowed to reroute. For more information, see the "Locking Partial Routes" section earlier in this chapter.

When you have made small changes to a design, or when the design density is not at maximum, use the options and settings shown in the following table.

Table 6-2 Default Incremental Design Options

Option	Setting
guide_blks	all
guide_routing	all
guide_thru_routes	whole_sigs
lock_routing	whole_sigs

When larger changes to the design are involved, or when the density of the design is at maximum, use the same options with the settings shown in Table 6-3.

Table 6-3 Other Incremental Design Options

Option	Setting
guide_blks	all
guide_routing	whole_sigs
guide_thru_routes	none
lock_routing	none

Placement and Routing in XDE

The design flow involved in using a guide file output by a previous iteration of PPR, editing it in XDE, and allowing PPR to complete the placement and routing is shown in the following illustration.

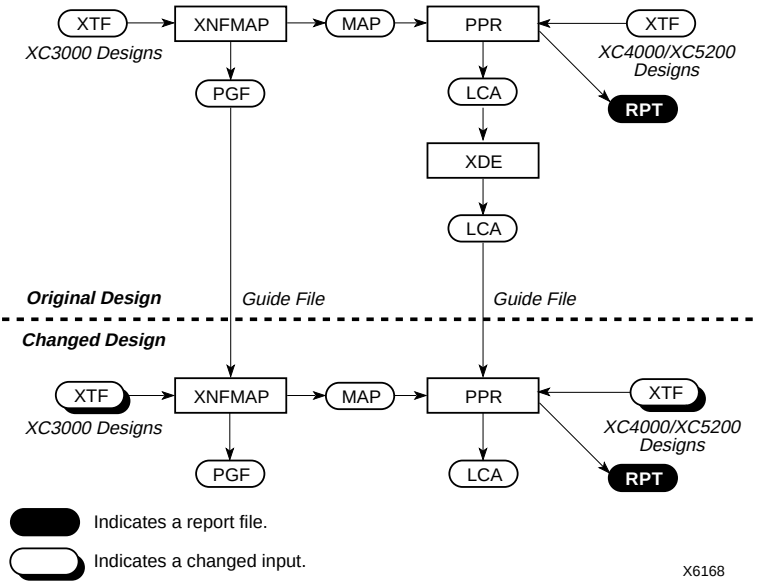


Figure 6-7 Guided Design with Manual Placement and Routing

Suppose that you use XDE to floorplan a design. Pre-placement and/or pre-routing has been completed for some of the blocks and nets. PPR can be expected to complete most of the design automatically, given this guidance.

Note: Set the Route option to False if you are planning to edit the design in XDE so PPR generates the LCA file as fast as possible.

If you use guided design to preserve manual placement and routing done in XDE, some of the default settings of the guide options are not appropriate.

By default, PPR copies the placement of every block in the guide file that it can match to the input design. When preserving manual placement and routing, the guide file and input design are logically equivalent, and PPR can match everything. However, it is likely that only a portion of the design was pre-placed in XDE and that the placement of the other blocks was not optimized. If PPR were to simply copy the placement of these blocks from the guide file, the overall placement would suffer.

To direct PPR to guide only those blocks that have routing connected to them in the guide file, set the Guide_blks option to Routed_only. With this option set, blocks that have no routing attached to them are subject to the normal PPR placement process.

In the guided routing process, PPR copies all of the routing in the guide file for each signal that it can match in the new input design. However, by default, PPR is allowed to reroute signals on which one or more pins have changed or been moved. If only a portion of a signal was pre-routed in the guide file, that routing may be changed.

As an example, suppose that in the guide file, all of the CLBs associated with a register are aligned in a single column and the clock-enable input pins are routed onto a vertical longline: since the source block of the enable signal may move, only the load pins of that signal are pre-routed. PPR most likely moves the source pin during the placement process, since that block was not routed. Because the placement of the source pin was not matched to the guide file, the guide routing process would, by default, copy the routing on the load pins but would consider the entire signal eligible to be rerouted.

To prevent PPR from changing any of the routing specified in the guide file, set both the Lock_routing and Guide_thru_routes options to All. In the example just given, these options would cause the routing of the load pins to be preserved, even though the source pin was not routed in the guide file.

One other useful option for guiding a manually edited design is Guide_only. If it is set to True, PPR guides the placement, places all unguided blocks, and guides the routing, but it does not route unguided signals. You can use Guide_only to guide a new design iteration from a previous one and use manual placement and routing done in XDE. First, run PPR on the new input design, using the previous design iteration as the guide file and setting Guide_only to True. This step produces a design that is guided against the last iteration; the LCA file is then loaded into XDE and pre-placed and routed as desired. The modified LCA file acts as a guide file to another run of PPR on the same input design but this time with Guide_only set to False. This process produces an LCA file that is guided against the manual work done in XDE, as well as against the previous design iteration.

The following table summarizes the options and settings to use with guided design to preserve the manual placement and routing done in XDE.

Table 6-4 XDE Placement and Routing Options

Option	Setting
guide_blks	routed_only
guide_routing	all
guide_thru_routes	whole_sigs
lock_routing	whole_sigs

Lock_routing and Guide_thru_routes Options

In general, the Lock_routing and Guide_thru_routes options both control whether or not PPR is allowed to reroute signals that were routed in the guide file, and these two options are usually set to the same value. However, there is a subtle difference between the two options that may occasionally be important.

The Lock_routing option simply controls whether or not the PPR router is allowed to unroute the guide file routing on a signal and reroute that signal to improve the timing. You can prevent it from doing so either for all signals with Lock_routing=All or only for signals that were completely guided with Lock_routing=Whole_sigs. A third setting, Lock_routing=None, allows PPR to unroute and reroute any signal.

However, in order to improve routability and timing, PPR may route some signals through a CLB or other block; this type of routing is known as a through-route. Although it improves routing, it requires a function generator to pass the signal from one side of the CLB to the other. In the XC4000 family, a flip-flop output pin can be used to pass the signal, since the XQ or YQ pin moves the signal out of the CLB. Similarly, in the XC5200 family, a signal may be routed from the DI pin of an LCE (in a CLB) to the DO pin. If a signal that is routed in this fashion is to be guided, certain CLB resources must be left unused during the placement process. Thus, the guide file routing is favored over the placement of the unguided logic.

The Guide_thru_routes option controls which through-routes are preserved, using the All and Whole_sigs settings similar to those of Lock_routing. With Guide_thru_routes=All, PPR preserves all through-routes. With Guide_thru_routes=Whole_sigs, PPR preserves through-routes only on signals that were completely matched in the input design; for other signals, those CLB resources can be used for other logic. As with Lock_routing, the None setting allows PPR to use all through-route resources for other logic.

Note: Using different settings for Lock_routing and Guide_thru_routes may be desirable when a design uses a high percentage of the LCA device resources. If you add new logic to the input design and use a guide file, the CLB resources used for through-routes may make placement of that new logic more difficult. By using Lock_routing=All and Guide_thru_routes=None, those signals that do not use through-routes are preserved, but the CLB resources used for other signals are available for new logic.

Guided Design and XACT-Performance

If guided design is used and XACT-Performance timing requirements have been specified, the information in the guide file normally takes precedence over the specified path delay requirements. If a given path is placed and routed in the guide file so that it exceeds the specified delay, PPR does not normally attempt to improve the timing on this path, and the specification is not met. To direct PPR to attempt some improvement on such paths, set the Lock_routing and Guide_thru_routes options to None; these settings allow the routing on these paths to be discarded and rerouted if the timing can be improved. However, the placement is still limited by the placement in the guide file.

Guided Design and Constraints

If guided design is used and any constraints have been specified either in the schematic or in a constraints (CST) file, the constraints normally take precedence over the guide file information. However, there are options that allow specific categories of constraints to be ignored, allowing the guide file to be used. These options are the following follows.

- For an XC4000 design, if the PPR Ignore_maps option is set to

True, PPR ignores all FMAP and HMAP symbols when grouping gates into function generators.

- For an XC5200 design, if the PPR Ignore_maps option is set to True, PPE ignores all F5MAP and FMAP symbols when grouping gates into function generators.
- For an XC3000A/L or XC3100A design, if the XNFMAP -m option is used, XNFMAP ignores all CLBMAP symbols when mapping gates and flip-flops into CLBs.
- If the PPR Ignore_xnf_locs option is set to True, PPR ignores all LOC constraints specified in the schematic. To ignore location constraints specified in a CST file, modify or rename the CST file.
- If the PPR Ignore_rlocs option is set to All, PPR ignores all relative location (RLOC) constraints specified in the input design. The constraints can come either from the schematic or from a synthesis program such as X-BLOX. If an XC4000 design contains any carry logic (CY4) symbols, the RLOC constraints on these symbols are not ignored. If an XC5200 design contains any carry mux or F5 mux symbols, then user-defined RLOC constraints are ignored and PPR will generate device-specific structures for these symbols.

See the "PPR Options" section later in this chapter for more information on the PPR options just listed. Also refer to the "XNFMAP" chapter in this reference guide for more information on the -m and other options.

XC3000A/L and XC3100A Guided Design with PPR

PPR guided design provides the same functionality as the APR guided design process for XC3000, although the method is slightly different. This section describes the correlation between the APR and PPR procedures.

For iterative and incremental design, the basic behavior of the PPR Guide option is the same as the APR -g option: it uses a guide LCA file to control the placement and routing of a changed input design. Although the input design to PPR is not an LCA file, the result is the same, since the mapping process is also guided by XNFMAP for XC3000A/L and XC3100A, or by PPR for XC4000.

For pre-placement and pre-routing work done in XDE, the PPR Guide option is used in place of the APR -p option. Rather than using the pre-placed and pre-routed LCA file as the input design, as is done with Apr -p, it is specified as the guide file. This guide file is generated by a previous run of PPR and is edited in XDE to specify placement and routing information. The input design to PPR on the guided run is the same netlist as that used on the previous run. You might use the Guide_blks=Routed_only option to restrict guided placement to routed blocks, as Apr -p would do; see "PPR Options for Guided Design" earlier in this chapter for more information on this option.

Constraints

You can set a number of optional constraints to control the PPR program. They control the placement of specified logic and the routing of certain nets. These constraints are listed and described in the *Libraries Guide*.

You can specify these constraints either on the schematic or in a CST file. Schematic constraints are attached to symbols or nets and are passed into the XNF file by the schematic-to-XNF translator. Constraints specified in a CST file are read directly by PPR.

By default, PPR reads the constraints file that carries the same name as the input design with a .cst extension; however, you can specify a different constraints file name with the Cstfile option, described in the "PPR Options" section of this chapter.

The exact syntax of all constraints available in the CST file is given in the "Constraints File Syntax" section at the end of this chapter and in the *Libraries Guide*. That guide also gives a number of examples of placement constraints. Each statement is terminated by a semicolon (;). No continuation characters are necessary if a statement exceeds one line, since a semicolon marks the end of the statement. Lines that begin with the pound sign (#) character are comments. Statements do not have to be placed in any particular order in the CST file.

The constraints in the CST file and the constraints in the schematic or XNF file are applied equally; it does not matter whether a constraint is entered in the schematic or in the CST file.

If by mistake two or more elements are locked onto a single location, PPR detects the conflict and stops processing so that you can correct the mistake.

How to Use PPR

This section gives examples that show how to perform PPR's major functions. For detailed information on the options that implement these functions, see the "PPR Options" section later in this chapter.

Invoking PPR

You can execute PPR from the XACT Design Manager (XDM) or from an operating system prompt. From XDM, you can execute PPR from either the menu or the command line. Executing PPR from the operating system provides more control over PPR functions than executing PPR from XDM.

From XDM

To execute PPR from the XDM menus, follow these steps:

1. Ensure that the correct family is specified in the Family field in the lower left corner of the screen; also ensure that your working directory is specified in the Directory field.
2. Click the left mouse button on **`PlaceRoute`** and then on **`PPR.`**
3. Click on the desired XTF or MAP file from the list of files that appears. If there are no XTF files in the directory in which PPR is invoked, you can specify a path name and a file name at the `select input file:` prompt.
4. Click on the desired options from the resulting list; if you do not see the option that you are interested in, click on **`(DISPLAY ADDITIONAL OPTIONS)`**.
5. After you have set the options, select **`Done`** to run PPR.

A window now appears in which PPR's processing is displayed.

To cancel a transaction at any point before you run PPR, click on Cancel; to terminate PPR's processing once it has started, press Control Break on PCs or Control C on workstations. If the run aborts

during processing, the error is displayed at the top of the XDM screen on workstations and at the bottom of the XDM screen on PCs.

From the Command Line

Use the following syntax to execute PPR from the XDM command line or the operating system prompt.

ppr *designname* [*options*]

Designname is the input design file. For an XC4000 or XC5200 design, an .xtf extension is assumed; for an XC3000 design, a .map extension is assumed.

Options can be any number of the PPR options listed in the "PPR Options" section of this chapter and their respective values. They do not need to be listed in any particular order. Separate multiple options with spaces. The syntax for each option is *option=value*.

Typing in Ppr with no arguments or Ppr -help displays the syntax of the Ppr command and an explanation of *design* and the Parttype, Estimate, and -Helpall options.

The DOS command line only accepts 127 characters per command. If this limit becomes a problem, use a parameter file.

A parameter file is a separate file that contains all the PPR commands that you intend to use in any one session. Using this file saves you from having to enter a number of options at the system prompt every time that you execute PPR. A parameter file is a text file, created with a text editor, containing a list of desired options and their respective values, as in the following example.

```
parttype=4005pg156
estimate=true
ignore_xnf_locs=io
```

Use the following syntax to execute PPR with a parameter file named *textfile*.

ppr *designname* **paramfile=***textfile* [*options*]

You can specify additional options at the command line whenever a parameter file is used; these override similar parameters specified in the parameter file.

Running PPR in XMake

XMake automatically runs PPR as part of its processing. See the "XMake" chapter of this reference guide for instructions on invoking XMake.

Suspending PPR Operation

You can suspend PPR operation during the placement or routing phase by entering Control Break on PCs or Control C on workstations. Specifically, you can suspend PPR operation when the following message appears on the screen:

```
+ Suspension enabled.
```

You cannot suspend PPR operation when the following message appears on the screen:

```
+ Suspension disabled.
```

When the ability to suspend PPR's operation is enabled, the following screen appears after PPR receives the interrupting keystroke. The screen may appear several minutes later, because interrupt messages are polled only at intervals. Pressing Control Break or Control C a number of times does not speed the process. Here is an example of this screen:

```
*** User Interrupted Execution with Ctrl-C ***

Select one of the following options:

    S - Save the current results and then quit
    the program
    C - Continue running the program; ignore
    this interrupt
    Q - Quit the program immediately

Option:
```

As this screen indicates, you have three options once you interrupt PPR's processing:

- Entering *s* stops PPR operation, saves the result (the current placement in the placement phase and the current routing in the routing phase) and terminates the interrupted PPR task. The placement phase is terminated after saving the current results and routing is initiated on that placement. However, when you interrupt PPR in the routing phase, entering *s* saves the current routing result and exits PPR since it is PPR's last phase.
- Entering *c* continues with the interrupted operation.
- Entering *q* exits PPR without saving the current results.

This flexibility is useful in running PPR for an extended period and then interrupting it when you are ready to save the current results. When saving the current routing results, you can resume routing by using the output LCA file as a guide file.

Using xactinit.dat Files

As an alternative to specifying options on the PPR command line or in a parameter file, as just described, most PPR options can also be specified in a file called xactinit.dat. The xactinit.dat file is similar to a parameter file, except that it is automatically used every time that PPR is run, so that you do not have to specify the file name each time.

The syntax of options in the xactinit.dat file is slightly different from the command line syntax. For this reason, the xactinit.dat syntax is shown for each option listed in the "PPR Options" section of this chapter.

The xactinit.dat file is a normal text file that you can create with any ASCII text editor. Although the name of the file must always be xactinit.dat, it can be located in any of three directories. PPR reads the xactinit.dat file in these directories in the order shown and uses the options specified in all three files. The xactinit.dat files are read as follows:

1. PPR first reads the global xactinit.dat file in the data subdirectory of the directory specified by the XACT environment variable, or $XACT/data. The options specified in this file are used by every run of PPR. In a multiple-user environment, options specified here apply to all users.

2. PPR next reads the user xactinit.dat file in your home directory, specified by the HOME environment variable or simply C:\ on the PC. The options specified in this file are used every time that you run PPR. In a single-user environment, this file is essentially no different than the global xactinit.dat file.
3. Lastly, PPR reads the local xactinit.dat file in the directory where PPR is run. The options specified in this file are used only when PPR is run from that directory. Typically, design-specific options are specified here.

If the same option is specified in more than one xactinit.dat file, the one read last is used. For example, if Guide_routing=All is in the global xactinit.dat file but Guide_routing=None is in the local xactinit.dat file, Guide_routing=None is used.

You can add a comment to an xactinit.dat file by placing a pound sign (#) at the beginning of the line.

Setting General Processing Options

The following examples show how to use a few basic PPR options. The name of the design used is "top."

Changing Output LCA and RPT File Names

By default, the output LCA and RPT files have the same name as the input design file. To specify a different name, invoke PPR as follows:

```
ppr top outfile=newtop
```

Changing Log File Name

By default, PPR echoes its screen output to the ppr.log file. To specify a different name, invoke PPR as follows:

```
ppr top logfile=pprtop.out
```

Determining Device Utilization

If you need an estimate of device utilization for a design that is not complete, PPR can generate a report file with these statistics when you enter the following:

```
ppr top estimate=true
```

Running PPR in this fashion does not generate an LCA file output but only a report file. If the design contains any sourceless or loadless signals that should be preserved, the XNFPrep program should be run with the Savesig=True option before running PPR, as shown in the "XNFPrep" chapter of this reference guide.

Placing and Routing a Partial Design

To support incremental design, PPR can place and route a partial design, which may contain signals that are still sourceless or loadless. To tell PPR that additional logic is still to be added to the design, you must set the Complete option as follows.

```
ppr top complete=false
```

With this setting, PPR preserves any sourceless or loadless signals in the design and avoids routing through resources that may be needed for future logic. The default of the Complete option is True.

Controlling Constraints

The following examples show how you can manipulate constraints with PPR options.

Specifying an Alternate CST File

By default, PPR reads the CST file that has the same name as the input design. You can use the Cstfile option to specify a different file. For example, if the constraints file is called pinout.cst, invoke PPR as follows:

```
ppr top cstfile=pinout
```

Ignoring MAP Symbols

For an XC4000 or XC5200 design, direct PPR to ignore all F5MAP, FMAP, and HMAP symbols as follows:

```
ppr top ignore_maps=true
```

For an XC3000A/L or XC3100A design, CLBMAP symbols can be ignored by using the -m option in XNFMAP. See the "XNFMAP" chapter of this reference guide for more information.

Ignoring Absolute Location Constraints

Absolute location constraints are specified either as LOC parameters on individual symbols or by an RLOC_ORIGIN parameter on an RLOC set. PPR can ignore all such constraints in the input design when you enter the following:

```
ppr top ignore_xnf_locs=all
```

To ignore all location constraints except those on I/O pads, invoke PPR this way:

```
ppr top ignore_xnf_locs=interior
```

To ignore only the location constraints on I/O pads, invoke PPR as follows:

```
ppr top ignore_xnf_locs=io
```

Ignoring Relative Location Constraints

To ignore any relative location (RLOC) constraints in the input design, use the following syntax:

```
ppr top ignore_rlocs=true
```

However, if the design contains any CY4 symbols or any Xilinx macros that are implemented using carry logic, the RLOC constraints are not ignored on those CY4 symbols.

Controlling Placement and Routing

The following examples show how the place and route process can be controlled through PPR options.

Setting Level of Placement Effort

By default, PPR attempts to balance good-quality placement with a reasonable execution time. However, it can work harder on placement at the cost of additional run time. This level of effort is controlled by the Placer_effort option, which takes an integer value between 1 and 5 to determine how hard PPR should work during placement. The default is 2. For example, to direct PPR to work as hard as possible on placement, invoke the program as follows.

```
ppr top placer_effort=5
```

Although the run time is extended, the placement results are usually improved over the default. Similarly, to direct PPR to do a fast, lower-quality placement, invoke the program as follows.

```
ppr top placer_effort=1
```

Setting Level of Router Effort

By default, PPR attempts to balance good-quality routing with a reasonable execution time. However, it can work harder on routing at the cost of additional run time or vice-versa. This level of effort is controlled by the Router_effort option, which takes an integer value between 1 and 4 to determine how hard PPR should work during routing. The default value for the Router_effort option is 2, which provides a good balance between execution and quality for most designs. To direct PPR to do a fast, lower-quality route, invoke the program as follows.

```
ppr top router_effort=1
```

Conversely, to direct PPR to work harder to achieve routing quality, invoke the program as follows.

```
ppr top router_effort=3
```

Although the run time is extended, the routing results are usually improved over the default. Setting the Router_effort option to 4 causes the router to work the hardest at achieving a quality result. However, this setting should be used with caution, since it can significantly increase the execution time.

```
ppr top router_effort=4
```

Controlling the Timing-Insensitive Quick Route

By default, PPR routes the design while considering the XACT-Performance timing requirements. For a low-speed design, PPR can perform only the timing-insensitive routing if you invoke the program as follows.

```
ppr top timing=ignored
```

The Timing=Ignored option completely turns off XACT-Performance.

On the other hand, if XACT-Performance timing requirements need to be considered only when the design is routable, you should run PPR as follows.

```
ppr top timing=when_routable
```

Controlling Through-Routes

In order to improve routability and timing, PPR may route some signals through a CLB or other block. This type of routing is known as a through-route. Although this technique usually improves the routing, PPR allows this feature to be limited or disabled entirely.

For incremental design, that is, building up a design by placing and routing functional blocks one at a time, it makes sense to prevent PPR from routing through blocks that do not contain any logic, since they may be needed for a functional block that is still to be added to the design. To direct PPR to avoid routing through such empty blocks, invoke the program as follows.

```
ppr top route_thru_blks=limit
```

In this mode, PPR may still route through blocks that are already used for logic.

PPR can also be directed to avoid through-routing completely for XC3000A/L and XC3100A designs, but doing so may make the design more difficult to route and may degrade the overall design timing. However, to direct PPR to avoid routing through any block, invoke the program as follows.

```
ppr top route_thru_blks=never
```

There are some situations in which it is impossible to route without using through-routes. If you use the Route_thru_blks=Never option in these cases, PPR cannot route the design completely. The Never option is not available for XC4000 and XC5200 designs.

Routing Through Global Buffers

Rather than routing only through general interconnect, PPR can route through unused global buffers — BUFGS in (SWAP) XC4000, BUFG in XC5200,or GCLK and ACLK in XC3000 — if doing so seems more efficient. Use the following syntax.

```
ppr top route_thru_bufg={ok|never}
```

If the Route_thru_bufg option is set to Ok, PPR routes signals through the unused buffers noted above. If it is Never, PPR does not use global buffers needed for later design iterations. The default value depends on the value of the Complete option. If Complete is set to True, Route_thru_bufg defaults to Ok; if Complete is set to False, Route_thru_bufg is set to Never.

Controlling Guided Design

The following examples using the "new" input file show how you can control guided design through PPR options.

Specifying a Guide File

If an LCA guide file named original.lca is to be used to guide the placement and routing of a modified version described in new.xtf, you would invoke PPR as follows.

```
PPR new guide=original
```

The specification of a guide file automatically causes PPR to use guided design.

Guiding Placement of Routed Blocks

By default, PPR copies the placement of every block in the guide file that it can match to the input design. When preserving manual placement and routing done in XDE, the guide file and input design are logically equivalent, and PPR is able to match everything. However, if only a portion of the design was pre-placed in XDE, and the placement of the other blocks was not optimized, the overall placement would suffer if PPR guided the placement of every block in the guide file.

To direct PPR to guide *only* those blocks that have routing connected to them, use the following syntax.

```
ppr new guide=original guide_blks=routed_only
```

Guiding Routing of Unchanged Signals Only

By default, PPR copies the routing of any signal in the guide file for which it can match any pins to the input design. However, it may be desirable to ignore routing in the guide file for signals on which pins

have been added or removed. To direct PPR to ignore the guide file routing on such signals, invoke the program as follows.

```
ppr new guide=original guide_routing=
whole_sigs
```

This option causes PPR to guide only the routing of signals for which all pins in the guide file are matched to the input design: these are the whole, or intact, signals.

Locking Routing from Guide File

By default, PPR does not change the routing found in the guide file on any signal when all pins in the guide file match those of the input design. However, signals that are not completely matched may be rerouted, if PPR can improve on the routing of the overall signal. To prevent PPR from changing any routing that was copied from the guide file, invoke the program as follows.

```
ppr new guide=original lock_routing=all
guide_thru_routes=all
```

To allow PPR to reroute any signal, even if routing was copied from the guide file, invoke the program as follows:

```
ppr new guide=original lock_routing=none
guide_thru_routes=none
```

As noted in the "Controlling Through-Routes" section earlier in this chapter, PPR may route signals through CLBs or other blocks. In order to guide such signals, those CLB resources must not be used for new logic during the placement process. By default, PPR sets these resources aside for signals when all pins in the guide file are matched with those in the input design. However, if the device is highly utilized and new logic has been added to the design, it may be desirable to favor the placement of that new logic over the guiding of these through-routes.

To allow PPR to use the through-route resources on any signal for the placement of new logic, start the program as follows.

```
ppr new guide=original guide_thru_routes=none
```

Copying Guide File Without Finishing Routing

To direct PPR to copy the placement and routing in the guide file without finishing the routing on unguided signals, invoke the program as shown in the following statement.

```
ppr new guide=original guide_only=true
```

This option can generate an LCA file in which a new design iteration has been guided against a previous iteration but in which no additional work has been done. You can then manually edit this LCA file in XDE and use it as a guide file for another run of PPR.

Using XACT-Performance Specifications

Using the XACT-Performance feature of PPR, you can specify path delay requirements on the schematic to guide the mapping, placement, and routing process. The "XACT-Performance Utility" chapter of this reference guide explains how to specify these requirements. Basic control over the XACT-Performance process is also available through the PPR command line options, as shown following.

Specifying Default Path Delays

For maximum flexibility, specify timing requirements in the schematic, as described in the "XACT-Performance Utility" chapter. However, for simple one-clock designs, you can specify basic timing requirements on the PPR command line.

As an example, consider the following PPR command.

```
ppr top dc2s=40 dc2p=auto dp2s=auto
dp2p=ignore
```

PPR interprets these options as follows.

- dc2s=40: PPR attempts to keep all clock-to-setup paths within 40 ns. Unlike requirements specified in the schematic, numbers on the command line must always be specified in nanoseconds.
- dc2p=auto: PPR chooses a reasonable path delay target for clock-to-pad paths and attempts to meet this target.
- dp2p=ignore: PPR ignores the path delay on pad-to-pad paths, which are strictly combinatorial. If the delays on such paths are

insignificant, the Ignored option allows PPR to concentrate on the other paths that are speed-critical.

You can use any of these option values — a number of nanoseconds, Auto, or Ignored — for any of the four options Dc2s, Dc2p, Dp2s, and Dp2p.

If any of these parameters is not specified, PPR does not supply a default value. However, for each flip-flop-related path type — flip-flops to flip-flops, flip-flops to pads, pads to flip-flops — for which you have not specified a parameter, PPR uses the Auto option to generate a specification that covers all paths of that type.

Controlling Delay When C2S Specifications Differ

By default, the path between two flip-flops that have different C2S specifications is not controlled by XACT-Performance; that is, no target delay is assigned to that path. You can assign such delay to these paths by setting the following option to True.

```
use_faster_c2s=true
```

The path between two such flip-flops is assigned a target delay equal to the faster of the two C2S specifications.

Controlling PPR if Specifications Cannot Be Met

By default, if PPR finds that it cannot meet a specified timing requirement, it relaxes the specification by a factor of 1.5 and continues with the place and route process.

To have PPR stop the placement and routing process if it finds that a specified timing requirement cannot be achieved, invoke it as follows.

```
ppr top stop_on_miss=true
```

Ignoring Path Delays in Place and Route

To place and route a design without taking path delays into consideration, use the following syntax when invoking PPR.

```
ppr top path_timing=false
```

This option causes any specified timing requirements to be ignored and also prevents PPR from choosing its own path delay targets.

Ignoring Specified Timing Requirements

To ignore the timing requirements specified in the design, invoke the program as follows.

```
ppr top ignore_timespec=all
```

Controlling Delays on Incomplete Paths

In a partial design, there may be paths that are not yet complete. You cannot control these paths with XACT-Performance specifications. Instead, use the following option to provide a maximum delay target for PPR to use when routing signals along such paths.

```
dflt_sig_dly=value
```

Value can be a number between 5 and 80, inclusive.

The delay represented by Dflt_sig_dly is not a path delay but only the routing delay of an individual signal. Dflt_sig_dly does not affect the partitioning or placement of an incomplete path.

Options

This section lists and explains the options that are available in PPR. They are listed in alphabetical order. The syntax of these options is *option=value*.

Complete

The Complete option tells PPR whether or not any additional logic is still to be added to the design.

Command line syntax:	`complete={true|false}`
Xactinit.dat file syntax:	None
XDM command:	`PlaceRoute → PPR →-nocomplete`
Values:	`true`, `false`
Default value:	`true`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Complete option is set to False, PPR preserves any sourceless or loadless signals in the design. It also avoids routing through

resources that may be needed for future logic, such as unused global buffers and completely empty CLBs. The default for this option is True.

Cstfile

The Cstfile option specifies the name of the constraints file to use. When this option is used, PPR ignores the *design*.cst file.

Command line syntax:	`cstfile=`*filename*
Xactinit.dat file syntax:	None
XDM command:	`PlaceRoute → PPR → cstfile=`*file*
Values:	*filename*, which can have any extension
Default value:	*filename*`.cst`
Applicable family:	XC3000A/L, XC3100A, XC4000

Dc2p

The Dc2p option specifies the default clock-to-pad time.

Command line syntax:	`dc2p=`*value*
Xactinit.dat file syntax:	`/ppr/dc2p=`*value*
XDM command:	`PlaceRoute → PPR → dc2p=`{*ns*`|AUTO|IGNORE}`
Values:	*floating-point_number*, `auto`, `ignore`
Default value:	None
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Dc2p option is used, PPR ensures that the delay along all clock-to-pad paths in the design that are not otherwise specified is less than the specified value, in nanoseconds. The value specified by this option overrides the Dc2p value specified in your schematic. *Value* can be one of the following settings.

- *floating-point_number* is a floating-point number in nanoseconds in the range of 0.1 to 3000.0.
- Auto selects a moderately aggressive target delay to apply to all clock-to-pad paths.

- Ignore ignores the timing of all clock-to-pad paths, which allows PPR to concentrate on other paths.

You can also specify XACT-Performance requirements on the schematic with much greater granularity than that available from the PPR command line. The "XACT-Performance Utility" chapter in this reference guide explains how to specify these requirements.

Dc2s

The Dc2s option specifies the default clock-to-setup time.

Command line syntax:	`dc2s=`*value*
Xactinit.dat file syntax:	`/ppr/dc2s=`*value*
XDM command:	`PlaceRoute` → `PPR` → `dc2s=` {*ns*|`AUTO`|`IGNORE`}
Values:	*floating-point_number*, `auto`, `ignore`
Default value:	None
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Dc2s option is used, PPR ensures that the delay along all clock-to-setup paths in the design that are not otherwise specified is less than the specified value, in nanoseconds. The value specified by this option overrides the Dc2s value specified in your schematic. *Value* can be one of the following settings.

- *floating-point_number* is a floating-point number in nanoseconds in the range of 0.1 to 3000.0.
- Auto selects a moderately aggressive target delay to apply to all clock-to-setup paths.
- Ignore ignores the timing of all clock-to-setup paths, which allows PPR to concentrate on other paths.

You can also specify XACT-Performance requirements on the schematic with much greater granularity than that available from the PPR command line. The "XACT-Performance Utility" chapter in this reference guide explains how to specify these requirements.

Dflt_sig_dly

In a partial design, there may be paths that are not yet complete. Such paths cannot be controlled by XACT-Performance specifications. The Dflt_sig_dly option provides a maximum delay target for PPR to use when routing signals along such paths.

Command line syntax:	`dflt_sig_dly=`*value*
Xactinit.dat file syntax:	`/ppr/dflt_sig_dly=`*value*
XDM command:	`PlaceRoute` → `PPR` → `dflt_sig_dly=`*num*
Values:	Number of nanoseconds, between 5 and 80, inclusive
Default value:	80 nanoseconds
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If XACT-Performance cannot determine a target routing delay for a given signal because the path that includes that signal is not complete, the value of the Dflt_sig_dly option is used as a target delay. The delay represented by Dflt_sig_dly is not a path delay but only the routing delay of an individual signal. Dflt_sig_dly does not affect the partitioning or placement of an incomplete path.

Dp2p

The Dp2p option specifies default pad-to-pad time.

Command line syntax:	`dp2p=`*value*
Xactinit.dat file syntax:	`/ppr/dp2p=`*value*
XDM command:	`PlaceRoute` → `PPR` → `dp2p=` {*ns*|`AUTO`|`IGNORE`}
Values:	*floating-point_number*, `auto`, `ignore`
Default value:	None
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Dp2p option is used, PPR ensures that the delay along all pad-to-pad paths in the design that are not otherwise specified is less than the specified value, in nanoseconds. The value specified by this

option overrides the Dp2p value specified in your schematic. *Value* can be one of the following settings.

- *floating-point_number* is a floating-point number in nanoseconds in the range of 0.1 to 3000.0.
- Auto selects a moderately aggressive target delay to apply to all pad-to-pad paths.
- Ignore ignores the timing of all pad-to-pad paths, which allows PPR to concentrate on other paths.

You can also specify XACT-Performance requirements on the schematic with much greater granularity than that available from the PPR command line. The "XACT-Performance Utility" chapter in this reference guide explains how to specify these requirements.

Dp2s

The Dp2s option specifies the default pad-to-setup time.

Command line syntax:	`dp2s=`*value*
Xactinit.dat file syntax	`/ppr/dp2s=`*value*
XDM command:	`PlaceRoute → PPR → dp2s=` {*ns*`|AUTO|IGNORE}`
Values:	*floating-point_number*, `auto`, `ignore`
Default value:	None
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Dp2s option is used, PPR ensures that the delay along all pad-to-setup paths in the design that are not otherwise specified is less than the specified value, in nanoseconds. The value specified by this option overrides the Dp2s value specified in your schematic. *Value* can be one of the following settings.

- *floating-point_number* is a floating-point number in nanoseconds in the range of 0.1 to 3000.0.
- Auto selects a moderately aggressive target delay to apply to all pad-to-setup paths.
- Ignore ignores the timing of all pad-to-setup paths, which allows PPR to concentrate on other paths.

You can also specify XACT-Performance requirements on the schematic with much greater granularity than that available from the PPR command line. The "XACT-Performance Utility" chapter in this reference guide explains how to specify these requirements.

Estimate

The Estimate option generates a report file containing device utilization statistics.

Command line syntax:	`estimate={true|false}`
Xactinit.dat file syntax:	None
XDM command:	`PlaceRoute → PPR → -estimate`
Values:	`true`, `false`
Default value:	`false`
Applicable family:	XC4000, XC5200

If the Estimate option is set to True, PPR maps the design without placing or routing it and creates a report (RPT) file; no LCA file is created. For partially completed designs, the Estimate=True option can be used to estimate resource utilization. False, the default, indicates that complete PPR processing will be performed.

Guide

The Guide option uses a specified LCA file to guide the design implementation. PPR follows the guide file's mapping, placement, and routing where possible.

Command line syntax:	`guide=`*filename*
Xactinit.dat file syntax:	None
XDM command:	`PlaceRoute → PPR → guide=`*file*
Values:	Any LCA file name
Default value:	None
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

Guide_blks

The Guide_blks option determines the type of blocks to be guided.

Command line syntax:	`guide_blks=`*value*
Xactinit.dat file syntax:	`/ppr/guide_blks=`*value*
XDM command:	`PlaceRoute` → `PPR` → `guide_blks=`*param*
Values:	`all`, `routed_only`
Default value:	`all`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

This option can be set to the following values:

- All guides all blocks in the guide file that are matched in the input design.
- Routed_only guides only those blocks that have some routing connected to them in the guide file and that are matched in the input design.

Guide_only

The Guide_only option guides placement and routing and completes all placement but does not complete routing that is not guided.

Command line syntax:	`guide_only={true|false}`
Xactinit.dat file syntax:	None
XDM command:	`PlaceRoute` → `PPR` → `-guide_only`
Values:	`true`, `false`
Default value:	`false`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

When this option is set to True, PPR guides and completes the placement but does not complete unguided routing. When it is set to False, it completely guides and routes the design.

Guide_routing

The Guide_routing option determines which routing to copy from the guide file.

Command line syntax:	`guide_routing=`*value*
Xactinit.dat file syntax:	`/ppr/guide_routing=`*value*
XDM command:	`PlaceRoute → PPR → guide_routing=`*param*
Values:	`whole_sigs, all`
Default value:	`all`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

You can set this option to the following values.

- Whole_sigs guides the routing only for signals that completely match all pins in the guide file.
- All guides the routing for all matching signals.

Guide_thru_routes

The Guide_thru_routes option controls which through-routes are preserved during guided routing. A through-route is a signal that PPR has routed through a CLB or other block. PPR uses through-routes to improve the routability and timing of a signal.

Command line syntax:	`guide_thru_routes=`*value*
Xactinit.dat file syntax:	`/ppr/guide_thru_routes=`*value*
XDM command:	`PlaceRoute → PPR → guide_thru_routes=`*param*
Values:	`all`, `whole_sigs`, `none`
Default value:	`whole_sigs`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

This option can be set to the following values.

- All locks the through-routes for all guided signals, preventing PPR from using those block resources to implement new user logic.

- Whole_sigs locks through-routes only on signals with completely matched pins. PPR can use the block resources used for through-routes on incompletely matched signals to implement new user logic.
- None allows any through-routes to be discarded, so PPR can use these block resources to implement new user logic.

–Helpall

The -Helpall option displays a list of the available PPR options.

Command line syntax:	`-helpall`
Xactinit.dat file syntax:	None
XDM command:	`PlaceRoute → PPR → -helpall`
Values:	None
Default value:	None
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

Ignore_maps

The Ignore_maps option ignores all FMAP and HMAP symbols in an XC4000 design.

Command line syntax:	`ignore_maps={true|false}`
Xactinit.dat file syntax:	`/ppr/ignore_maps={true|false}`
XDM command:	`PlaceRoute → PPR → -ignore_maps`
Values:	`true,false`
Default value:	`false`
Applicable family:	XC4000, XC5200

When this option is set to True, PPR ignores FMAP and HMAP symbols in the design; when it is set to False, it does not.

Ignore_rlocs

The Ignore_rlocs option ignores all RLOC parameters in the design and in the constraints filfile.

Command line syntax:	`ignore_rlocs={true|false}`
Xactinit.dat file syntax:	`/ppr/ignore_rlocs={true|false}`
XDM command:	`PlaceRoute → PPR → -ignore_rlocs`
Values:	`true`, `false`
Default value:	`false`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

When this option is set to True, PPR ignores RLOC parameters; when it is set to False, it does not.

Note: XC4000 carry logic (CY4) symbols must have either RLOC or LOC constraints. If a design contains CY4 symbols with RLOC constraints, the Ignore_rlocs option will not ignore these RLOC constraints.

Ignore_timespec

The Ignore_timespec option ignores timing requirements specified for a design.

Command line syntax:	`ignore_timespec=`*value*
Xactinit.dat file syntax:	`/ppr/ignore_timespec=`*value*
XDM command:	`PlaceRoute → PPR → ignore_timespec=`*param*
Values:	`none`, `all`
Default value:	`none`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Ignore_timespec option is set to All, PPR ignores any timing requirements. If it is set to None, the specified timing requirements are used in placing and routing.

Note: To selectively ignore timing requirements from only the design file or the CST file, use the Ignore_timespec option of the XNFPrep program. The PPR Ignore_timespec=All option ignores timing requirements from both files.

Ignore_xnf_locs

The Ignore_xnf_locs option ignores the LOC parameters in the input design file for specific types of logic.

Command line syntax:	`ignore_xnf_locs=`*type*
Xactinit.dat file syntax:	`/ppr/ignore_xnf_locs=`*type*
XDM command:	`PlaceRoute` → `PPR` → `ignore_xnf_locs=`*type*
Values:	`all`, `io`, `interior`, `none`
Default value:	`none`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

Type specifies the logic on which to ignore LOC parameters.

- All ignores LOC parameters on all logic.
- Io ignores LOC parameters on I/O symbols.
- Interior ignores LOC parameters on internal logic, that is, on everything except I/O symbols.
- None does not ignore any LOC parameters.

Note: XC4000 carry logic (CY4) symbols must have either RLOC or LOC constraints. If a design contains CY4 symbols with LOC constraints, the Ignore_xnf_locs option will not ignore these LOC constraints.

Lock_routing

The Lock_routing option determines which routing PPR can change from the guide file.

Command line syntax:	`lock_routing=`*value*
Xactinit.dat file syntax:	`/ppr/lock_routing=`*value*

XDM command:	`PlaceRoute` → `PPR` → `lock_routing=`*param*
Values:	`all`, `whole_sigs`, `none`
Default value:	`whole_sigs`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

This option can be set to the following values:

- All locks the guided routing for all guided signals, preventing PPR from rerouting those signals.
- Whole_sigs locks guided routing only on signals with completely matched pins.
- None allows guided routing to be discarded and re-routed to improve the timing.

Logfile

The Logfile option specifies an alternate name for the ppr.log file.

Command line syntax:	`logfile=`*filename*
Xactinit.dat file syntax:	None
XDM command:	None
Values:	*filename*[.log]
Default value:	ppr.log
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

Use the Logfile option to assign a different name to the ppr.log file. The .log extension is appended if you do not specify an extension with the new file name. If the Logfile option is not used, the screen output is written to the ppr.log file, overwriting any previous versions of this log file.

Open_guide_blocks

The Open_guide_blocks option places new logic into guided blocks.

Command line syntax:	open_guide_blocks=`{true|false}`

Xactinit.dat file syntax: `/ppr/open_guide_blocks={true|false}`

XDM command: None

Values: `true`, `false`

Default value: false

Applicable family: XC3000A/L, XC3100A, XC4000, XC5200

If you set the value to true, new logic can be placed into guided blocks even if lock_routing={whole_sigs, all}. If you set the value to false, logic may be added to guided blocks only if lock_routing=none.

Outfile

The Outfile option specifies an alternate output file name for the LCA and RPT files.

Command line syntax: `outfile=`*filename*

Xactinit.dat file syntax: None

XDM command: `PlaceRoute` → `PPR` → `outfile=`*name*

Values: *filename*

Default value: *designname* (input file name)

Applicable family: XC3000A/L, XC3100A, XC4000, XC5200

Use the Outfile option to name the RPT and LCA file with a name different from the input design name. The .lca and . rpt extensions are appended to these file names.

Paramfile

Using the Paramfile option, you can specify PPR options in a separate file, called a parameter file, instead of from the operating system prompt. A parameter file is a text file containing a list of desired options and their respective values, as in the following example.

```
parttype=4005pg156
estimate=true
ignore_xnf_locs=io
```

Command line syntax: `paramfile=`*filename*

Xactinit.dat file syntax:	None
XDM command:	`PlaceRoute` → `PPR` → `paramfile=`*file*
Values:	*filename*
Default value:	None
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

Use the name of the parameter file as the value for the Paramfile option.

You can specify additional options on the command line whenever a parameter file is used; these override similar options specified in the parameter file.

Parttype

The Parttype option specifies the target LCA device, the package, and the speed. It overrides the part type, if any, specified in the input file.

Note: The XNFPrep program performs numerous checks on the design based on the LCA part type. For this reason, it is recommended that the part type not be changed in PPR directly. To quickly target a design to a new part, rerun XNFPrep using that program's Parttype option and proceed with the normal design flow from that point.

Command line syntax:	`parttype=`*parttype*
Xactinit.dat file syntax:	None
XDM command:	None; the XDM Part command specifies the part type.
Values:	Valid LCA part
Default value:	Taken from XTF or MAP file
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

Use the Parttype option to specify a target LCA device, package, and speed. The part type specified with this option overrides any part type previously specified in the design flow. Examples of valid part types are as follows.

```
4010PG191-5
4003APC84-6
```

```
3042APG132-6
5210M208-5
```

The Parttype option is not available from XDM, since XDM passes the correct part to PPR automatically.

Path_timing

The Path_timing option controls whether path delays are considered in placement and routing.

Command line syntax:	`path_timing={true|false}`
Xactinit.dat file syntax:	`/ppr/path_timing={true|false}`
XDM command:	`PlaceRoute` → `PPR` → `-nopath_timing`
Values:	`true`, `false`
Default value:	`true`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Path_timing option is set to True, the mapping, placement, and routing phases are controlled by the XACT-Performance timing requirements specified in the design file, or if none are specified, by path delay targets chosen by PPR. If Path_timing is set to False, PPR is controlled by routability requirements only.

Placer_effort

The Placer_effort option controls the compromise between PPR execution time and placement quality. A lower value directs PPR to produce a quick result that may be of lower quality, while a higher value directs PPR to take more time to produce a better result.

Command line syntax:	`placer_effort=`*value*
Xactinit.dat file syntax:	`/ppr/placer_effort=`*value*
XDM command:	`PlaceRoute` → `PPR` → `placer_effort=`*num*
Values:	Integer between 1 and 5, inclusive
Default value:	2

Applicable family: XC3000A/L, XC3100A, XC4000, XC5200

The settings of Placer_effort are interpreted as follows.

- 1 produces a quick placement, which may or may not be routable within the specified timing requirements.
- 2, 3, and 4 are relative degrees of effort that PPR should use in placing the design; higher numbers indicate more effort.
- 5 keeps working on the placement until no more improvement can be made.

Report_pagelength

The Report_pagelength option specifies the number of lines on a page.

Command line syntax: `None`

Xactinit.dat file syntax: `report_pagelength=`*value*

XDM command: None

Values: `positive integer`

Default value: `66`

Applicable family: XC3000A/L, XC3100A, XC4000, XC5200

The report_pagelength option is a global option, so if you specify this option in PPR, it will effect report files generated by other programs.

Report_leftmargin

The Report_leftmargin specifies the width of the left page margin in characters.

Command line syntax: `None`

Xactinit.dat file syntax: `report_leftmargin=`*value*

XDM command: None

Values: `positive integer`

Default value: `4`

Applicable family: XC3000A/L, XC3100A, XC4000, XC5200

The report_leftmargin option is a global option, so if you specify this option in PPR, it will effect report files generated by other programs.

Report_textwidth

The Report_textwidth option specifies the number of characters across the width of the page. This may also be referred to as line length.

Command line syntax:	`None`
Xactinit.dat file syntax:	`report_textwidth=`*value*
XDM command:	None
Values:	`positive integer`
Default value:	`70`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

The report_textwidth option is a global option, so if you specify this option in PPR, it will effect report files generated by other programs.

Route

The Route option controls whether or not the design is routed.

Command line syntax:	`route={true|false}`
Xactinit.dat file syntax:	None
XDM command:	`PlaceRoute` → `PPR` → `-noroute`
Values:	`true`, `false`
Default value:	`true`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

Set this option to False only if you want an unrouted LCA file. To use guided routing, select the Guide_only option instead.

Route_thru_blks

In order to improve routability and timing, PPR may use through-routes, which route some signals through a CLB or other block. Although this technique usually improves the routing, the Route_thru_blks option allows it to be limited or disabled entirely.

Command line syntax:	`route_thru_blks=`*value*
Xactinit.dat file syntax:	`/ppr/route_thru_blks=`*value*
XDM command:	`PlaceRoute` → `PPR` → `route_thru_blks=`*param*
Values:	`ok`, `limit`, `never`. `Never` is not available with XC4000 and XC5200 designs.
Default value:	`ok` if `complete=true`; `limit` if `complete=false`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

The *value* can be one of the following settings.

- Ok allows PPR to route through any block in order to improve the routability and timing of the design.
- Limit prevents PPR from routing through any block that has not already been used for design logic. That is, if the flip-flops in a given CLB are already used, PPR is allowed to route through the unused function generator in that CLB. The Limit value is helpful for incremental design, because it prevents PPR from using CLBs outside of the current functional block's area.
- Never prevents PPR from ever routing through a block. Doing so may make the design more difficult to route and degrade the overall design timing. If the Never value is used in these cases, PPR is not able to route the design completely. This option cannot be used with XC4000 designs.

Route_thru_bufg

The Route_thru_bufg command allows PPR to route through unused global buffers — BUFGS in XC4000, BUFG in XC5200, or GCLK and ACLK in XC3000 — if doing so seems more efficient than using general interconnect.

Command line syntax:	`route_thru_bufg=`*value*
Xactinit.dat file syntax:	`/ppr/route_thru_bufg=`*value*
XDM command:	`PlaceRoute` → `PPR` → `route_thru_bufg=`*param*

Valid settings:	`ok`, `never`
Default Value:	`never` if `complete=false`; `ok` if `complete=true`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Route_thru_bufg option is set to Ok, PPR routes signals through the unused buffers noted above.

The default value of the Route_thru_bufg options is dependent on the value of the Complete option. If Complete is set to True, Route_thru_bufg defaults to Ok; if Complete is set to False, Route_thru_bufg defaults to Never. If the Complete=False option is used, Route_thru_bufg is set to Never automatically so that global buffers that may be needed for later design iterations are not used by PPR.

Router_effort

The Router_effort option controls the compromise between PPR execution time and routing quality. A lower value directs PPR to produce a quick result that may be of lower quality, and a higher value directs it to take more time to produce a better result.

Command line syntax:	`router_effort=`*value*
Xactinit.dat file syntax:	`/ppr/router_effort=`*value*
XDM command:	`PlaceRoute` → PPR → `router_effort=`*num*
Values:	Integers between 1 and 4 (inclusive), `not-in-use`
Default value:	2
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

The settings of Router_effort are interpreted as follows.

- 1 produces a quick route, which may or may not meet the specified timing requirements.
- 2 is the default.
- 3 works harder than the default to meet the specified timing requirements.

- 4 keeps working at routing until the design has been routed to meet the timing requirements, or until no more improvement can be made, given the current placement.
- Not-in-use allows detailed manual control of advanced router options from the xactinit.dat file.

Rpt_net_loc

The Rpt_net_loc option prints a summary of net locations in the RPT file.

Command line syntax:	`rpt_net_loc={true|false}`
Xactinit.dat file syntax:	`/ppr_rpt_net_loc={true|false}`
XDM command:	None
Values:	`true`, `false`
Default value:	`false`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Rpt_net_loc option is set to True, a list containing the final location of the sources for all nets in your design is included in the PPR report file, *design*.rpt. False indicates that the PPR report file will not contain a list of net locations.

Rpt_net_loc

The Rpt_net_loc option prints a summary of net locations in the RPT file.

Command line syntax:	`rpt_net_loc={true|false}`
Xactinit.dat file syntax:	`/ppr_rpt_net_loc={true|false}`
XDM command:	None
Values:	`true`, `false`
Default value:	`false`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Rpt_net_loc option is set to True, a list containing the final location of the sources for all nets in your design is included in the

PPR report file, *design*.rpt. False indicates that the PPR report file will not contain a list of net locations.

Rpt_sym_loc

The Rpt_sym_loc option prints a summary of symbol locations in the RPT file.

Command line syntax:	`rpt_sym_loc={true|false}`
Xactinit.dat file syntax:	`/ppr/rpt_sym_loc={true|false}`
XDM command:	None
Values:	`true`, `false`
Default value:	`false`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Rpt_sym_loc option is set to True, a list containing the placement of all symbols from your original design is included in the PPR report file, *design*.rpt. When this option is set to False, which is the default, this symbol placement list is not included in the PPR report file.

Save_files

The Save_files option saves PPR intermediate files after PPR completes.

Command line syntax:	`save_files={true|false}`
Xactinit.dat file syntax:	`/ppr/save_files=`*value*
XDM command:	Not available from XDM
Values:	`true`, `false`
Default value:	`false`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

The Save_files option determines whether or not PPR deletes its intermediate files after execution. If this option is set to True, PPR preserves these temporary files.

Seed

A seed is a random number that determines the order of the cells in the design to be placed. The Seed option specifies the seed used during placement improvement.

Command line syntax:	`seed=`*value*
Xactinit.dat file syntax:	None
XDM command:	`PlaceRoute` → `PPR` → `seed=`*num*
Values:	Positive integer
Default value:	Random seed generated by PPR
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

Normally PPR should be allowed to generate its own random seed, which is the default condition.

Stop_on_miss

The Stop_on_miss option stops PPR if the XACT-Performance timing requirements cannot be met.

Command line syntax:	`stop_on_miss={true|false}`
Xactinit.dat file syntax:	`/ppr/stop_on_miss={true|false}`
XDM command:	`PlaceRoute` → `PPR` → `-stop_on_miss`
Values:	`true`, `false`
Default value:	`false`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

If the Stop_on_miss option is set to True, PPR stops if it cannot meet an XACT-Performance requirement that you have specified. It also includes in the log file information about path endpoints whose XACT-Performance requirements cannot be met. If this option is set to False, PPR continues to process the design, producing the best results possible.

Timing

The Timing option controls how PPR uses delay information in routing.

Command line syntax:	`timing=`*value*
Xactinit.dat file syntax:	`/ppr/timing=`*value*
XDM command:	`PlaceRoute` → `PPR` → `timing=`*param*
Values:	`when_routable`, `forced_on`, `ignored`
Default value:	`forced_on`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

Use the Timing option to specify how PPR uses timing information when routing the design. Three settings are available for this option.

- When_routable completes up to two routing passes. Routing during the first pass is based on minimizing the resources used. If this pass is successfully completed, PPR initiates a second pass, which is based on timing information.
- Forced_on completes one routing pass, which is based on timing information.
- Ignored routes the design once, ignoring timing information. This setting can reduce PPR run time but usually yields a lower-performance design. This option turns off XACT-Performance completely.

Xilinx recommends that you use delay information, that is, that the timing option be set to When_routable or Forced_on.

Use_faster_c2s

The Use_faster_c2s option controls whether or not PPR applies a path delay target to a path between two flip-flops that have different C2S specifications.

Command line syntax:	`use_faster_c2s={true|false}`
Xactinit.dat file syntax:	`/ppr/use_faster_c2s={true|false}`

XDM command:	`PlaceRoute → PPR → -use_faster_c2s`
Values:	`true`, `false`
Default Value:	`false`
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

By default, the path between two flip-flops that have different C2S specifications is not controlled by XACT-Performance; that is, no target delay is assigned to that path. If the Use_faster_c2s option is set to True, the path between two such flip-flops is assigned a target delay equivalent to the faster of the two C2S specifications.

User_search_path

The User_search_path option specifies a path for searching for data files.

Command line syntax:	None
Xactinit.dat file syntax:	`user_search_path=`*value*
XDM command:	None
Values:	*path*
Default value:	None
Applicable family:	XC3000A/L, XC3100A, XC4000, XC5200

The User_search_path option specifies additional directories where PPR should search for data files. The *value* can contain one or more fully qualified directory paths, each with a trailing slash. These directories are searched for data files — for example, XTF files — that are not found in the current directory.

Options Summary

Table 6-5 summarizes the options available in PPR. They are listed in alphabetical order within each group.

Table 6-5 PPR Options

Option	Default	Description	Families
General Options			
`complete= {true|false}`	`true`	Tells PPR if design needs to add or remove logic	XC3000A/L, XC3100A, XC4000, XC5200
`estimate= {true|false}`	`false`	Report estimated resource use	XC4000, XC5200
`-helpall`	No default	Show all help information	XC3000A/L, XC3100A, XC4000, XC5200
`logfile=`*filename*	ppr.log	Rename log file	XC3000A/L, XC3100A, XC4000, XC5200
`outfile=`*filename*	Input file name	Specify output file name	XC3000A/L, XC3100A, XC4000, XC5200
`paramfile=`*filename*	No default	Specify parameter file name	XC3000A/L, XC3100A, XC4000, XC5200
`parttype=`*type*	Taken from XTF or MAP file	Set device, package, speed	XC3000A/L, XC3100A, XC4000, XC5200
`save_files= {true|false}`	`false`	Save intermediate files	XC3000A/L, XC3100A, XC4000, XC5200

Constraint Options			
`cstfile=`*filename*	*filename*.cst file	Specify constraints file to use and ignore *filename*.cst file	XC3000A/L, XC3100A, XC4000, XC5200
`ignore_maps= {true|false}`	`false`	Ignore F5MAP, FMAP, and HMAP symbols	XC4000, XC5200
`ignore_rlocs= {true|false}`	`false`	Ignore RLOC parameters in design file	XC3000A/L, XC3100A, XC4000, XC5200
`ignore_xnf_locs= {all|io|interior none}`	`none`	Ignore LOC parameters in design file for specified type	XC3000A/L, XC3100A, XC4000, XC5200
Placement and Routing Options			
`dflt_sig_dly=`*value*, where *value* = 5 – 80 ns	80 ns	Specify signal delay target for incomplete paths not controlled by XACT-Performance	XC3000A/L, XC3100A, XC4000, XC5200
`placer_effort=`*num*, where *num* = 1 – 5 1=least effort 5=most effort	2	Set level of placer effort. Balance of speed vs. quality.	XC3000A/L, XC3100A, XC4000, XC5200
`route={true|false}`	`true`	Continue after placement through routing	XC3000A/L, XC3100A, XC4000, XC5200
`route_thru_blks= {ok|limit|never}` `ok`: any available block `limit`: partially used blocks only `never`: no blocks	`ok` if `complete= true; limit` if `complete= false`	Allow routing through blocks	XC3000A/L, XC3100A, XC4000, XC5200 (`never` option is not available with XC4000 and XC5200)

route_thru_bufg= {ok|never}	never if complete =false; ok if complete=true	Allow routing through unused BUFG, BUFGS, GCLK, or ACLK buffers	XC3000A/L, XC3100A, XC4000, XC5200
router_effort=*num*, where *num* = 1 – 4 1=least effort 4=most effort	2	Set level of router effort. Balance of speed vs. quality.	XC3000A/L, XC3100A, XC4000
seed=*num*, where *num* = positive integer	Random seed generated by PPR	Set random seed used during placement improvement	XC3000A/L, XC3100A, XC4000, XC5200
timing={|forced_on| when_routable| ignored}	when_routable	Allow use of delay information to guide router	XC3000A/L, XC3100A, XC4000, XC5200
XACT-Performance Options			
dc2p={*number*|auto| ignore}	No default	Set default clock-to-pad time	XC3000A/L, XC3100A, XC4000, XC5200
dc2s={*number*|auto| ignore}	No default	Set default clock-to-setup time	XC3000A/L, XC3100A, XC4000, XC5200
dp2p={*number*|auto| ignore}	No default	Set default pad-to-pad time	XC3000A/L, XC3100A, XC4000, XC5200
dp2s={*number*|auto| ignore}	No default	Set default pad-to-setup time	XC3000A/L, XC3100A, XC4000, XC5200
ignore_timespec= {all|none)	none	Ignore design file timing specifications	XC3000A/L, XC3100A, XC4000, XC5200
path_timing= {true|false}	true	Use path analysis to guide implementation	XC3000A/L, XC3100A, XC4000, XC5200

`stop_on_miss= {true|false}`	`false`	Stop if a timing specification is not met	XC3000A/L, XC3100A, XC4000, XC5200
`use_faster_c2s= {true|false}`	`false`	Use faster of two C2S specifications for paths between independently specified flip-flops	XC3000A/L, XC3100A, XC4000, XC5200
Report Options			
`rpt_net_loc= {true|false}`	`false`	Print summary of net locations in the RPT file	XC3000A/L, XC3100A, XC4000, XC5200
`rpt_sym_loc= {true|false}`	`false`	Print a summary of symbol locations in the RPT file	XC3000A/L, XC3100A, XC4000, XC5200
Guide Options			
`guide=`*filename*	File with .lca extension	Use specified LCA file to guide design implementation	XC3000A/L, XC3100A, XC4000, XC5200
`guide_blks={all| routed_only}` `all:` any blocks with matching signals `routed_only:` blocks with matching signals and some connected routing	`all`	Set criteria for guiding blocks	XC3000A/L, XC3100A, XC4000, XC5200
`guide_only= {true|false}`	`false`	Guide placement and routing without finishing routing	XC3000A/L, XC3100A, XC4000, XC5200

guide_routing= {all|whole_sigs} all: for any matching pins whole_sigs: only for signals that completely match all pins	all	Restore routing for signals	XC3000A/L, XC3100A, XC4000, XC5200
guide_thru_routes= {all|whole_sigs|none}	whole_sigs	Control which through-routes are preserved during guided routing	XC3000A/L, XC3100A, XC4000, XC5200
lock_routing={all|whole_sigs|none} all: lock routing for all guided signals whole_sigs: lock only signals with completely matched pins none: allow rip-up improvement on all guided routing	whole_sigs	Lock routing from guide file	XC3000A/L, XC3100A, XC4000, XC5200

Constraints File Syntax

Attributes, Constraints, and Carry Logic

For information, refer to the chapter with this title in the *Libraries Guide*.

The MakeBits Program

This program is compatible with the following families.

- XC2000
- XC2000L
- XC3000
- XC3100
- XC3000A
- XC3000L
- XC3100A
- XC4000
- XC4000A
- XC4000H
- XC5200

The MakeBits program creates configuration bitstreams for your FPGA designs. The configuration bitstream describes the internal logic functions and interconnections of an FPGA device. MakeBits provides commands to generate configuration bitstreams, as well as change startup options and run the XACT Design Rules Checker (DRC).

There are two functionally equivalent versions of MakeBits. There is a stand-alone version, used by XMake, that you use from the operating system shell. Another version is available from XDM and the XACT Design Editor (XDE). The two versions differ only in their user interface; both produce an identical BIT file.

You can also use MakeBits to generate other types of files and reports, in addition to the configuration bitstream, that contain information about the design and the design rule violations.

The first part of this chapter describes the MakeBits syntax, and the input and output files for the program. Next, is a description of the options available with the stand-alone version, followed by a description of the MakeBits version run within XDE.

Syntax

MakeBits accepts multiple FPGA design files as input and creates an individual bitstream for each input file.

The following command line syntax creates a binary bitstream file from your FPGA design file.

`makebits` [*options*] *design*`.lca ....`

where:

- *options* are one or more valid command line options
- *design*.lca is one or more placed and routed FPGA designs

Files

The input files that MakeBits requires and the output files that MakeBits generates are described below. Regardless of the MakeBits version you use (stand-alone, XDM, or part of XDE), the input and output files are the same.

Input Files

MakeBits requires an LCA file for input.

design.lca

This input file contains a design partitioned into the FPGA architecture. The design must be completely routed and free of the XACT system fatal DRC errors before you run MakeBits.

Output Files

The first three output files are created when you use MakeBits. The last three files, *design*.rbt, *design*.msk and *design*.lca are created when you use the -b option, the -m option, and the -n and -t options, respectively.

design.bit

This binary file contains the configuration data for an FPGA design.

design.ll

This file is created when you use the MakeLL option. It is an ASCII file that lists the positions of flip-flop outputs, CLB, and RAM memory cells in the bitstream. The location of each output is referenced by frame and frame offset within the bitstream. It is used by the XChecker program as part of the readback.

design.mbo

This file is created automatically when MakeBits generates the bitstream file. The MBO file contains CONFIGURE commands that record the state of all configuration options at the time the bitstream file was created.

design.rbt

This file is created when you use the -b option. It is an ASCII version of the configuration bitstream file.

design.msk

This file is created by the -m option. This MSK file is needed to read back the configuration data of an operating FPGA device.

_design.lca

This file is created by the -n option when used with the tie (-t) option. It saves the tied FPGA design file as *_design*.lca, so the tied design can also be used for simulation.

Options (Stand-Alone Version)

The options that you can use with the stand-alone MakeBits program are described below in alphabetical order. The options and settings for the -f option are also listed in alphabetical order.

–b Generate ASCII Configuration Data

This option creates a rawbits (RBT) file and a binary BIT file. The RBT file is a text file that contains ASCII 1s and 0s. These characters represent the actual bits that are contained in the configuration bitstream and are downloaded to the FPGA design.

Designers using a microprocessor to configure a single FPGA design can process the RBT file to create a source file that can be included in the microprocessor source code. The sequence of characters in the RBT file is the same as the bit sequence that must be written to an FPGA device. When the RBT file is included in the source code, some file reformatting is required. For example, eight ASCII characters, each defined by eight bits, must be converted into one byte of configuration data.

Note: Do not use the -b option with the -m option. If you want both a rawbits file and a mask file, run MakeBits twice — once with the -b option, and once with the -m option.

Note: If you specify both the -b and -m options, and only run the program once, MakeBits ignores the -m option.

–c Set CMOS Input Signal Threshold (XC2000 and XC3000 Only)

This option sets the FPGA design input-signal thresholds for XC2000 and XC3000 designs to CMOS level for interface capability. If this option is not used, the input thresholds default to TTL levels. The special-purpose clock inputs, Tclkin, Bclkin, and Pwrdwn (the PowerDown pin), always require CMOS-level signals, even if the FPGA design input thresholds are specified as TTL compatible.

–d Run the Design Rules Checker

This option runs DRC. You should run DRC in Editls or in MakeBits before creating a bitstream. DRC detects any design errors that could

cause the FPGA device to function improperly. If a fatal error occurs, you must correct the error before creating the bitstream.

MakeBits does not generate a bitstream when you use this option. After DRC is run and no fatal errors are detected, re-start MakeBits with the desired options; do not specify the -d option again.

Note: The -d option cannot be combined with any other command-line options.

–f Set Configuration (XC4000 and XC5200 Only)

This option specifies the startup timing and other bitstream options for the XC4000 and XC5200 devices. Timing sequences are predefined startup defaults that use the following syntax.

```
makebits -f timing_sequence
```

There are four valid startup sequences: Cclk_Nosync, Cclk_Sync, Uclk_Nosync, and Uclk_Sync. These startup sequences are described in the next section. Refer to Figure 7-1 to view the startup sequences for the XC4000 or XC5200 family. For more information about startup timing, refer to *The Programmable Logic Data Book.*

The default for the -f option is Cclk_Nosync. This startup sequence makes an XC4000 or XC5200 device compatible with an XC3000 device that is set for early Done and late Reset. Enter the following,

```
makebits -f cclk_nosync
```

The -f option has sub-options that represent settings you use to set the configuration for an XC4000 or XC5200 design. These options have the following syntax.

```
makebits -f option:setting
```

For example, to enable Cyclic Redundancy Checking (CRC), use the following syntax.

```
makebits -f crc:enable
```

The following sections describe the startup sequences for the -f option.

Startup Sequences (-f option)

This section describes the four startup sequences and their defaults; then it describes the options, their settings, and their defaults.

Note: When mixing devices, the one with the latest "finished point" should be the master. (The master stops clocking when it reaches the finished point.) See Figure 7-1 for more information.

Cclk_Nosync

Selecting this sequence causes the following defaults to take effect.

StartupClk:	`Cclk`
SyncToDone:	`No`
DoneActive:	`C1`
OutputsActive:	`C2`
GSRActive:	`C3`

This startup sequence makes an XC4000 or XC5200 device consistent with an XC3000 device set for early Done and late Reset.

Cclk_Sync

Selecting this sequence causes the following defaults to take effect.

StartupClk:	`Cclk`
SyncToDone:	`Yes`
DoneActive:	`C1`
OutputsActive:	`DI_PLUS_1`
GSRActive:	`DI_PLUS_1`

This startup sequence is the most consistent with the XC2000 and XC3000 devices, since it synchronizes the release of GSR and I/Os to the external DoneIn signal. This startup sequence makes an XC4000 or XC5200 device consistent with an XC3000 device set for early Done and late Reset.

Uclk_Nosync

Selecting this sequence causes the following defaults to take effect.

StartupClk:	`Userclk`
SyncToDone:	`No`
DoneActive:	`U2`

OutputsActive: `U3`
GSRActive: `U4`

This startup sequence makes XC4000 or XC5200 devices inconsistent with XC2000 or XC3000 devices if they are in the same daisy chain, since the release of Done is synchronized to an external User Clock. There is no synchronization of I/Os or GSR to DoneIn.

Uclk_Sync

Selecting this sequence causes the following defaults to take effect.

StartupClk: `Userclk`
SyncToDone: `Yes`
DoneActive: `U2`
OutputsActive: `DI_PLUS_1`
GSRInActive: `DI_PLUS_2`

This startup sequence makes XC4000 or XC5200 devices inconsistent with XC2000 or XC3000 devices if they are in the same daisy chain, since the release of Done is synchronized to an external User Clock. IOs and GSR are synchronous to the clocks following DoneIn.

Note: When using Vclk_sync or Vclk_nosync you must provide a user clock to finish the configuration sequence. Without a user clock the FPGA will not configure.

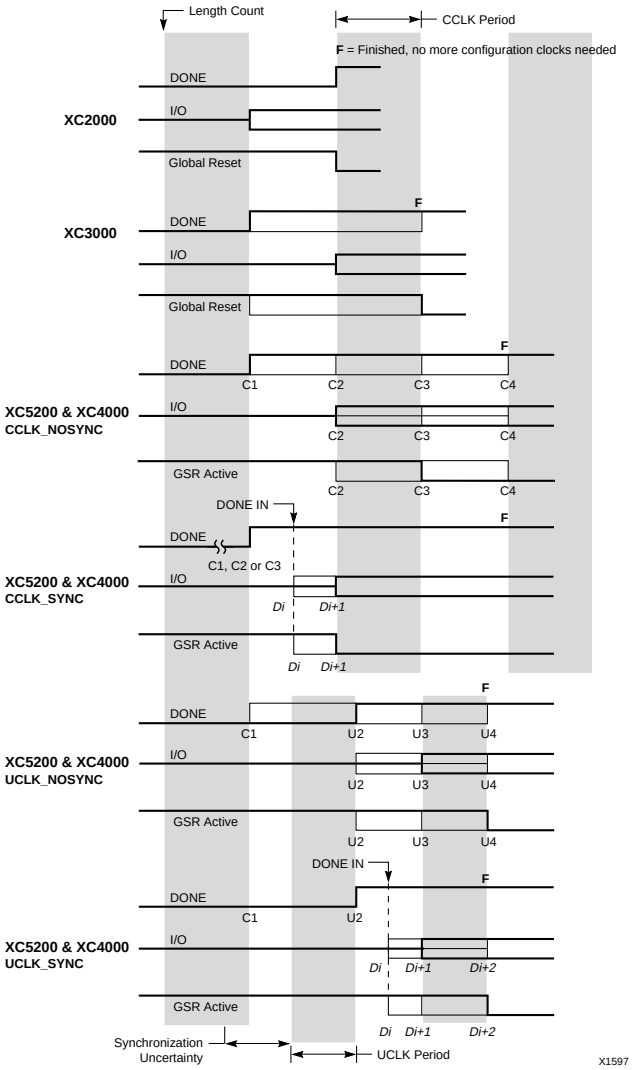


Figure 7-1 Startup Timing Sequences for XC2000, XC3000, XC4000, and XC5200 Devices

Startup Sequence Options

The options available with the four startup sequences are described below.

CRC

Enable or disable Cyclic Redundancy Checking (CRC) on a chip-by-chip basis during configuration.

Settings: `Enable, Disable`
Default: `Enable`

ConfigRate

Select the configuration clock rate. There are two choices: slow or fast. Slow is equivalent to 1 MHz, and fast is equivalent to 8 MHz (nominal).

Settings: `Slow, Fast`
Default: `Slow`

DonePin

Enable or disable an internal pull-up to the Done pin.

Settings: `Pullup, Nopullup`
Default: `Pullup`

TdoPin (XC4000 Only)

Add a pull-up or a pull-down to the TDO pin (Test Data Out for Boundary Scan).

Settings: `Pullup, Pulldown`
Default: `Neither`

M1Pin (XC4000 Only)

Add a pull-up or a pull-down to the M1 pin.
Settings: `Pullup, Pulldown`
Default: `Neither`

Note: The Pullup and Pulldown options for the TDO and M1 pins act as toggles and are mutually exclusive. The default is inactive.

Selecting one option enables it and disables the other. Selecting the same option a second time disables it.

BSReconfig (XC5200 Only)

Enable or disable reconfiguration via boundary scan.

Settings: `Enable, Disable`
Default: `Disable`

OscClk (XC5200 Only)

Drive oscillator with either user clock or internal oscillator.

Settings: `UserClk` (*user supplied*), `Cclk` (*internal*)
Default: `Cclk`

ReadCapture

Enable or disable readback of configuration bitstream.
Settings: `Enable, Disable`
Default: `Disable`

ReadAbort

Enable or disable aborting the readback sequence during the readback sequence.
Settings: `Enable, Disable`
Default: `Disable`

ReadClk

Set the readback clock to be Cclk or a user-supplied clock (from a net inside the FPGA that is connected to the 'i' pin of the rdclk block).
Settings: `Cclk` (*pin**), `Rdbk` (*user supplied*)
Default: `Cclk`

StartupClk

Select a user-supplied clock or the internal Cclk for controlling the post-configuration startup phase of the FPGA initialization.
Settings: `Cclk` (*pin**), `UserClk` (*user supplied*)
Default: `Cclk`

*In modes where Cclk is an output, this pin is driven by the internal oscillator.

SyncToDone

Synchronize the I/O startup sequence to the external DoneIn signal.
Settings: `Yes, No`
Default: `No`

DoneActive

Select the event that activates the FPGA Done signal. There are a maximum of four events that you can select from at one time. These events are Cclk edges or external (user) clock edges. The actual options available at any time depend on the selections made for StartupClk and SyncToDone.

Settings: `C1` – first-Cclk rising edge after the length count is met
`C2` – second-Cclk rising edge after the length count is met
`C3` – third-Cclk rising edge after the length count is met
`C4` – fourth-Cclk rising edge after the length count is met
`U2` – second-valid-user-clock rising edge after C1
`U3` – third-valid-user-clock rising edge after C1
`U4` – fourth-valid-user-clock rising edge after C1
Default: `C2`

Note: U2, U3, and U4 options are available only when UserClk is selected.

OutputsActive

Select the event that releases the I/O from 3-state condition and turns the configuration related pins operational. There are a maximum of four events that you can select from at one time. These events are Cclk edges, external (user) clock edges, and the external signal DoneIn. The actual options available at any time depend on the selections made for StartupClk and SyncToDone.

Settings: `C2` – second-Cclk rising edge after the length count is met
`C3` – third-Cclk rising edge after the length count is met
`C4` – fourth-Cclk rising edge after the length count is met
`U2` – second-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)

`U3` — third-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
`U4` — fourth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
`DI` — when the DoneIn signal goes High
`DI_PLUS_1` — first Cclk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High
`DI_PLUS_2` — second Cclk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High

Default: `C3`

Note: U2, U3, U4, DI, DI_PLUS_1, and DI_PLUS_2 options are available only when SyncToDone is selected (set to Yes).

GSRInactive

Select the event that releases the internal set-reset to the latches and flip-flops. There are nine events that you can select. These events are Cclk edges, external (user) clock edges and the external signal DoneIn. Only some of these events become options at one time depending on the combination of StartupClk and SyncToDone selected.

Settings: `C2` — second-Cclk rising edge after the length count is met
`C3` — third-Cclk rising edge after the length count is met
`C4` — fourth-Cclk rising edge after the length count is met
`U2` — second valid user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
`U3` — third valid user-clock rising edge after C1 (first-Cclk rising edge after length count is met)`U4` — fourth valid user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
`DI` — when the DoneIn signal goes High
`DI_PLUS_1` — first-Cclk or valid-user-clock rising edge (depending on selection of StartupClk) after DoneIn goes High
`DI_PLUS_2` — second-Cclk or valid-user-clock rising edge, depending on selection of StartupClk, after DoneIn goes High

Default: `C4`

Note: You can only select DI, DI_PLUS_1, and DI_PLUS_2 when SyncToDone is yes.

–help Display the Usage Help Screen

This option displays the help screen for the MakeBits program and its associated options.

–i Ignore Critical Net Flags

This option ties the design using critical nets. The -i option uses critical nets in the 'normal' tie order. It tells TIE to ignore critical net flags and use any nets that are flagged as critical whenever it is necessary to do so. These critical nets are not used as a last resort; instead, they are used in the normal TIE order.

–l Create a Logic-Allocation (design.ll) File

This option creates a logic allocation file (design.ll) for the selected design. Select this option if you use XChecker to verify or probe the internal logic of a design.

The logic-allocation file indicates the bitstream position of latches, flip-flops, and IOB inputs and outputs. Other Xilinx programs, such as XChecker, use this file to locate signal values inside a readback bitstream.

–m Generate a Mask File

This option causes MakeBits to create a MSK file (masked bitstream) that contains information about flip-flop and RAM bits in the configuration bitstream. These bits represent the current state of each flip-flop and IOB slip-flop, and can be ignored during readback.The MSK file contains a 1 for each of these bits, and can be used to mask them when a "readback" bitstream is compared to the bitstream programmed into the FPGA. The MSK file has the same format as a BIT file.

Refer to the XC2000 and XC3000 FPGA readback procedure information about readback verification and signature analysis in the *Development System User Guide.* Some of the bits read back from the FPG0..... A device are not the same as the bits in the original configuration bitstream. These bits represent the current

state of each flip-flop and IOB flip-flop, and must be ignored during readback. The Mask option creates an ASCII file that contains a 0 in each bitstream location to be ignored.

For XC4000 or XC5200 designs, you can disable the ReadCapture option to mask out flip-flop values in the readback bit stream. A mask file is only needed if bitstream verification and readback of flip-flop states are performed on the same device.

If the bitstream mask file is stored in an EPROM, you can use the MakePROM program to create a HEX file from the mask file. Since the MakePROM program requires a file extension of .bit, the mask file, *design*.msk, must be renamed to *design*.bit.

Do not use the -m option with the -b option. If you want both a mask file and a rawbits file, run MakeBits twice, once with the -m option and once with the -b option.

Note: If you specify both the -b and -m options, and only run the program once, MakeBits ignores the -m option.

–n Save a Tied Design

This option, used with the Tie (-t) option, saves the tied FPGA design file as *_design*.lca, where *design* is the name of the input FPGA design file. When you start MakeBits with the Tie option, all unused interconnect is tied to a known level. The -n option saves the tied design so it can be processed for simulation or other similar post-processing. The *_design*.lca file contains all of the nets created by the Tie option.

Note: In the PC environment, file names are limited to eight characters. Please notice that the -n prefixes an underscore at the beginning of the original file name. The new file name is truncated to eight characters.

–o Rename a Configuration Bitstream File

This option specifies a file name for the output BIT file. If you do not use this option, the BIT file is called *design*.bit, where *design* is the name of the input FPGA design file.

You can only specify one output file name with the -o option. Do not use this option when MakeBits generates multiple bitstreams, as successive bitstreams are written to the same file.

–p Disable Internal Pull-up for Done Pin

This option disables the internal pull-up resistor on the Done/$\overline{\text{Program}}$ (D/$\overline{\text{P}}$) pin in the XC3000 and XC2000 devices or the Done pin in XC4000 or XC5200 devices. Use this option only if you are planning to connect an external pull-up resistor to this pin. The internal pull-up resistor has a value of 2 to 8 kΩ and is automatically connected if you do not use this option. The Done (XC4000 or XC5200) or D/$\overline{\text{P}}$ (XC2000/XC3000) pins configure an open-drain driver that requires a pull-up resistor to indicate the end of the configuration.

–r (0|1|2) Specify Readback Options (XC2000 and XC3000 Only)

This option specifies readback options for XC2000 and XC3000 families only. After the FPGA design has been configured, the FPGA configuration data can be read back and compared with the original configuration data. Readback is initiated by a Low-to-High transition on the M0/RTRIG pin. Once you give the readback command, external logic must drive the Cclk input to read back each data bit. The readback data appears on the $\overline{\text{Rdata}}$ pin. Refer to the *Development System User Guide* for more information about readback verification and signature analysis.

The r0 option disables readback. The r1 option enables a one-time readback and r2 enables readback on command. Readback on command is the default option.

The r0 and r1 options are used for design security. The r1 option allows only one readback, typically performed during manufacturing. After this, readback can never be invoked again. If the FPGA device is powered by a stand-by battery and the configuration source is removed, the FPGA design configuration data is completely secure from being read or copied.

–s (0|1|2) Set up Crystal Oscillator (XC3000 Only)

This option specifies crystal-oscillator options for XC3000-series devices. The crystal oscillator is associated with the auxiliary clock buffer in the lower-right corner of the die, as seen in EditLCA.

The s0 option disables the FPGA crystal oscillator; s1 enables it. The s2 option enables the oscillator and divides the crystal output frequency by two in order to guarantee a symmetrical clock signal. The default option is s0.

Note: In the XC2000 family, the crystal oscillator is enabled or disabled by Programmable Interconnect Points (PIPs) in the routing, and is automatically enabled whenever its use is detected in the design. The XC4000 devices do not have crystal oscillators.

–t Tie Unused Interconnects

This option produces configurations that minimize internal noise and power consumption that can result from undefined levels on CMOS gate inputs. The Tie option causes all unused interconnects to be tied Low only, to User net Low, or to a defined signal. When you specify this option, the XACT DRC is started before the interconnect is tied. A message appears on the screen if any errors are found; messages for these errors are sent to a file named *design*.drc. Any fatal DRC error causes the MakeBits program to abort. DRC warnings do not abort MakeBits, but might cause tiedown to fail.

Note: You must start MakeBits with the -tie and -norestore options inside XDE for the Querynet -Tiechange option to work.

Following the DRC, the Tie option causes interconnect to be tied Low as follows.

- As many unused interconnect lines as possible are tied to unused CLB outputs that are configured to output logic 0 (F=0 or G=0).
- An attempt is made to tie any remaining interconnect to existing CLB outputs that were not marked critical in XACT with the Flagnet Critical command set to on. This is not the Flagnet Critical command used in an APR or PPR constraints file.

  To flag a net critical in XDE go to the XACT Editor (with EditLCA), select FlagNet from the Net menu, select critical, then select the nets you want marked as critical.
- An attempt is made to tie remaining interconnect to the global or the auxiliary clock buffer outputs.

Tie does not add interconnect, under any circumstances, to a net that meets any of the following requirements.

- The net is marked Critical (see below).
- The net is sourced by 3-state buffer outputs. Tie assumes that the timing on such nets is critical and must not be disturbed.
- The net is sourced by a XC3000 series IOB Q pin.

In addition, Tie does not add a XC3000, XC4000, or XC5200 series 3-state buffer I (input) or T (3-state) pin to a net, whether or not that net was created by Tie. These pins are left unconnected to ensure that no internal contention is created.

Adding interconnect to used CLB or buffer outputs might add delays to the nets connected to these outputs. You can use the Flagnet Critical command in the XACT system to guarantee that a net is unaffected by tiedown. However, if too many nets are flagged critical, the tiedown process might fail.

Each time an interconnect is tied to a user-defined net, a message is printed giving the name of the PIP that was programmed. The delay of the net associated with that source might be altered.

When the MakeBits Tie option cannot tie certain pins, it issues error messages. You should not ignore these messages; MakeBits does not generate a partially tied design. You must edit the design so that MakeBits can complete the tiedown process, or remove the Critical flag on nets.

The error messages in MakeBits give very specific information about the untied PIPs (interconnects) in the design. You should edit the design to eliminate the obstacles causing Tie to fail, in one of two ways.

- Re-evaluate nets flagged as critical in EditLCA. This problem is the most common cause of Tie failure.
- Manually re-route a net that cannot source (tie) the untied PIP.

For example, a pin can have several input PIPs that can be used to source the pin. If all these PIPs are associated with critical nets, Tie does not use them and the input pin is left untied. To correct this problem, make one of the critical nets non-critical using the Flagnet Noncritical command in EditLCA. Any delay added to the net by the tiedown process can be evaluated with the Querynet Tie change command.

As stated earlier, the MakeBits Tie option ties unused interconnect, internal to the FPGA, to a known level. In addition to unused interconnect, unused package pins must have defined levels; they must not be left floating. In XC2000-series FPGA devices, there are two ways to define the logic level of an unused package pin: unused IOBs can be configured as outputs and driven by internally generated High or Low signals (F=1 or F=0), or they can be left unconfigured, as long as the package pins are externally connected to a High or Low level or to an available signal. In XC3000-, XC4000-, and XC5200-series FPGAs, unused IOBs are automatically configured as inputs with pull-up resistors.

–u Use Critical Nets Last

This option uses the nets marked as critical to complete the tiedown process if necessary. Critical nets are used as a source of last resort, after an attempt is made to use nets not marked critical.

Note: In the example shown in Figure 7-2, the tag <tie> means a segment was added to the original net. In most cases the delay before and after tie remains unchanged. Querynet still reports every net touched by tie.

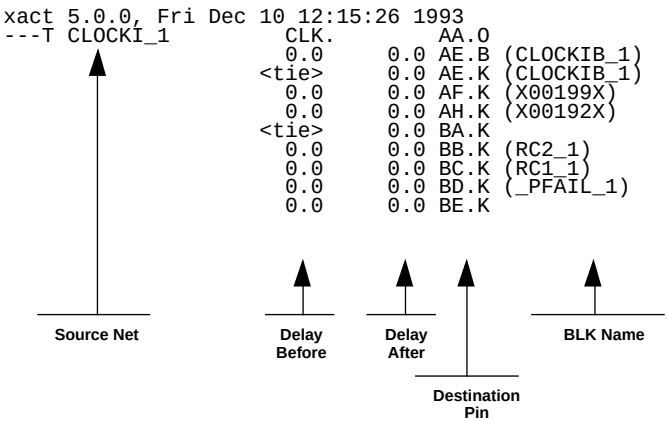


Figure 7-2 Using -querynet with -tiechange Option to Determine Slower Nets

–v Verbose Mode

This option displays status messages on the screen while MakeBits is running. This option permits you to track the progress of MakeBits. The -v option should precede any other option.

–w Rewrites FPGA Design File

Rewrites the FPGA design file after MakeBits has retimed all nets in the design. Alternatively, you can use the -w option with XDelay. Refer to "The XDelay Timing Analysis Program" chapter in the *Development System Reference Guide* for more information.

–xa Late Done (XC3000 Only)

This option specifies that the D/$\overline{P}$ pin goes High one Cclk period after the outputs go active. In XC2000-series devices, D/$\overline{P}$ always goes High one Cclk period after the outputs go active.

Late Done clearly indicates the end of the configuration process. Early Done can be used to de-activate external configuration drivers so that they do not contend with active outputs on the same pin. The use of Late Done would create a 1-Cclk-period contention. The alternative, using the $\overline{LDC}$ output, might cause a short contention spike. Early Done avoids these problems.

Late Done cannot be used in XC3000 slave devices driven from an XC2000 master. The XC2000 generates only two more Cclk pulses after length count is reached, but a Late Done XC3000 device requires one additional Cclk pulse.

–xb Early Done (XC3000 Only)

This option specifies that the D/$\overline{P}$ pin goes High one Cclk period before the outputs go active. This option is the default.

–ya Late Reset (XC3000 Only)

This option specifies that the Internal Reset on XC3000 devices be removed one Cclk period after the outputs go active. In XC2000-series devices, the Internal Reset is always removed one Cclk period after the outputs go active.

With the -ya option, the outputs go active while all internal flip-flops are still being held in Reset. This option is the default.

–yb Early Reset (XC3000 Only)

This option specifies that the Internal Reset on XC3000 devices be removed one Cclk period before the outputs go active.

When you specify the -yb option, the internal logic becomes operational before the outputs go active.

–mbo=file Use Specified MBO file as MakeBits Options

This option reads an MBO file when MakeBits generates a bitstream file. This file contains CONFIGURE commands that record the state of all configuration options at the time the bitstream was made. This file is read before MakeBits applies any -f command line options, permitting values in the file to be overridden by the command line -f options.

Each time MakeBits is run it creates an .mbo file that records the set of configuration options that were used, including those options not explicitly set. This file can be specified to later runs of the MakeBits program using the -mbo=*file* option.

Stand-Alone Command Line Examples

This section shows three examples of the MakeBits syntax you can use on the command line.

The following command generates three configuration bitstreams, one for each of the three input files specified. The bitstreams are saved as input1.bit, input2.bit, and input3.bit.

```
makebits input1 input2 input3
```

The following command ties all unused interconnect in the input.lca file to a known value before the configuration bitstream is made (using the -t option). The tied FPGA design file is saved as _input.lca (using the -n option). The old tied design is saved as _input.odf.

```
makebits -n -t input
```

or

```
makebits -nt input
```

The following command creates a configuration bitstream called output.bit (using the -o option) from the input FPGA design file, input.lca. A mask file called output.msk is also created (using the -m option). The output.bit bitstream configures the target FPGA to have readback disabled (-r0) and input thresholds set to CMOS levels (using the -c option).

```
makebits -o output -m -c -r0 input
```

or

```
makebits -o output -mcr0 input
```

The following command enables the design input.lca CRC checking and the startup option uclk_sync set. Note that you use the -f option for each configuration setting.

```
makebits -f crc:enable -f uclk_sync input.lca
```

Running MakeBits from XDE

You can access MakeBits from XDE and take advantage of the menus in the graphic user interface. If you are not familiar with XDE, please review the "The XACT Design Editor" chapter in the *Development System Reference Guide*.

Enter either 'xde' or 'xact' at the system prompt to start XDE.

Use the Design command or choose from the Designs menu to select the appropriate design from the executive screen before using MakeBits.

To access MakeBits, type 'makebits' at the command prompt or select MakeBits from the Programs menu. The XDE MakeBits generates only bitstreams for one design at a time. To create bitstreams for multiple designs requires, you must exit MakeBits and select a different design from the XDE screen each time. When the bitstream file is created, you should save it using the Writebits command.

To exit MakeBits, type "exit" or select the Exit command in the MakeBits screen. Always check the Status Bar for error messages and warnings before exiting.

The MakeBits Screen

The MakeBits screen is divided into four sections: the Command line, the Status Bar, the Program Options box, and the Menu Headers (see Figure 7-3). At the Command line prompt, you type commands that you want to run. The Status Box indicates the name of the current design and the status of the bitstream (made, not made, written, and so forth). It is important to know the status of the bitstream, since some programs overwrite the bitstream buffer. The Program Options box provides a quick way for reviewing selected FPGA design configuration options, as well as changing such options by clicking on them with the mouse. See the Configure command.

The Menu Headers, at the top of the screen, provide interactive selection of commands. Commands, as well as headers, are divided into four groups (three only for the workstation) according to their function, as follows.

- Config menu — Contains commands for changing FPGA design-configuration options and generating the configuration bitstream.
- Download menu — Contains commands for Download cable self-test and operation (does not apply to workstations).
- Misc menu — Contains commands for saving and displaying information and generating reports.
- Profile menu — Contains commands for modifying screen and mouse settings.

Figure 7-3 shows the MakeBits Screen for the PC. The example shows the XC3000 family options.

```
Config Download Misc Profile
```

```
Program Options
                                    TTL CMOS
Input                               Disable Enable Div2
Done                                0 1 Cmd
Read                                Pullup No Pullup
XtalOsc                             Before After
DoneTime                            Before After
ResetTime

Cable: Xchecker        Design: TRYCLKLCA
Bit stream: not made

Cmd:
```

Figure 7-3 MakeBits Screen (PC only) with XC3000 Options

MakeBits Commands

The MakeBits commands are listed alphabetically. The command description includes usage information, the name of the menu, the keyboard syntax, as well as any abbreviations that apply to the keyboard syntax.

Configure — Change Configuration Options

Menu	Config
Syntax	configure [*option setting*]
Abbreviation	conf

The Configure command is used to set the options for MakeBits. You can change FPGA-device configuration options via changes in the bitstream. The configuration options that are available are dependent on the device family being used, for example, XC2000, XC3000, XC4000, or XC5200.

To select the configuration options, use the Config command or click the desired option in the Configuration Options box. You must run a

MakeBits command for the Configure command to take effect and be reflected in the bitstream.

XC2000 Configuration

The following options apply only to the XC2000 parts.

Input

Sets the FPGA input threshold level.

Settings: TTL, CMOS
Default: TTL

DonePad

Enables or disables an internal pull-up resistor on the FPGA D/$\overline{P}$ pin.

Settings: Pullup, Nopullup
Default: Pullup

Read

Permits readback of the configured FPGA device never, once, or on command.

Settings: 0 (never), 1 (once), CMD (on command)
Default: CMD

XC3000 Configuration

The following options apply only to the XC3000 devices.

DonePad

Enables or disables an internal pull-up resistor on the FPGA Done/$\overline{\text{Program}}$ pin.

Settings: Pullup, Nopullup
Default: Pullup

DoneTime

Selects the timing of Done to be one configuration clock before or after the IOB outputs become active.

Settings: Before, After
Default: After

Input

Sets the FPGA input threshold level.

Settings: TTL, CMOS
Default: TTL

Read

Permits readback of the configured FPGA device never, once, or on command.

Settings: 0 (never), 1 (once), CMD (on command)
Default: CMD

ResetTime

Selects the timing of release of the internal global Reset to be a configuration clock before or after the IOB outputs become active.

Settings: Before, After
Default: After

XTALOSC

This option allows the crystal oscillator to be disabled, enabled, or enabled with a divide-by-two.

Settings: Disable, Enable, Div2
Default: Disable

XC4000 and XC5200 Configuration

The following options apply only to the XC4000 and XC5200 devices.

CRC

Enables or disables CRC (Cyclic Redundancy Checking) during configuration.

Settings: Enable, Disable
Default: Disable

ConfigRate

Selects the configuration clock rate. There are two choices: slow or fast. Slow is equivalent to 1 MHz and fast is equivalent to 8 MHz (nominal).

Settings: Slow, Fast
Default: Slow

DonePin

Enables or disables an internal pull-up to the Done pin.

Settings: Pullup, Nopullup
Default: Nopullup

TdoPin

Adds either a pull-up or a pull-down to the TDO pin for Boundary Scan.

Settings: Pullup, Pulldown
Default: Neither

M1Pin

Adds either a pull-up or a pull-down to the M1 pin.

Settings: Pullup, Pulldown
Default: Neither

Note: The Pull-up and Pull-down options for the TDO and M1 pins act as toggles and are mutually exclusive. The default is inactive. Selecting one option enables it and disables the other. Selecting the same option a second time disables it.

BSReconfig (XC5200 Only)

Enable or disable reconfiguration via boundary scan.

Settings: `Enable, Disable`
Default: `Disable`

OscClk (XC5200 Only)

Drive oscillator with either user clock or internal oscillator.

Settings: `UserClk` (*user supplied*), `Cclk` (*internal*)
Default: `Cclk`

ReadCapture

Enables or disables inclusion of flip-flop and latch contents in the readback bitstream. If ReadCapture is disabled, the contents are then read back as Highs.

Settings: Enable, Disable
Default: Disable

Note: Logic levels of latches and flip-flops are inverted when they are written into the readback bitstream.

ReadAbort

Enables or disables aborting the readback sequence during the readback sequence.

Settings: Enable, Disable
Default: Disable

ReadClk

Sets the readback clock to be Cclk or a user-supplied clock (from a net inside the FPGA connected to the i pin of the rdclk block).

Settings: Cclk (*pin**), Rdbk (*user supplied*)
Default: Cclk

*In modes where Cclk is an output, this pin is driven by the internal oscillator.

StartupClk

Controls the post-configuration startup phase of the FPGA initialization with either a Configuration clock (Cclk) or a user-clock supplied clock (UserClk) that is connected to the CLK input of the startup block of the FPGA.

Settings: Cclk (*pin**), UserClk (*user supplied*)
Default: Cclk

*In modes where Cclk is an output, this pin is driven by the internal oscillator.

SyncToDone

Synchronizes the post-configuration startup phase of the FPGA initialization to the Done signal when you use the Yes setting. This option is useful in situations in which there are multiple XC4000 or XC5200 family devices in a daisy chain, which should all start operation synchronously.

Settings: Yes, No
Default: No

DoneActive

Defines the clock edge that is used to activate the Done signal at the end of the startup phase. There are a maximum of four events that you can select from at one time, depending on other selections made for StartupClk and SyncToDone. These events are Cclk edges or external (user) clock edges.

Settings:
- `C1` – first-Cclk rising edge after the length count is met
- `C2` – second-Cclk rising edge after the length count is met
- `C3` – third-Cclk rising edge after the length count is met
- `C4` – fourth-Cclk rising edge after the length count is met
- `U2` – second-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- `U3` – third-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)

`U4` – fourth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)

Default: `C2`

OutputsActive

Defines the clock edge that is used to de-activate the internal Global Set/Reset signal at the end of the startup phase. There are four events that you can select from at one time, depending on other selections made for StartupClk and SyncToDon. These events are Cclk edges, external (user) clock edges and the external signal DoneIn.

Settings:
- `C2` – second-Cclk rising edge after the length count is met
- `C3` – third-Cclk rising edge after the length count is met
- `C4` – fourth-Cclk rising edge after the length count is met
- `U2` – second-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- `U3` – third-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- `U4` – fourth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- `DI` – when the DoneIn signal goes High
- `DI_PLUS_1` – first-Cclk or valid-user-clock rising edge, depending on the selection of StartupClk, after DoneIn goes High
- `D1_PLUS_2` – second-Cclk or valid-user-clock rising edge, depending on the selection of StartupClk, after DoneIn goes High

Default: `C3`

GSRInactive

Selects the event that releases the internal set-reset to the latches and flip-flops. There are four events that you can select from at one time, depending on other selections made for StartupClk and SyncToDone. These events are Cclk edges, external (user) clock edges and the external signal DoneIn.

Settings:
- `C2` – second-Cclk rising edge after the length count is met
- `C3` – third-Cclk rising edge after the length count is met
- `C4` – fourth-Cclk rising edge after the length count is met
- `U2` – second-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)

`U3` — third-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
`U4` — fourth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
`DI` — when the DoneIn signal goes High
`DI_PLUS_1` — first-Cclk or valid-user-clock rising edge, depending on the selection of Startup Clk,afterDoneIn goes High
`D1_PLUS_2` — second-Cclk or valid-user-clock rising edge, depending on the selection of StartupClk, after DoneIn goes High

Default: `C4`

Note: DI, DI_PLUS_1, and DI_PLUS_2 can only be selected when SyncToDone is yes.

Defaults — Select From Four Startup Defaults

Menu Config

Syntax defaults *startup sequence*

Abbreviations def

The Default command lets you select from one of four predetermined startup sequences that define the settings of StartupClk, SyncToDone, DoneActive, OutputsActive and GSRActive. Refer to the startup section of *The Programmable Logic Data Book,* to a see the diagram of the startup timing for each Xilinx device family. These startup sequences are described below.

Cclk_Nosync

This startup sequence makes an XC4000 or XC5200 device consistent with an XC3000 device set for early Done and late Reset. Selecting this sequence causes the following defaults to take effect.

StartupClk: `Cclk`
SyncToDone: `No`
DoneActive: `C1`
OutputsActive: `C2`
GSRActive: `C3`

Cclk_Sync

This startup sequence is the most consistent with the XC2000 and XC3000 devices, since it synchronizes the release of GSR and I/Os to the Done signal. Selecting this sequence causes the following defaults to take effect.

StartupClk:	`Cclk`
SyncToDone:	`Yes`
DoneActive:	`C3`
OutputsActive:	`DI`
GSRActive:	`DI_PLUS_1`

Uclk_Nosync

This startup sequence makes XC4000 or XC5200 devices inconsistent with XC3000 or XC2000 devices in the same daisy chain, since the release of Done is synchronized to an external user clock. There is no synchronization of I/Os or GSR to DoneIn. Selecting this sequence causes the following defaults to take effect.

StartupClk:	`UserClk`
SyncToDone:	`No`
DoneActive:	`U2`
OutputsActive:	`U3`
GSRActive:	`U4`

Uclk_Sync

This startup sequence makes XC4000 or XC5200 devices inconsistent with XC3000 or XC2000 devices in the same daisy chain, since the release of Done is synchronized to an external User Clock. I/Os and GSR are synchronized to DoneIn. Selecting this sequence causes the following defaults to take effect.

StartupClk:	`UserClk`
SyncToDone:	`Yes`
DoneActive:	`U2`
OutputsActive:	`DI_PLUS_1`
GSRActive:	`DI_PLUS_2`

DOS — Enter Temporary DOS Shell (PC Only)

Menu Misc

Syntax dos *dos_command*

Abbreviations none

The DOS command temporarily leaves the MakeBits program and passes control to DOS. To return to the program, type 'exit' at the prompt.

When the specified DOS command finishes, control is returned to MakeBits automatically.

Download — Transfer the Current Bitstream to an FPGA

Menu Download

Syntax download

Abbreviations downl, dl

See also: Port, Writebits, MakeBits, Selftest

The Download command in the Download menu transfers the current bitstream to an FPGA device via a Download cable.

You must use the MakeBits command to make a bitstream file for the current design. For PCs only, use the Port command to select the cable port. If you want to use your own download routine, you need to set the port to User with the Port command. See the Port command later in this chapter for details about using customized routines in place of the default Download program.

DRC — Invoke the Design Rules Checker

Menu Misc

Syntax DRC *options*

Abbreviations none

See also: Report

The DRC command starts the Design Rules Checker for the current design to check the design for design rules violations. The results

display as the check progresses. To list the results in a file rather than on the screen, use the Report command.

The DRC command is identical to the DRC command in the XACT Design Editor and the Executive. The DRC options are described below.

Note: If everything needs to be checked, just select DRC and Done. The Net and Block options are usually used to restrict the amount of checking.

Net

Checks only the specified net. Use * as a wildcard. For example, DRC -net g* causes DRC to check only those nets with names that begin with the character "g."

Nonet

Skips checking the nets of a routed design.

Block

Checks only the specified block. Use * as a wildcard. For example, DRC -net g* causes DRC to check only those blocks with names that begin with the character "g."

Noblock

Prevents block checking from being performed.

Noroute

Checks the block but not for routing errors. This option is useful when you have not yet completed routing.

Verbose

Provides running comments of the MakeBits program progress.

Informational

DRC informs of irregularities and general hints that can provide better device utilization.

Execute — Perform Commands from a Command File

Menu	Misc
Syntax	execute *filename parameter ...*
Abbreviations	exec

Execute performs the XDE commands in a command file (text file) (except Exit) in the order in which they appear in the file. When the end of the file or the Endfile command is encountered, command input transfers back to the keyboard or mouse.

Exit — Return to the XACT Executive

Menu	Misc
Syntax	exit
Abbreviations	none

Exit terminates MakeBits and returns control to the Executive.

Note: If a MakeBits Tie Norestore command has been executed without a Restore command afterwards, Exit issues a warning.

Keydef — Define a Function Key

Menu	Misc
Syntax	keydef *keyname_definition*
Abbreviations	ke
See Also:	Saveprofile, Readprofile, Settings

The key name must be one of the IBM PC function keys, F2 through F12; F1 is reserved for help. You can use the Shift, Ctrl, or Alt key in combination with a function key; enter Shift, Ctrl, or Alt before entering the function-key name.

Pressing F1 displays help in the XACT MakeBits screen. When you place the cursor over one of the MakeBits configuration table tags, pressing F1 displays the online help for that tag.

The keystrokes you type as the function-key definition, terminated by ↵ or the backslash (\), are stored under that function key. Use function keys to store command sequences you use frequently.

For example, enter the following definition.

```
keydef f10 querynet -c
```

Pressing the F10 key displays the Querynet command and requests critical net information. When you enter the desired nets, the net information is displayed.

You can terminate a function-key definition using the Return key or backslash (\). If you terminate the keystroke sequence by pressing ↵, the carriage return is included in the key definition. Thus, if you define a key function as Mouse COM2 and press ↵, the command Mouse COM2 is typed and entered when you press the function key. If you terminate the keystroke sequence by pressing the backslash key (\), the key definition includes only the typed characters but not a carriage return. Thus, if you define a key function as Mouse B1 Done and press the backslash, Mouse B1 Done is typed when you press the function key, but you must press Return to execute the command.

To delete a function-key definition, enter Keydef, the keys, such as Ctrl F10, and press ↵ without entering a definition.

MakeBits has its own function-key settings independent of the Executive or other sub-programs. Initial settings for the MakeBits function keys are contained in the makebits.pro file. Changes to function-key definitions can be saved in the makebits.pro file using the Saveprofile command.

MakeBits — Generate a Bitstream for a Design

Menu	Config
Syntax	MakeBits *tie norestore verbose usecritical*
Abbreviations	makeb

The MakeBits command generates a bitstream for a design and stores it in the Write Bitstream buffer. The following options are available.

tie Tie Unused Interconnect

You should always use this option for production configurations to minimize internal noise and power consumption that can result from undefined levels on CMOS gate inputs. In a tied design, all unused interconnect is tied High or Low or to a defined signal. When the Tie option is specified, the XACT DRC is invoked before interconnect is tied. A fatal DRC error causes the MakeBits program to abort. DRC warnings do not abort MakeBits but can cause tiedown to fail.

Following DRC, the Tie option causes interconnect to be tied as follows.

- As many unused interconnect lines as possible are tied to unused CLB outputs that are configured to output logic 0 (F=0 or G=0).
- An attempt is made to tie any remaining interconnect to existing CLB outputs that were not marked critical in XACT with the Flagnet Critical command or flagged critical from the schematic. This is not the Flagnet Critical command used in an APR constraints file.
- An attempt is made to tie remaining interconnect to the global or the auxiliary clock-buffer outputs.
- If Tie cannot be completed successfully, MakeBits uses critical nets.

Tie does not add interconnect, under any circumstances, to a net that meets any of the following requirements:

- The net is sourced by an XC3000-series or XC4000-series IOB Q pin individually.
- The net is sourced by 3-state buffer outputs. Tie assumes that the timing on such nets is critical and must not be disturbed.

In addition, Tie does not add a XC3000-, XC4000-, or XC5200-series 3-state buffer I (input) or T (3-state) pin to a net, even if that net was created by Tie. These pins are left unconnected to ensure that no internal contention is created.

Adding interconnects to used CLB or buffer outputs can add delay to the nets connected to these outputs. Use the Flagnet Critical command in the XACT system to guarantee that a net is unaffected

by tiedown. If too many nets are flagged critical, however, the tiedown process can fail.

Each time interconnect is tied to a user-defined net, a message is printed giving the name of the PIP that was programmed. The delay characteristics of the net can be altered.

When the MakeBits Tie program cannot tie certain pins, error messages are issued. You should not ignore these messages; MakeBits does not generate a partially tied design. The design must be edited so that MakeBits can complete the tiedown process.

The error messages in MakeBits give very specific information about the untied PIPs in the design. You should edit the design to eliminate the obstacles causing Tie to fail. Two suggestions are as follows.

- Manually re-route a net that cannot source (Tie) the untied PIP.
- Re-evaluate nets flagged as critical from EditLCA.

For example, a pin might have several input PIPs, each of which could be used to source the pin. If all these PIPs are associated with critical nets, Tie does not use them and the input pin is left untied. To correct this problem, make one of the critical nets non-critical using the Flagnet Noncritical command in EditLCA. Any delay added to the net by the tiedown process can be evaluated with the Querynet Tie change command.

As stated earlier, the MakeBits Tie option ties unused interconnect, internal to the FPGA device, to a known level. In addition to unused interconnects, unused package pins must also have defined levels; they must not be left floating.

There are two ways to define the logic level of an unused package pin XC2000-series FPGA devices: unused IOBs can be configured as outputs and driven by internally generated High or Low signals (F=1 or F=0), or they can be left unconfigured, as long as the package pins are externally connected to a High or Low level or to an available signal. In XC3000-, XC4000-,and XC5200-series FPGA devices, unused IOBs are automatically configured as inputs with pull-up resistors.

Norestore

This option reflects the effects of tiedown on the timing of the design. Norestore causes the internal representation of an FPGA design to remain "tied." After tiedown is complete, you can use Querynet and XDelay to check timing. Restore the interconnect using the Restore command before exiting MakeBits.

To examine the tied design, load it into EditLCA and run the Querynet command. Querynet -Tiechange provides a listing of the nets that have been changed by the Tie process. Two delays are given for each net, the original net delay and the new, post-Tie delay. If the new delay is longer than the original delay, make sure the timing requirements of your design are still satisfied.

Note: If you run MakeBits with Tie Norestore options, original routing data is not restored until you execute a Restore command. As a consequence, if the design is saved, the routing added to implement Tie is saved as well. An appropriate warning is issued before you can save the tied FPGA file.

Verbose

This option displays running messages during the tiedown process.

IgnoreCriticalNetFlags

This option allows MakeBits to tie the design using critical nets. The -i option uses critical nets in the 'normal' tie order. It tells Tie to ignore critical net flags and use any nets that are flagged as critical whenever it is necessary to do so. These critical nets are not used as a last resort; instead, they are used in the normal Tie order.

UseCriticalNetsLast

This option allows the tiedown process to use critical nets. If too many nets have been flagged as critical from XDE, the tiedown might fail since there are insufficient resources to tie the unused interconnect. You can do one of three things if the tiedown fails.

- Use the FlagNet Noncritical command in EditLCA to flag some nets as non-critical.
- Use the UseCriticalNetsLast option (or the MakeBits -u command)

to tell MakeBits that it can tie the design using critical nets. This option builds a list of critical nets and uses them as a last resource.

- Use the IgnoreCriticalNetFlag option (or the MakeBits -i command) to tell MakeBits to ignore critical net flags during the tiedown operation. This option essentially removes the critical flags from all critical nets, and causes MakeBits to use these nets in the 'normal' way to tie interconnect, if it becomes necessary to do so. Note that these nets are not used as a last resource (as they would be if you had used the UseCriticalNetsLast option). The critical net flags are restored to the nets after the tiedown operation is complete.

Makell

The Makell option creates an ASCII file with a .ll extension that contains the locations of flip-flop outputs and CLB RAM bits in the current bitstream. In some applications, you might want to observe the contents of the FPGA internal registers at different times. The file created by the Makell option helps you identify which bits in the current bitstream represent outputs of flip-flops and latches. Bits are referenced by frame and bit number within the frame.

This is the same as the -l option in stand-alone MakeBits.

Makeconfigset — Create a Configuration Set

Menu	Config
Syntax	makeconfigset *name*
Abbreviations	makec

Makeconfigset takes the current MakeBits options, such as FPGA input levels TTL or CMOS, and creates a configuration set with the current architecture and specified name. The configuration sets are saved in the makebits.pro file as a sequence of 'Configure' statements, preceded by a Defineconfigset statement and closed with an Endconfigset statement.

Configuration sets are a convenient way to save and use multiple sets of MakeBits options. You can have multiple configuration sets at one time.

Use Queryconfigset to view the options in a configuration set.

Makemask — Write a Bitstream Mask to a File

Menu	Config
Syntax	makemask *filename*
Abbreviations	makem

Makemask is for FPGA-device designers who use a microprocessor in their FPGA design to periodically perform a configuration readback and compare during FPGA operation. You can use the FPGA readback facility to verify that an FPGA device has been properly configured. See the *Development System User Guide* for more information about readback verification and signature analysis.

Makemask writes the bitstream mask data to the specified file. The file extension must be MSK. The XACT system automatically appends the MSK extension if it is not supplied.

If a bitstream mask file is to be stored in an EPROM in your system, the MakePROM program can be used to create a HEX file from the bitstream mask file. Since the MakePROM program requires an extension of BIT, you must rename the MSK file to a BIT extension in order to use MakePROM to create a HEX version of the mask file.

Mouse — Change the Mouse Configuration

Menu	Profile
Syntax	mouse *[button function*]
Abbreviations	mou
See Also:	Saveprofile, Readprofile, Settings

Mouse defines the mouse button functions. You define these functions by selecting the button and the function.

Note: This command applies to a three-button mouse.

B1	Selects mouse button #1
B2	Selects mouse button #2
B3	Selects mouse button #3 (if applicable)

If you select B1, B2, or B3, you must also enter one of the four functions that determine the action of the selected button.

Select

Enters the cursor location. This is the default setting for all the mouse buttons.

Done

Applies the function you entered for the selected mouse button and returns you to the Select Mouse Options menu.

Menu

Displays the most recently displayed menu and moves the cursor to the most recently selected item. If a menu is already displayed, the mouse button enters the cursor location (same as Select).

Switch

This option is not relevant in MakeBits; any mouse button that is programmed as 'switch' is ignored.

Use the Saveprofile command to save the MakeBits configuration in the makebits.pro file. Use the Settings command to display the current configuration.

The following example defines button #1 as Done, and button #2 and button #3 as Select.

```
mouse b1 done b2 select b3 select
```

Port — Specify the XChecker/Download Cable Port

Menu	Misc
Syntax	port *options*
Abbreviations	port
See Also:	Download

Port selects the port used for downloading the current bitstream. The ports could be parallel ports (lptx) that are supported for the Download cable, or serial ports (COMx) that are supported with XChecker.

When you select MakeBits from XDE, it checks for the presence of the XChecker cable. If it is present, it is used, and the Port command shows only the User option.

If you select User, a prompt asks you to provide the name of your customized routine. For example, if the name of your customized routine is myprog, you would reply as follows.

```
Enter user command line: myprog
```

The executable for the routine myprog should be placed either in your working directory or in your search path. The download command then passes a temporary RBT file to your customized routine.

Note: The download command passes an RBT (ASCII) file to your customized routine. This file is a temporary file with no extension. See the Rawbits command for details on RBT files.

If your customized routine requires more arguments or options, you can indicate the place of the RBT file in the command line with a "%" (percent sign). For example, you can select User and reply to the prompt as follows.

```
Enter user command line: myprog % -port1 -speed1
```

This causes the download command to execute the following command string.

```
myprog temp_file -port1 -speed1
```

where temp_file is the temporary RBT file passed to your routine.

Note: When selecting a port, make sure you select a COM port that the mouse is NOT on, otherwise, you could make the mouse inoperable.

Print — Create a Printable File of Display Information

Menu	Misc
Syntax	print *options*
Abbreviations	none

Print stores screen or display information in a printable file with the specified file name. This command is a subset of the Print command in the XACT Design Editor.

Print options are Screen and Display, as follows.

- Screen information includes the entire screen.
- Display information includes only the center portion of the screen without menu headers.

Printer — Set the Printer Type for the Print Command

Menu	Misc
Syntax	printe *type*
Abbreviations	printe
See Also:	Print

The printer type determines the format of the printable file. Choose one of the printer types listed in the menu. "The XACT Design Editor" chapter lists the printers supported by XDE and describes how to implement drivers for printers not listed.

You can change the initial value for the printer type in the xact.pro or makebits.pro file. If a makebits.pro file does not exist, the default value for the printer is taken from the xact.pro file.

Queryconfigset — Display Configuration Sets

Menu	Config
Syntax	queryconfigset
Abbreviations	queryc

Queryconfigset displays all configuration sets that are available for all architectures. The status line on the MakeBits screen displays the name of the configuration set currently in effect. Using the mouse to modify the MakeBits options removes the current configuration set.

Querynet — Display Net Information for the Design

Menu	Misc
Syntax	querynet *option net*
Abbreviations	queryn, qn
See Also:	MakeBits

You can specify nets by name or with wildcards. You can use the option in conjunction with a wildcard to select a group of nets based on a common characteristic. Information is displayed for a net only if it is specified and matches all specified options.

Options	Characteristics
-ALL	All nets in design
-Inputs	Nets that include IOB I pin
-Outputs	Nets that include IOB O pin
-Threestates	Nets that include IOB T pin and TBUFs
-Unrouted	Net not completely routed
-Nosource	Net with no source pin
-Nodest	Net with no load pin(s)
-Critical	Nets flagged critical for Tie (not APR)
-Tiechange	Nets changed by -Tie Option
-Tieadd	Nets added by -Tie Option
-Baddly	Nets with tilde (~) delays
-Locked	Nets flagged as locked
-Unlocked	Nets not flagged as locked
-Delayless	Nets with a delay less than specified
-Delaygreater	Nets with a delay greater than specified
-Clocknets	Nets connected to CLB K, IOB IK or OK pins
-Probed	Nets connected to an external pad for use as a probe point

The wildcard (*) can be used to match any characters. For example, typing A* as a net name represents all net names that begin with the character A.

Querynet is identical to the Querynet command in the XACT Design Editor. Execution of this command allows you to examine the net delays of your design at this stage of the design process. Querynet is useful for determining any changes in net delays produced by tiedown by using the Tiechange option. You must choose the MakeBits Norestore option to see the tiedown effects in path delays using Querynet.

Rawbits — Create an ASCII Configuration File

Menu	Config
Syntax	rawbits *filename*
Abbreviations	raw

Rawbits produces an ASCII Configuration Bitstream to simplify inclusion of bitstream data into microprocessor source code.

Rawbits writes a design bitstream into an ASCII text file, as opposed to a binary file created by MakeBits**.** Designers using a microprocessor to initialize an FPGA design can use the Rawbits file as initialization data.

The bitstream data appears in the file in the same order as it should be written to an FPGA in the Peripheral Mode, with ASCII 1 and 0 corresponding to a binary 1 and 0, respectively. Some reformatting of the file is typically required, depending upon the software language used to implement the download routine, such as reformatting groups of eight ASCII bits into ASCII bytes, inserting punctuation, and so forth.

The file extension must be .rbt. The XACT system automatically appends the .rbt extension if it is not supplied.

Readbits — Read the Specified Bitstream File

Menu	Config
Syntax	readbits *file*
Abbreviations	readb

Readbits reads the specified bitstream data file into the bitstream buffer. The XACT system supplies the extension BIT if it is not typed as part of the file name.

Since the startup options are written into the bitstream, these options are restored as the bitstream is read, even if they are not displayed in the options box.

Readprofile — Update Profile

Menu	Profile
Syntax	readprofile
Abbreviations	readp
See Also:	Saveprofile, Settings

Readprofile sets options to the settings in the makebits.pro file. When the commands in the makebits.pro file, such as Mouse and Keydef, are executed, the MakeBits options are set. Use Saveprofile to create the makebits.pro file.

Report — Create a Report

Menu	Misc
Syntax	report *filename command*
Abbreviations	rep

Report saves block, net delay or other information in a text file. Use Report to send Querynet and DRC information to a text file instead of displaying it on the screen. The command must include Querynet or DRC followed by the required command parameters. Report is identical to the Report command in the XACT Design Editor. For the full syntax of each command, see Queryblk**,** Querynet, and DRC in "The XACT Design Editor" chapter in the *Development System Reference Guide*.

Restore — Restore Design to Untied State

Menu	Misc
Syntax	restore
Abbreviations	resto
See Also:	MakeBits

Restore returns tied interconnects to the unused state following bitstream generation using the Tie and Norestore options. When you use MakeBits with the Tie option (to tie down the unused interconnects), the Norestore option saves the interconnects in the

tied state so that you can examine the effects of tiedown using Querynet, Xdelay, and Report.

If you use the Norestore option, run Restore before exiting MakeBits.

Note: The tie process adds, and possibly modifies, nets in the original design. With NoRestore specified, these modifications can be saved to the disk or viewed in EditLCA. When saving the design to disk, specify a different name to the entered FPGA file. Upon exiting, a warning is issued if a Restore command has not been executed following MakeBits Tie NoRestore.

Saveprofile — Save Current Profile

Menu	Profile
Syntax	saveprofile
Abbreviations	savep
See Also:	Readprofile, Settings

Saveprofile saves the current setting of the MakeBits profile options in the makebits.pro file. The information saved in the file consists of the following information.

- FPGA configuration options
- Current mouse configuration
- Current function-key definitions

If a profile was previously read with Readprofile or created with Saveprofile, the profile is saved in the directory of the last profile read or saved. If no directories contain a makebits.pro file, the profile is saved in the current directory, if you answer “yes” when asked whether you want it saved in the current directory.

Commands in this file set the MakeBits options, such as Mouse and Keydef, and are normally executed automatically when MakeBits is first started.

Note: XC2000, XC3000, XC4000, and XC5200 profiles are essentially different. Appropriate warnings are issued if you attempt to read an incompatible profile.

Selftest — Test Download Cable (PC Only)

Menu	Download
Syntax	selftest
Abbreviations	selft

Selftest performs several circuitry checks on the Download Cable. You must select the cable port with Port before running Selftest.

Setconfigset — Apply Configuration Set to MakeBits Options

Menu	Config
Syntax	setconfigset *name*
Abbreviations	setcon

Setconfigset applies the specified configuration set to the MakeBits options. The options in the configuration set take effect with subsequent MakeBits commands as reflected on the MakeBits screen. The prompt menu displays only those configuration sets that are relevant for the current architecture.

Settings — Change Current Profile

Menu	Profile
Syntax	settings
Abbreviations	sett
See Also:	Saveprofile, Readprofile, Mouse, Keydef, and Configure

Settings displays the current MakeBits profile settings listed below:

- Current mouse configuration
- Current bitstream-configuration set up
- Current function-key definitions
- Current printer

Writebits — Save Current Configuration Bitstream

Menu	Config
Syntax	writebits *filename*
Abbreviations	write
See Also:	MakeBits

Writebits writes the contents of the bitstream buffer to the specified file. The file extension must be BIT. The XACT system automatically appends the BIT extension if it is not supplied.

Chapter 8

The MakePROM Program

This program is compatible with the following families.

- XC2000
- XC2000L
- XC3000
- XC3100
- XC3000A
- XC3000L
- XC3100A
- XC4000
- XC4000A
- XC4000H
- XC5200

MakePROM formats a MakeBits-generated configuration bitstream (BIT) file into a PROM format file. The PROM file contains configuration data for the FPGA device. MakePROM converts a BIT file into one of three PROM formats: MCS-86 (Intel), EXORMAX (Motorola), or TEKHEX (Tektronix).

There are two functionally equivalent versions of MakePROM. There is a stand-alone version you can access from an operating system prompt. There is also an interactive version you can access from inside the XACT Design Editor (XDE). This chapter first describes the stand-alone version, then the XDE version.

You can also use MakePROM to concatenate bitstream files to daisy-chain FPGAs.

Note: If the destination PROM is one of the Xilinx Serial PROMs, you are using a Xilinx PROM Programmer, and the FPGAs are not being daisy-chained, it is not necessary to make a PROM file. See the *Hardware and Peripherals User Guide* for more information about daisy-chained designs.

Stand-Alone MakePROM

You use the stand-alone MakePROM from the operating system shell to create a PROM-formatted file for one or several configuration bitstream files.

MakePROM creates two files: a PROM file containing the configuration information and a PROM image file containing a memory map of the created PROM file. The PROM file has the extension of .mcs, .exo, or .tek depending on the selected format. The image file has a PRM extension.

Syntax

To start the MakePROM program from the operating system prompt, use the following syntax.

`makeprom` *options*

Options is one of the options described in the next section.

Note: If you do not specify any parameters, MakePROM automatically goes into the interactive mode. See the "Using MakePROM in the XACT Design Editor" section in this chapter.

Options

This section describes the options that are available for the stand-alone version.

–d

This option formats one or more PROM files from the starting address in a downward direction. Specifying several files after this option causes the files to be concatenated in a daisy chain. You can specify multiple -d options to load files at different addresses. You must specify this option immediately before the input bitstream file.

Here is the multiple file syntax.

makeprom -d *hexaddress0 filename filename*...

Here is the multiple -d options syntax.

makeprom -d *hexaddress1 filename* **-d** *hexaddress2 filename*...

–f

This option sets the PROM format to one of the following: MCS (Intel (MCS86)), EXO (Exormax(Informix)), or TEK (Tekhex(Tektronix)). If specified, the -f option must precede any -u, -d, or -n options. Below is an example.

makeprom -f mcs -d *hexaddress filename*

–help

This option displays help that describes the MakePROM options.

–n

This option loads BIT files up or down from the next available address following the previous load. The -n option must follow after a -u, -d, or -n option. Files specified with this option are not be daisy-chained to previous files. Files are loaded in the direction established by the nearest prior -u, -d, or -n option.

The following syntax shows how to specify multiple files. When you specify multiple files, MakePROM daisy-chains the files.

makeprom -n *file1 file2*...

The syntax for using multiple -n options follows. Using this method prevents the files from being daisy-chained.

makeprom -d *hexaddress file0* **-n** *file1* **-n** *file2*...

–o

This option specifies the output file name of a PROM if it is different from the default. If you do not specify an output file name, the PROM file has the same name as the first BIT file loaded.

makeprom -d *hexaddress file0* **-o** *filename*

-s

This option sets the PROM size in kilobytes. The PROM size must be a power of 2. The default value is 64 K. The -s option must precede any -u, -d, or -n options.

makeprom -s *promsize*

Note: MakePROM PROM sizes are specified in bytes. The *Programmable Logic Data Book* specifies PROM sizes in bits.

-u

This option formats one or more PROM files from the starting address in an upward direction. When you specify several files after this option, MakePROM concatenates the files in a daisy chain. You can load files at different addresses by specifying multiple -u options.

This option must be specified immediately before the input bitstream file.

Here is the multiple file syntax.

makeprom -u *hexaddress0 filename filename*...

Here is the multiple -u options syntax.

makeprom -u *hexaddress1 filename* **-u** *hexaddress2 filename*...

-v

This option enables the verbose mode. It displays the running messages that MakePROM generates while it creates a BIT file. The following is an example syntax.

makeprom -v *other options filename*

Examples

To load the file test.bit up from address 0x0000 in MCS format enter the following information at the command line.

```
makeprom -u 0 test
```

To daisy-chain the files test1.bit and test2.bit up from address 0x0000 and the files test3.bit and test4.bit from address 0x4000 while using a

32K PROM and the Motorola Exormax format, enter the following information at the command line.

```
makeprom -s 32 -f exo -u 00 test1 test2 -u 4000
test3 test4
```

To load the file test.bit into the PROM programmer at address 0x400 enter the following information at the command line.

```
makeprom -d 400 test
```

To specify a PROM file name that is different from the default file name enter the following information at the command line.

makeprom *options filename* **-o** *newfilename*

Using MakePROM in the XACT Design Editor

You can use MakePROM in the XACT Design Editor. If you are not familiar with XDE, review "The XACT Design Editor" chapter in the *Development System Reference Guide*. It is not necessary to select a design prior to starting MakePROM.

Note: MakePROM is also available from the Verify menu of the XACT Design Manager (XDM). See the "The XACT Design Manager" chapter in the *Development System Reference Guide*.

To start MakePROM, enter makeprom at the XDE prompt or select MakePROM from the Programs menu. To exit MakePROM and return to the XDE screen, enter exit or select it from the Misc menu.

MakePROM Screen

The MakePROM screen has five parts: the command prompt, the status bar, the current directory bar, the PROM memory table, and menus. The workstation Make PROM screen is shown in Figure 8-1.

You can enter MakePROM commands at the command prompt, or use the mouse to select them from the menus.

For PCs, the status bar at the bottom of the screen displays the selected PROM format (MCS-86, EXORMAX, or TEKHEX) and the selected size of the current PROM in kilobytes.

The PROM memory table, located in the center of the screen, displays the addresses at which different design files have been placed. From

left to right, it displays the starting address, the ending address, and the name of the design or designs. Designs listed in the same line are concatenated daisy-chained.

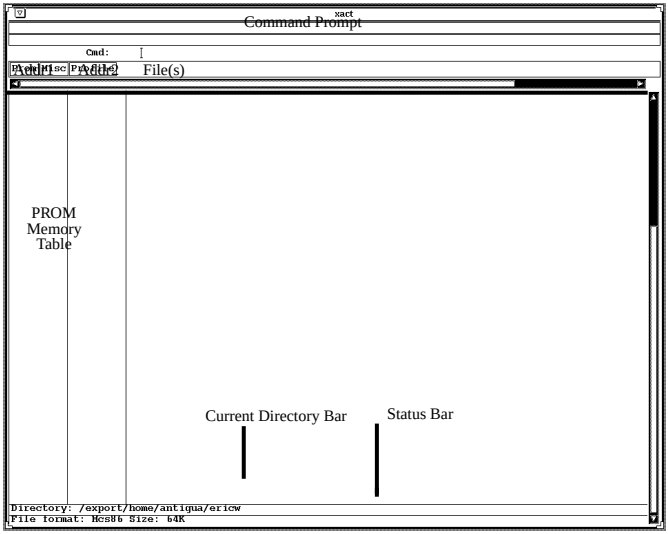


Figure 8-1 The MakePROM Screen (Workstation)

MakePROM Menus

MakePROM commands are nested under three menu headers: Prom, Misc, and Profile.

- Prom — This menu contains commands for specifying PROM formats, specifying the designs to load, saving the PROM file, and clearing designs from the PROM.
- Misc — This menu contains commands for online help, exiting, printing, information on PROM size, setting PROM size, and executing command files.
- Profile — This menu contains commands used to define screen, mouse, and function key defaults. The Profile menu also contains the Saveprofile command for writing over the default file, makeprom.pro.

A detailed description of each command follows in the “Command Descriptions” section.

Command Descriptions

Note: MakePROM commands operate only on BIT files that have been correctly made using the MakeBits program. If you specify an improper file, the system returns an error message.

This section lists the commands in alphabetical order; parameters are listed after syntax and option parameters are shown in square brackets.

Clear — Clear PROM

Menu	Prom
Syntax	clear
Abbreviations	none

Clear removes all files that are currently loaded into the PROM.

Delete — Remove File at Address from PROM

Menu	Prom
Syntax	delete *address* delete *segment:offset*
Abbreviations	d

Note: The segment**:**offset parameters are valid for the MCS-86 format only.

Delete removes files located at a specified address from the PROM memory files. Memory previously allocated for that file in the PROM memory file becomes available for other data.

When the PROM format is set to MCS-86, Delete addresses can be specified as a hexadecimal segment and offset, for example:

```
segment:offset
```

forms an address equivalent to

(*segment* **x 16) +** *offset*

DOS — Temporarily Suspend MakePROM and Enter Operating System (PC Only)

Menu	Misc
Syntax	dos *command*
Abbreviations	none

Use the DOS command to execute any DOS command without ending the current MakePROM session.

If you specify DOS without specifying a command, you get a DOS prompt. Entering Exit at the prompt returns MakePROM to the state you left it before you entered DOS.

If you specify DOS with a DOS command, MakePROM immediately executes the specified DOS command, then prompts you to press a key to resume the current MakePROM session.

A limited amount of memory is available when you use DOS in this manner. Some commands that require significant amounts of memory may not operate correctly and result in an error.

You should not run those DOS commands that cause programs to be installed in memory.

Directory — Change Working Directory

Menu	Misc
Syntax	directory *directory name*
Abbreviations	dir

Use this command to change the current working directory, without exiting to the operating system prompt.

Execute — Perform Commands in Command File

Menu	Misc
Syntax	execute *filename_parameter*
Abbreviations	exec

Execute causes MakePROM to run the commands in a specified command file. A command file is a text file that contains MakePROM

commands. MakePROM runs the commands in the order in which they appear in the file. When it reaches the end of the file, MakePROM can receive commands from the keyboard or mouse.

If a command file contains an Execute command, MakePROM runs a second command file within the first one. You can run up to eight nested command files in this fashion.

Parameters passed to command files are indicated with a %*n* (*n*=1, 2, 3…) within the command file, and refer to the number of the parameter in the command line. See Figure 8-2 for an example of how to use parameters within the Execute command.

Command file TEST:

```
set promsize  %1
load  %2  up  %3
load  %4  up  %5  %6
```

Input command line:

```
TEST 64  0000  design1  022c  design2  design3
```

Commands Performed

```
set promsize  64
load  0000  up  design1
load  022c  up  design2  design3
.
.
.
```

Figure 8-2 Example of an Execute Command

Exit — Quit MakePROM

Menu	Misc
Syntax	exit
Abbreviations	none

Exit ends the current MakePROM session. If you exit from XDE, control returns to the XDE Executive screen. If you exit from DOS, control returns to DOS. If the current PROM memory image has not been saved in a file, a message stating that the PROM file will be lost appears, and you are asked to confirm that this is acceptable.

Format — Select the PROM File Format

Menu	Prom
Syntax	format *format*
Abbreviations	f

Format sets the type of file format in which the PROM data are stored. The following are the available formats.

- MCS86 — Intel MCS-86 hexadecimal object
- EXORMAX — Motorola Exormax
- TEKHEX — Tektronix hexadecimal

The file format must be set before executing the Save command or MakePROM issues a warning message.

Keydef — Define a Function Key

Menu	Profile
Syntax	keydef [shift | ctrl | alt] *keyname*
Abbreviations	ke

Use this command to assign a command to a function key. The *keyname* value must be one of the function keys from F2 through F10. (F1 is reserved for Help). You can use the Shift, Ctrl, or Alt key in combination with the function key. To specify a combination key, select "shift," "ctrl," or "alt" before entering the *keyname*.

To delete a function key definition, enter "keydef *keyname,*" then ↵ or a new definition if you want to replace the old definition.

Load — Load a Bitstream File into PROM Memory at Specified Address

Menu	Prom
Syntax	load *address* up | down *filename* load *address:segment* up | down *filename*
Abbreviations	l

Note: The segment**:**offset parameters are valid for the MCS-86 format only.

When you use the Load option, the PROM file that is created is used to load the data, either up or down, from the specified address. Specifying Up means that the data is loaded from the given address toward a larger PROM address. Specifying Down means that the data is loaded from the given address towards 0.

Note: Load addresses are assumed to be in hexadecimal.

If you name more than one bitstream (BIT) file, MakePROM concatenates the files and includes the appropriate daisy chain information for loading multiple FPGA devices as a daisy chain. The length count in such a concatenated file is the sum of the length counts of all the concatenated bitstream files.

When the PROM format is set to MCS86, you can specify the Load addresses as either a hexadecimal address, or as an extended base address consisting of a hexadecimal segment and offset, where:

*segment***:***offset*

forms an address equivalent to

(*segment* **x 16) +** *offset*

The following command creates a PROM configuration file for the single bitstream in *design*.bit and loads it in incremental PROM addresses from location 0 up.

load 0 up *design***.bit**

The following command creates a PROM configuration for a daisy chain of FPGAs devices where *design1*.bit represents the first FPGA device in the chain, *design2*.bit the second, and *design3*.bit the third. The PROM addressing begins at location FFFF (hex) and is decremented.

load FFFF down *design1***.bit** *design2***.bit** *design3***.bit**

Mouse — Change the Mouse Configuration

Menu	Profile
Syntax	mouse b1 | b2 | b3 *function*
Abbreviations	mou

Mouse changes the mouse button functions. You define these functions by selecting the mouse button and then the function. A three-button mouse is needed for full functionality.

- B1 selects the left mouse button.
- B2 selects the middle mouse button.
- B3 selects the right mouse button.

After selecting a mouse button, you must also select one of four functions that determine the action of the selected button.

- Select enters the cursor location. This is the default setting for all the mouse buttons.
- Done applies the specified function and returns to the selection menu.
- Menu displays the most recently displayed menu and moves the cursor to the most recently selected item. If a menu is already displayed, the mouse button enters the cursor location; it is the same as Select.

Note: Read the mouse and cursor command descriptions in the "The XACT Design Editor" chapter of this reference guide for more information about configuring the mouse.

Print — Print the Current PROM Memory Image Screen

Menu	Misc
Syntax	print *options filename*
Abbreviations	pr

Note: See also the "Printer" section in "The XACT Design Editor" chapter of the *Development System Reference Guide*.

Print prints the MakePROM screen to the specified file. Three options are available with Print.

- Screen prints all the information that appears on the screen, including menus.
- Display prints the current working window.
- Map generates an ASCII file of the current PROM memory image.

Printing with the Screen or Display option generates a binary graphics file formatted to a specific type of printer. Printing with the Map option generates an ASCII text file containing the PROM table. You select the printer type from the XDE Executive screen with the Printer command.

Query — Display the Current Setting for the PROM Size

Menu	Misc
Syntax	query promsize
Abbreviations	que

Query displays the current PROM size in kilobytes.

Readprofile — Set MakePROM Options to Settings in makeprom.pro File

Menu	Profile
Syntax	readprofile
Abbreviations	readp

Readprofile executes the commands in the makeprom.pro file, which contains commands for configuring MakePROM such as Mouse, Set Promsize, and Format. Use Saveprofile to rewrite the makeprom.pro file with current settings.

Save — Save Currently Specified, Formatted Data into File

Menu	Prom
Syntax	save *filename*
Abbreviations	s

Save saves the currently specified and formatted data into the named file. Save also builds a default save file using the name of the first BIT file in the first daisy chain in the PROM. If you specify a file format and it is not MCS, EXO, or TEK, MakePROM displays an error message and does not save the data. You must use Format to set the file format before executing Save.

Saveprofile — Save Current MakePROM Options Settings to makeprom.pro File

Menu	Profile
Syntax	saveprofile
Abbreviations	savep

Saveprofile saves current MakePROM settings in the makeprom.pro file, which contains commands for configuring MakePROM, such as Mouse**,** Set Promsize, and Format**.** The commands in this file are automatically executed when you start MakePROM.

Set (PROMSize) — Specify PROM Size

Menu	Misc
Syntax	set promsize *size*
Abbreviations	none

Set specifies the size of the PROM to be loaded with the current data. It displays PROM sizes up to 32 megabytes and allows you to enter any size up to 2 gigabytes on the command line. PROM size must be a power of 2, as shown in the following example.

```
set promsize 32
```

sets the PROM size to 32 kilobytes. The default size is 64 K.

Set (Endclocks) — Modify Daisy-Chain Length

Menu	Misc
Syntax	set endclocks *n*
Abbreviations	none

You can modify the length count that is embedded in each daisy-chain that is built using the Set endclocks command. The actual length of the daisy chain is unchanged, since no padding 1s are added. You can use this whether you have one BIT file or multiple BIT files. The value is part of the output from Settings and SaveProfile.

Settings — Display Current Values of MakePROM Settings

Menu	Profile
Syntax	settings
Abbreviations	sett

Settings displays the value of MakePROM settings, such as Set Prom, Mouse, and Format.

Index

Symbols

A

B

L

M

N

O

P

R

Trademark Information

XILINX®, XACT, XC2064, XC3090, XC4005, and XC-DS501 are registered trademarks of Xilinx. All XC-prefix product designations, XACT-Floorplanner, XACT-Performance, XAPP, XAM, X-BLOX, X-BLOX plus, XChecker, XDM, XDS, XEPLD, XPP, XSI, BITA, Configurable Logic Cell, CLC, Dual Block, FastCLK, HardWire, LCA, Logic Cell, LogicProfessor, MicroVia, PLUSASM, SMARTswitch, UIM, VectorMaze, VersaBlock, VersaRing, and ZERO+ are trademarks of Xilinx. The Programmable Logic Company and The Programmable Gate Array Company are service marks of Xilinx.

IBM is a registered trademark and PC/AT, PC/XT, PS/2 and Micro Channel are trademarks of International Business Machines Corporation. DASH, Data I/O and FutureNet are registered trademarks and ABEL, ABEL-HDL and ABEL-PLA are trademarks of Data I/O Corporation. SimuCad and Silos are registered trademarks and P-Silos and P/C-Silos are trademarks of SimuCad Corporation. Microsoft is a registered trademark and MS-DOS is a trademark of Microsoft Corporation. Centronics is a registered trademark of Centronics Data Computer Corporation. PAL and PALASM are registered trademarks of Advanced Micro Devices, Inc. UNIX is a trademark of AT&T Technologies, Inc. CUPL, PROLINK, and MAKEPRG are trademarks of Logical Devices, Inc. Apollo and AEGIS are registered trademarks of Hewlett-Packard Corporation. Mentor and IDEA are registered trademarks and NETED, Design Architect, QuickSim, QuickSim II, and EXPAND are trademarks of Mentor Graphics, Inc. Sun is a registered trademark of Sun Microsystems, Inc. SCHEMA II+ and SCHEMA III are trademarks of Omation Corporation. OrCAD is a registered trademark of OrCAD Systems Corporation. Viewlogic, Viewsim, and Viewdraw are registered trademarks of Viewlogic Systems, Inc. CASE Technology is a trademark of CASE Technology, a division of the Teradyne Electronic Design Automation Group. DECstation is a trademark of Digital Equipment Corporation. Synopsys is a registered trademark of Synopsys, Inc. Verilog is a registered trademark of Cadence Design Systems, Inc.

Xilinx does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx 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 will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx devices and products are protected under one or more of the following U.S. Patents: 4,642,487; 4,695,740; 4,706,216; 4,713,557; 4,746,822; 4,750,155; 4,758,985; 4,820,937; 4,821,233; 4,835,418; 4,853,626;

4,855,619; 4,855,669; 4,902,910; 4,940,909; 4,967,107; 5,012,135; 5,023,606; 5,028,821; 5,047,710; 5,068,603; 5,140,193; 5,148,390; 5,155,432; 5,166,858; 5,224,056; 5,243,238; 5,245,277; 5,267,187; 5,291,079; 5,295,090; 5,302,866; 5,319,252; 5,319,254; 5,321,704; 5,329,174; 5,329,181; 5,331,220; 5,331,226; 5,332,929; 5,337,255; 5,343,406; 5,349,248; 5,349,249; 5,349,250; 5,349,691; 5,357,153; 5,360,747; 5,361,229; 5,362,999; 5,365,125; 5,367,207; 5,386,154; 5,394,104; 5,399,924; 5,399,925; 5,410,189; 5,410,194; 5,414,377; RE 34,363, RE 34,444, and RE 34,808. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.