Application Note

Designing for Xilinx LCAs with FutureNet

Section 1: Entering an LCA Design Using FutureNet

Contents

This section includes information you need to enter an LCA design with FutureNet, and is divided according to the following categories.

- Before You Begin — Information on setting up your schematic, using title blocks and creating multiple-page drawings.
- Creating an LCA Design Using FutureNet — This section provides detailed information about creating functional blocks, using naming conventions, and adding various characteristics to your LCA design.
- Using Xilinx Primitives and Macros — Xilinx primitives and macros are the building blocks of an LCA design. This section provides information to assist you in using these library elements to create an LCA design.

Before You Begin

Setting Up Your Schematic

Before beginning your LCA design, you should be aware of two important aspects of FutureNet design hierarchy and documenting your design.

- Design Hierarchy — Set up your design hierarchically, using functional block symbols to create a hierarchical design file.
- Documenting Your Design — Set up your design in a manner that allows you to track design revisions accurately and to create design documentation that is consistent in appearance. FutureNet provides title block primitives to assist you in this process.

Using Title Blocks

Using title blocks allows you to select items such as the LCA part type, document, and design revisions. The title block primitives in the Xilinx library are listed in the following table.

Title Block	Size	Dimension	Border
TBLOCKA	A	8-1/2 x 11	Yes
TBLOCKB	B	11 x 17	Yes
TBLOCKC	C	17 x 22	Yes
TBLOCKD	D	22 x 34	Yes
TBLOCKCP	C	17 x 22	Yes
TBLOCKDP	D	22 x 34	Yes
TBLOCK	B	11 x 17	No
TITL	Same as TBLOCKB (XC2000 family only)		

Each title block contains a default string (attribute 81) that you can edit to reflect the desired part type. This part type is passed on to the XACT implementation software.

Note: *You can use the -P option on the PIN2XNF command line to override any part type text in the title block.*

Creating Multiple-Page Drawings

The XMAKE Automatic Design Translator runs DCM correctly if multiple-page drawings are properly identified. Multiple pages must have the same filename with sequentially-numbered extensions (.D01, .D02) instead of the default DWG extension. For example, if your top-level schematic (TOP) contains three pages, name these drawings TOP.D01, TOP.D02, and TOP.D03. If a lower-level module called MYMAC consists of two pages, the symbol name (File (8)) should be MYMAC and the two MYMAC schematics should be named MYMAC.D01 and MYMAC.D02.

When processing the MYMAC logic, XMAKE recognizes from the .D01 and .D02 extensions that MYMAC is a multiple-page schematic. XMAKE then automatically runs DCM with the following command line.

```
Cmd:DCM MYMAC.D01 MYMAC.D02
```

The resulting DCM file is called MYMAC.DCM and contains the logic from both pages. XMAKE automatically merges the MYMAC logic into the TOP logic at a later step in the translation process.

Creating an LCA Design Using FutureNet

FutureNet Attributes

FutureNet Attributes Allowed by PIN2XNF

The PIN2XNF program allows only a subset of the attributes provided by FutureNet. The following FutureNet attributes are used by PIN2XNF to translate a FutureNet design to an XNF file.

#	Attr.	Description
0	COM	Assigns a comment, which is ignored by PIN2XNF.
2	LOC	Assigns an instance name to a symbol. Although LOC (2) names are not required, they must be unique if used.
3	PART	Defines a Xilinx primitive symbol. Do not create text with the PART (3) attribute.
5	SIG	Assigns a signal name to a net. Used for both individual nets and bus signals.
8	FILE	Assigns a filename to a functional block symbol. Each functional block must carry one and only one FILE (8) name. Xilinx macros are named with an attribute 8.
21	PINO	Defines an output pin name.
22	PNBT	Defines a bidirectional pin name.
23	PINI	Defines an input pin name.

Note: *FutureNet reserves attributes 80 through 99 for use by application programs, such as PIN2XNF.*

#	Description
80	Assigns CLB locations to flip-flops and CLB primitives, and IOB locations to I/O pads and IOB primitives. For XC4000 devices, attribute 80 labels can be used in place of some LOC= (attribute 83) parameters.
81	Assigns LCA part type.
82	Defines configuration statements on CLB and IOB primitives.
83	Assigns special LCA options.

Assigning Attributes

Each alphanumeric field in FutureNet is assigned an attribute, whether it's the default attribute, - or one that gives more information about the field. The easiest way to assign an attribute for an alphanumeric field is to assign the attribute before you begin typing.

There are two ways to assign attributes to alphanumeric fields:

- For existing fields, use 'CH A to change the attribute currently assigned to the field.
- For new fields, use 'A to set the attribute before typing the alphanumeric data. Then that attribute is automatically assigned to each new field.

To change the text of an alphanumeric field, place the mouse cursor on the text and use 'R or press Esc to enter the text editing mode.

Creating Functional Blocks

A hierarchical module in FutureNet is represented by a functional block symbol. A functional block can have any number of pins, each of which represents an input to or an output from the hierarchical module.

The functional block symbol itself is created by using the .F command. The format of this command is as follows.

```
.F width  height  border_width  border_height
```

The first two parameters, *width* and *height*, determine the dimensions of the block symbol itself. The last two parameters, *border width* and *border height*, specify the size of the border area around the symbol. The pins on the functional block are displayed in this border area.

To add pins to the functional block, use the following commands.

Command Line Entry	Function
.-	Add regular pin stub.
.>	Add clock pin stub.
.-O	Add inverted (bubbled) pin stub.
.>O	Add inverted (bubbled) clock pin stub.
.D	Delete pin stub.

The different types of pin stubs are provided only as a graphical convenience. Each pin stub has the same meaning to the translation software.

Naming Functional Blocks

Once a functional block has been created, both the symbol itself and its associated pins must be named. Use the following conventions when assigning names to a functional block for an LCA design.

Each functional block must be identified by a single label which carries the FILE (8) attribute. This label represents the name of the functional block symbol, as well as the name of the underlying drawing. Therefore, the FILE (8) name must be a valid DOS filename of 8 characters or less. Do not specify an extension as part of the FILE (8) name, as the translation software uses this name with various extensions. A FILE (8) name must also be a legal LCA name: see "LCA Naming Conventions" below. Each functional block in an LCA design must carry one and only one FILE (8) name.

For example, if the FILE (8) label on a functional block is MUX, the lower-level drawing which describes this block must be called MUX.DWG. Similarly, the PIN2XNF translation program will expect to find a file called MUX.PIN, which is created from MUX.DWG as described later in this application note.

The name given to each pin on a functional block must be assigned one of three attributes: PINI (23) for input pins, PINO (21) for output pins, or PNBT (22) for bidirectional pins. Each pin name must match exactly the name of the corresponding signal on the lower-level drawing. See "Naming Signals and Buses" for more information on signal naming.

Using Functional Blocks for Multiple-page Drawings

A single functional block can be used to represent a module which is contained on more than one lower-level drawing page. In this case, the single FILE (8) name on the functional block represents the combined lower-level schematic, no matter how many drawing pages it contains.

If you intend to use the XMAKE program to translate your design, you must name each page of the lower-level schematic using the same base filename, with the extensions D01, D02, D03 and so on. The functional block for this schematic should carry one FILE (8) label, which matches the base filename of the lower-level drawings. Do not include an extension in the FILE (8) name. Drawing files named in this manner are recognized by XMAKE as multiple pages of a single schematic, and will be combined correctly.

For example, if a lower-level schematic consists of the drawings MUX.D01, MUX.D02 and MUX.D03, the FILE (8) label on the functional block would read "MUX."

For more information on XMAKE and the translation process, see section 1.

Naming Signals and Buses

Every signal in an LCA design, including both individual nets and buses, should be assigned a name. A name for either a net or a bus is assigned using an attribute SIG (5) label.

The only restriction on individual net names is that each must be a valid LCA name: see "LCA Naming Conventions" later in this section. If two nets on the same drawing carry an identical name, they will be considered connected by the translation software. Therefore, if nets are not intended to be connected, they must have unique names. Each distinct net must carry no more than one name. If a name is not assigned to a net, the translation software assigns an arbitrary unique name. Since these unique names are difficult to trace when debugging the design, it is highly recommended that you name every net.

Note: *The name given to IOBs in the completed LCA file is based on the name of the net connected to the I/O pad. If you do not name this net, the resulting IOB does not carry a user name, and is only identified by the actual package pin designation.*

Note: *If a hierarchical module contains multiple pages (see "Using Functional Blocks for Multiple-page Drawings" earlier in this section), nets which have identical names are connected across page boundaries.*

There are two steps to naming a bused signal:

1. Name the individual nets of the bus.
2. Name the bus itself to identify the component nets.

Each individual net in a bus is assigned a normal SIG (5) name, which must be a valid LCA name. Alternatively, an individual net may be assigned a positive integer value as a name, also carrying the SIG (5) attribute. Using consecutive integer values will make the name of the bus itself simpler, as shown below.

The name of the parent bus signal (or trunk) specifies the name of the bus itself, as well as the names of the individual nets. The parent bus name carrying the SIG (5) attribute is expressed in one of the following forms.

```
name<net1, net2, j:k, net3,...>

name[net1, net2, j:k, net3,...]
```

The name of the parent bus is represented by *name*. Any individual nets that are named with consecutive integer labels can be specified in the form *j*:*k*, where *j* and *k* are the bounds of the consecutive range. Individual nets that are not part of an integer range must be specified explicitly. These various net specifications must be separated by commas.

If a bus signal goes into a functional block using a bus pin stub, the individual net specifications for the upper-level bus signal must match exactly the individual net specifications for the bus signal on the lower-level drawing. Although the name of the bus pin must match the name of the lower-level bus signal, these names need not match the connecting upper-level bus signal. Every individual net on the upper-level bus does not have to be split out from the bus, but the translation software will issue a binding mismatch message to warn you of this condition.

As an example of bus naming, consider a bus consisting of ten individual signals. If eight of the component nets are labeled with the integers 0 through 7, and the remaining two nets are labeled A and B, the parent bus could be specified as IN<0:7, A, B>. When expanded by the translation software, the individual nets would be called IN<A>, IN<B>, IN<0>, IN<1>, IN<2> and so on.

LCA Naming Conventions

All names given to symbols and signals in an LCA design must be valid for the XACT Development System. The requirements for all LCA names are outlined below.

- User-defined LCA names may contain only the following characters:
  ```
  A-Z  a-z  0-9  $  _  -  <  >
  ```
- The characters < and > have special meaning in FutureNet and should only be used when defining bus signals.

- Names can be up to 1024 characters in length, although some translation programs may not preserve the entire name. Names must contain at least one non-numeric character, and can begin with any legal character. The case of alphabetic characters is ignored; uppercase and lowercase characters are considered identical.

Creating Symbols with MEMGEN

The Xilinx memory compiler, the MEMGEN program, creates RAM and ROM memories within Xilinx XC4000 LCAs. With MEMGEN, you can automatically create memories ranging from 1 to 32 bits wide and up to 256 words deep.

The MEMGEN program is described in the XACT Design Implementation User Guide. The example below illustrates how to use the MEMGEN program to create a memory symbol for the FutureNet schematic editor. For a full description of the memgen program, consult the Design Implementation User Guide.

To use the MEMGEN program, a description of the memory — including its type, size and contents — is placed in a Memory Definition File. In this example, MEMGEN creates the file. Run MEMGEN with the filename that is used for the memory.

```
memgen shifter
```

MEMGEN creates a Memory definition file called SHIFTER.MEM and an XNF file to represent it. To create a FutureNet symbol for this example memory function, enter the command:

```
memgen shifter -f
```

The Memory Definition File for shifter was created previously by MEMGEN. This command causes MEMGEN to read the SHIFTER.MEM definition file, from which it creates a FutureNet command file called SHIFTER.CMD. MEMGEN displays the following messages on the screen.

```
Reading your Memory Definition File called 'shifter.mem' ...

Creating a FutureNet(tm) symbol command file for the memory function
...
Symbol command file saved as 'shifter.cmd' ...

NOTE:
----
To automatically save the FutureNet memory symbol in your MEMGEN update
library, do either of the following:

   From the prompt, enter
     'fn shifter.cmd'

   Within the Xilinx Design Manager (XDM) select
     'DesignEntry',
     'Symbol'
```

The command file created by MEMGEN can then be used to create the actual RAM symbol. To use the SHIFTER.CMD file, you would run FutureNet from the DOS prompt in the following manner.

```
fn shifter.cmd
```

This runs the FutureNet schematic editor, draws the memory symbol using the instructions in the SHIFTER.CMD command file, and saves it into a FutureNet update library named MEMGEN.SYM. The last instruction in the command file exits from the FutureNet program.

The command file created by MEMGEN always stores its symbol in the library MEMGEN.SYM, which it creates if it does not already exist. You can then use the symbol by specifying the MEMGEN symbol library in FutureNet in the following manner.

```
LIB MEMGEN.SYM
```

Then load the memory symbol using the load (.L) or load and tag (*) command. Alternatively, the memory symbol can be created using the XACT Design Manager, using the Symbol function found under the FutureNet menu selection. When you select the Symbol option, the design manager will display all FutureNet command files (CMD file extension) found in the current directory. If you select one of these files, the Design Manager runs FutureNet, draws the symbol, saves it into the MEMGEN.sym update library, and then returns to the XDM screen.

Using the Xilinx Library — Xilinx Primitives and Macros

As stated at the beginning of this section, the Xilinx library contains primitives and macros, which are the building blocks of any LCA design. Primitives are basic logic elements, such as gates and flip-flops. Macros are combinations of various primitives or other macros that are used to implement common functions. They are classified as either hard or soft macros.

This section provides information on using the Xilinx library to create an LCA design. This section also describes several Xilinx symbols that have a special usage in FutureNet.

Note: *The functions of all Xilinx primitives and macros are explained in the Macro Library. FutureNet-specific information on certain symbols is described in this section.*

The Xilinx Libraries

The Xilinx library for the FutureNet Schematic Editor consists of FutureNet logic symbols and macros for schematic entry of LCA designs. It also contains translator programs that translate a FutureNet drawing file (DWG) into the Xilinx Netlist Format file (XNF).

Note: *Use only parts from the Xilinx library for creating LCA designs and user macros.*

Loading Parts from the Xilinx Libraries

To load symbols from the Xilinx symbol library into your design, use the following FutureNet commands.

Note:

- Load & Tag (*) — The Load Symbol command loads the symbol at the graphics cursor location and initiates a tag and drag.
- Load (.L) — This command works the same way as the * command, except that tag and drag is not initiated.

- Reflect (.RE) — This command reflects a symbol about its horizontal or vertical axis, creating a mirror image of the symbol.
- Rotate (.R) — The rotate command enables you to rotate the symbol in 90-degree increments. In addition, the symbol is tagged so that it can be moved using the mouse or arrow keys.

Note: *Refer to the appropriate Macro Library for more detailed information regarding gate-naming conventions.*

XC2000 Library Exceptions for Use with FutureNet

The XC2000 Family CLB and IOB Primitive

CLB Primitive

You can enter portions of a design's logic directly in terms of CLBs instead of entering the logic schematically. A CLB primitive allows you to directly specify CLB configurations. This permits precise control of logic partitioning. In addition, when using FutureNet you can only specify a CLB's FGM base configuration in a CLB primitive. The FGM BASE configuration is the same as the FG — two functions of up to three variables, but the two function outputs are multiplexed together and controlled by the B input.

Figure 1 contains the symbols for a blank CLB primitive, a configured CLB primitive, and the equivalent circuit. You must specify the BASE, CONFIG, EQUATE F, and EQUATE G commands for the CLB. These text strings in the symbol (attribute 82) are passed through PIN2XNF, XNFMAP, and MAP2LCA to XACT as configuration for the CLB. These commands are described in further detail later in this section.

Command	**Description**
BASE	Sets the CLB's base configuration (F, FG, or FGM).
CONFIG	Specifies the CLB's internal interconnections.
EQUATE F=	Configures a CLB's logic F function.
EQUATE G=	Configures a CLB's logic G function.

The CLB symbol has A, B, C, D, K, X, and Y pins that correspond to the CLB's pins. Signals connected to these pins in the schematic are connected to the corresponding CLB in the LCA design.

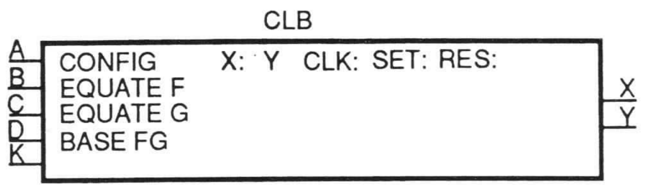
Note: *The configuration commands must be consistent with the connections. For example, if you use the A input in an equation, then connect a signal to the A pin.*

The CLB symbol contains a blank template for each of the required configuration text strings. Edit the strings to include the desired configuration commands using the -Esc key editing mode in FutureNet.

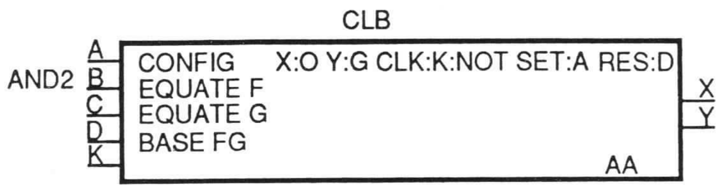
You must ensure that the configuration commands are correct. Neither PIN2XNF, XNFMAP, nor MAP2LCA checks them; they pass them to XACT. When XNFMAP reads the design file, it issues error messages if these commands contain errors. Also use the DRC command in XACT to detect errors in the CLB configuration.

You can specify (in the symbol) the location in which to place the CLB or IOB using an alphanumeric string with attribute 80: for example, the -AA string in Figure 1. The string must be a valid CLB name (for example, AA, AB and AC). If you do not specify a location, an arbitrary location is assigned. Then use APR or XACT to assign a suitable location.

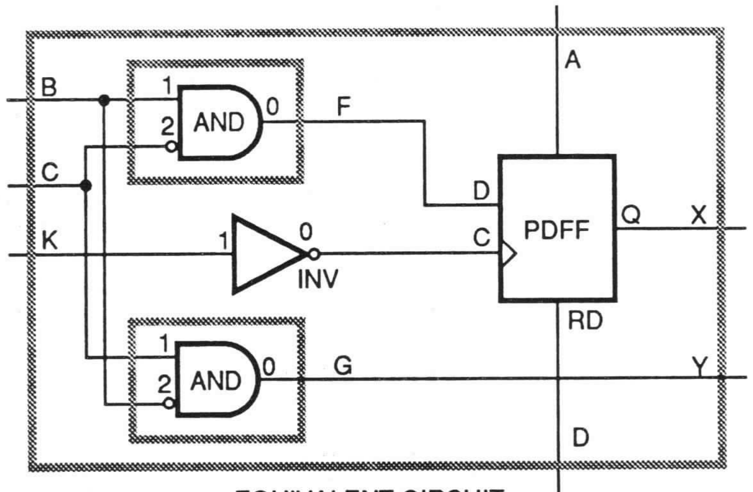
Figure 1 *XC2000 CLB Primitive and Example*



Blank 2000 CLB Primitive



2000 CLB Primitive Example



EQUIVALENT CIRCUIT

Above is a symbol for a blank XC2000 CLB primitive, a configured CLB primitive, and the equivalent circuit. You must specify the BASE, CONFIG, EQUATE F, and EQUATE G commands for the CLB. CLB primitives allow you to directly specify CLB configurations. This permits precise control of logic partitioning.

IOB Primitives

Figure 2 contains the symbols for a blank IOB primitive, a configured IOB primitive, and the equivalent circuit. You must specify the CONFIG and BASE commands for the IOB. These alpha strings in the symbol (attribute 82) are passed through PIN2XNF, XNFMAP, and MAP2LCA to XACT as configuration for the IOB. These commands are described in further detail later in this section.

Command	Description
BASE	Sets the CLB's base configuration (I/O).
CONFIG	Specifies the CLB's internal interconnections.

The CLB and IOB symbol contains a blank template for each of the required configuration text strings. Edit the strings to include the desired configuration commands using the Esc key editing mode in FutureNet.

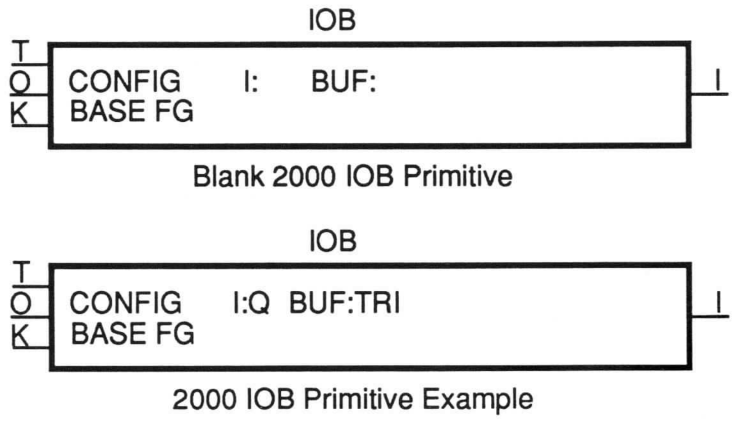
You must ensure that the configuration commands are correct. Neither PIN2XNF, XNFMAP, nor MAP2LCA checks them; they pass configuration commands on to XACT. When XACT reads the design file, it issues error messages if these commands contain errors. Use the DRC command in XACT to detect errors in the block configuration.

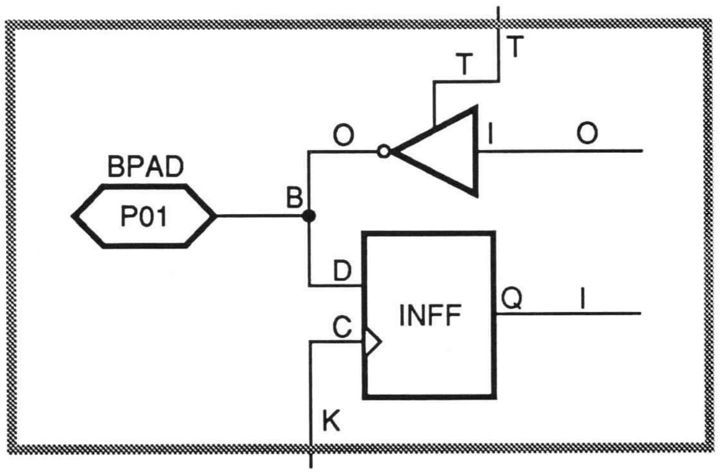
You can specify the location (in the symbol) in which to place the IOB by using an alphanumeric string with attribute 80. The string must have a valid IOB pin (for example, P12 or A13). Refer to the datasheet pin assignments for the valid IOB pin names.

For a package type other than PGA, add a P in front of the pin number. For PGAs the pin name is as listed in the datasheet (for example, J11). To use an unbonded IOB, you must run **editlca** in XACT to find the unbonded pad name (for example, U74).

Note: *IOB primitives allow you to directly specify IOB configurations.*

Figure 2 *IOB Primitive and Example*





EQUIVALENT CIRCUIT

CLB and IOB Primitive Commands

BASE [F/FG/FGM/IO]

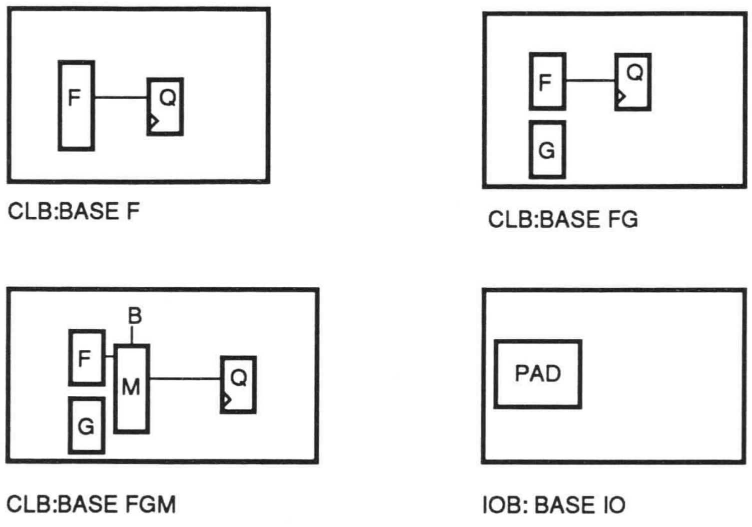
This command sets the base configuration.

Command	Description
I/O	For IOBs only.
F	One function of up to four variables.
FG	Two functions up to three variables each.
FGM	Same as FG, but the two function outputs are multiplexed together and controlled by the B input.

See Figure 3 for an illustration of these XC2000 Family CLB and IOB base configurations.

Note: *BASE F, FG and FGM are for CLBs; BASE IO is for IOBs.*

Figure 3 *XC2000 Family CLB and IOB CLB Primitives BASE Configurations*



CONFIG tag:setting

This command specifies logic element inputs and the storage element function. For XC2000 Family Designs the config tags and settings are as follows:

TAG	BASE F	BASE FG	BASE FGM
X:	F Q	F G Q	M Q
Y:	F Q	F G Q	M Q
Q:	FF LATCH	FF LATCH	FF LATCH
SET:	A F	A F	A M
RES:	D F	D G	D M
CLK:	K C F NOT	K C G NOT	K C M NOT

Note: *For BASE FGM, M=F if B=1 and M=G if B=0.*

TAG	BASE I
I:	PAD Q
BUF:	ON TRI

EQUATE (Block): Configure a CLB's Logic Function(s)

[EQUATE] tag = Boolean expression

The symbols you can use in the Boolean expression are as follows.

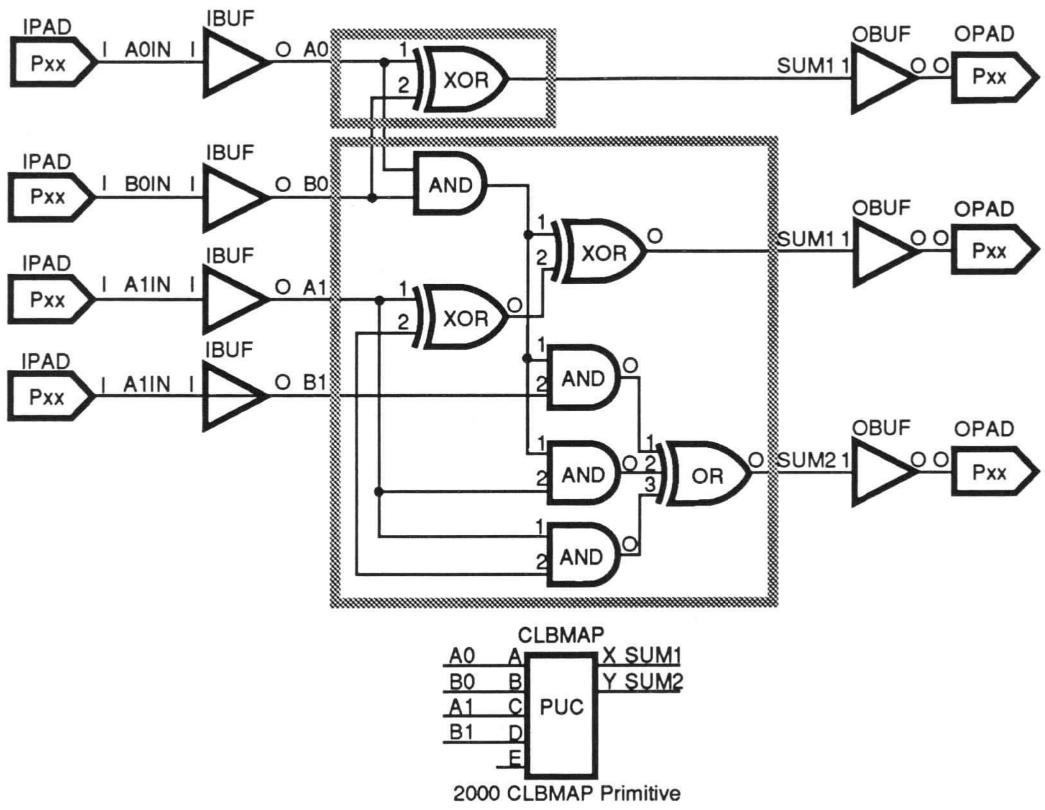
Symbol	Boolean Equivalent
~	NOT
*	AND
@	XOR
+	OR

XC2000 CLBMAP Primitive

Use CLBMAP primitives to control the partitioning of logic into a CLB. Though a CLBMAP is not logic, you use it to change the mapping of logic. To use a CLBMAP primitive, first define a logic group on the schematic that fits into one CLB. Determine the input and output signals for this logic group and assign them to the appropriate pins on the CLBMAP as shown in Figure 4. You may leave some pins unconnected; however, you must specify all signals associated with the logic group on the CLBMAP.

The PUC text string (attribute 84) inside the CLBMAP primitive stands for Pins Unlocked and Closed. This means that CLBMAP pins can be swapped with other pins (unlocked), but no more additional logic can be added to the CLB (closed). PLC (Pins Locked and Closed) is the other option; this means that the CLBMAP pins are locked onto the specified CLB pins. You can use the P (Pinlock APR Constraint Flag) in conjunction with the PUC attribute to lock only the pins which have pinlock flags attached to them.

Figure 4 *CLBMAP Primitive and Example*



Note: *CLBMAP primitives are used to control the partitioning of logic into a CLB.*

XC3000 Library Exceptions for Use with FutureNet

The XC3000 Family CLB and IOB Primitive

You can enter portions of a design's logic directly in terms of CLBs and IOBs instead of entering the logic schematically. CLB and IOB primitives allow you to directly specify CLB or IOB configurations. This permits precise control of logic partitioning. In addition, you can specify (in FutureNet only) the FGM configuration via the CLB primitive. The FGM BASE configuration is similar to the FG configuration: two functions of up to four variables, but the two function outputs are multiplexed together and controlled by the E input.

CLB Primitive

Figure 5 contains the symbols for a blank CLB primitive, a configured CLB primitive, and the equivalent circuit. You must specify the BASE, CONFIG, EQUATE F, and EQUATE G commands for the CLB. These alphanumeric strings in the symbol (attribute 82) are passed through PIN2XNF, XNFMAP, and MAP2LCA to XACT as configuration for the CLB. These commands are described in further detail later in this section.

Command	Description
BASE	Sets the CLB's base configuration (F, FG, or FGM).
CONFIG	Specifies the CLB's internal interconnections.
EQUATE F=	Configures a CLB's logic F function.
EQUATE G=	Configures a CLB's logic G function.

The CLB symbol has A, B, C, D, E, DI, EC, K, RD, X, and Y pins, which correspond to the pins of the CLB; signals connected to these pins in the schematic are connected to the corresponding CLB in the LCA design.

Note: *The configuration commands must be consistent with the connections. For example, if you use the A input in an equation, then a signal should be connected to the A pin.*

You can specify (in the symbol) the LCA location of the CLB using a text string with attribute 80 (for example, the -AA string in Figure 1-5). The string must be a valid CLB name (for example, AA, AB or AC). If you do not specify a location, an arbitrary location is assigned. Then use APR or the XACT Design Editor to assign a suitable location.

Note: *CLB primitives allow you to directly specify CLB configurations. This permits precise control of logic partitioning.*

IOB Primitives

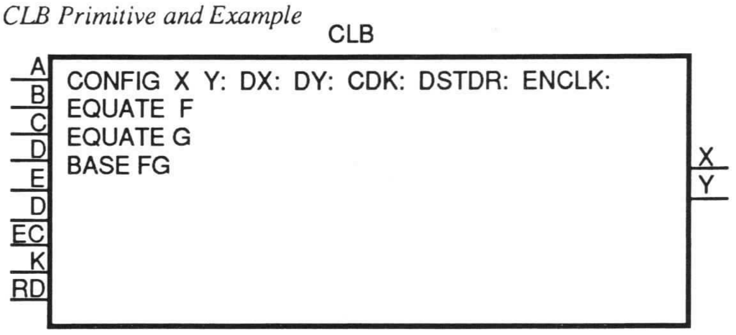
Figure 6 contains the symbol for a blank IOB primitive, a configured IOB primitive, and the equivalent circuit. You must specify the CONFIG and BASE commands for the IOB. These alphanumeric strings in the symbol (attribute 82) are passed through PIN2XNF, XNFMAP, and MAP2LCA to the LCA file as configuration for the IOB. These commands are described in further detail later in this section.

Command	Description
BASE	Sets the CLB's base configuration (I/O).
CONFIG	Specifies the CLB's internal interconnections.

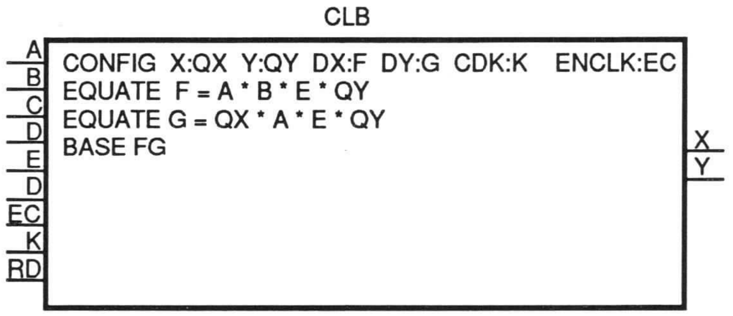
The CLB and IOB symbols contain a blank template for each of the required configuration text strings. Edit the strings to include the desired configuration commands using the Esc key editing mode in FutureNet.

You must ensure that the configuration commands are correct. Neither PIN2XNF, XNFMAP, nor MAP2LCA checks them; they pass them to the LCA file. When **editlca** within XACT reads the design file, it issues error messages if these commands contain errors. Use the DRC command in XACT to detect errors in the CLB configuration.

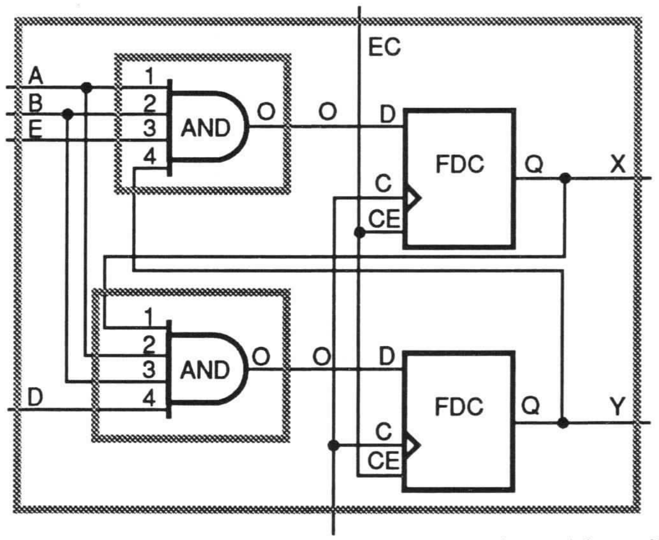
Figure 5 *XC3000 CLB Primitive and Example*



Blank 3000 CBL Primitive


CLB
A
B
C
D
E
D
EC
K
RD
CONFIG X:QX Y:QY DX:F DY:G CDK:K ENCLK:EC
EQUATE F = A * B * E * QY
EQUATE G = QX * A * E * QY
BASE FG
X
Y

3000 CBL Primitive Example

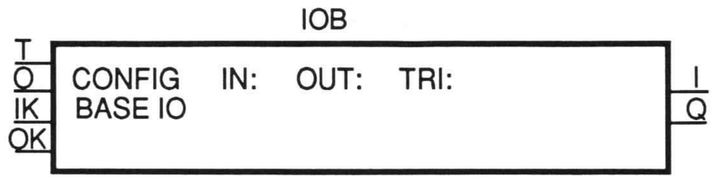


You can specify (in the symbol) the IOB's location using a text string with attribute 80. The string must have a valid IOB pin (for example, P12 or A13). Refer to the datasheet pin assignments for the valid IOB pin names.

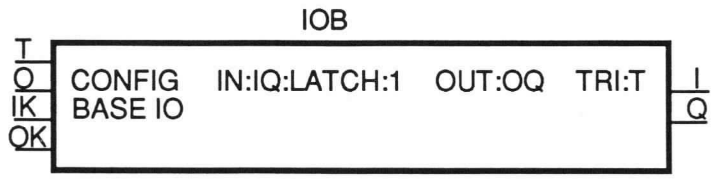
Note: *For package types other than PGA, add a P to the front of the pin number. For PGAs the pin name is as listed in the datasheet. To use an unbonded IOB, use the unbonded pad name used by XACT (such as U74).*

Note: *IOB primitives allow you to directly specify IOB configurations.*

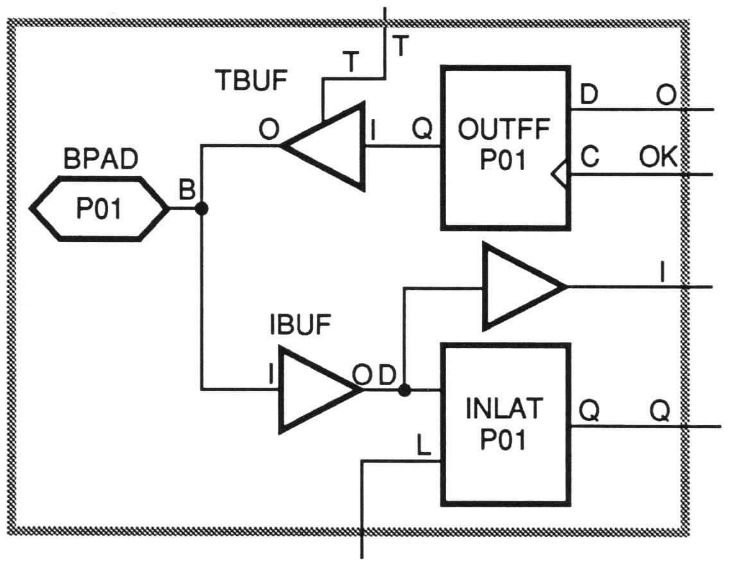
Figure 6 *XC3000 IOB Primitive and Example*



Blank 3000 IOB Primitive



3000 IOB Primitive Example



EQUIVALENT CIRCUIT

CLB and IOB Primitive Commands

BASE [F/FG/FGM/IO]

BASE [F/FG/FGM/IO]

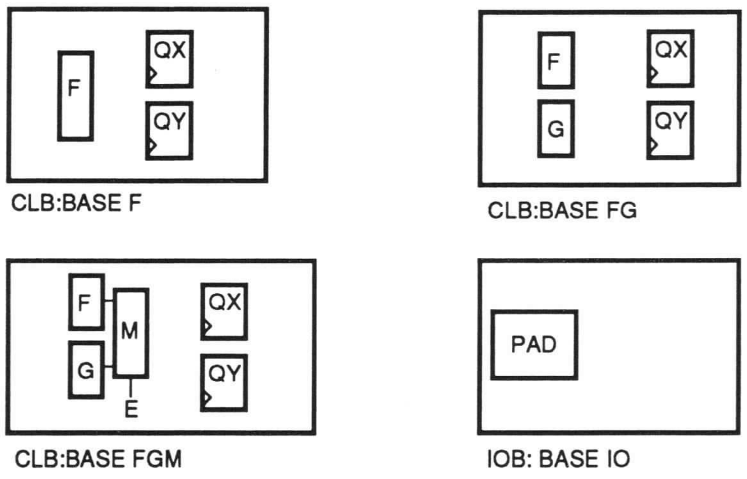
This command sets the base configuration.

Command	**Description**
I/O	For IOBs only.
F	One function of up to five variables.
FG	Two functions up to four variables each.
FGM	Same as FG, but the two function outputs are multiplexed together and controlled by the B input.

See Figure 7 for an illustration of these XC3000 Family CLB and IOB base configurations.

Note: *BASE F, FG and FGM are for CLBs; BASE IO is for IOBs.*

Figure 7 *XC3000 Family CLB and IOB Primitive BASE Configurations*



CONFIG tag:setting

CONFIG tag:setting

This command specifies logic element inputs and the storage element function. For XC3000 Family Designs the config tags and settings are as follows:

TAG	BASE F	BASE FG	BASE FGM
X:	F QX	F QX	M QX
Y:	F QY	G QY	M QY
DX:	DI F	DI F G	DI M
DY:	DI F	DI F G	DI M
CLK:	K NOT	K NOT	K NOT
RSTDIR:	RO	RO	RO
ENCLK:	EC	EC	EC

TAG	BASE I
IN:	I IQ IKNOT FF LATCH PULLUP
OUT:	O OQ NOT OKNOT FAST
TRI:	T NOT

Note: *For BASE FGM, M=F if E=0 and M=G if E=1.*

TAG	BASE I
I:	PAD Q
BUF:	ON TRI

EQUATE (Block): Configure a CLB's Logic Function(s)

[EQUATE] tag = Boolean expression

The symbols you can use in the Boolean expression are as follows.

Symbol	Boolean Equivalent
~	NOT
*	AND
@	XOR
+	OR

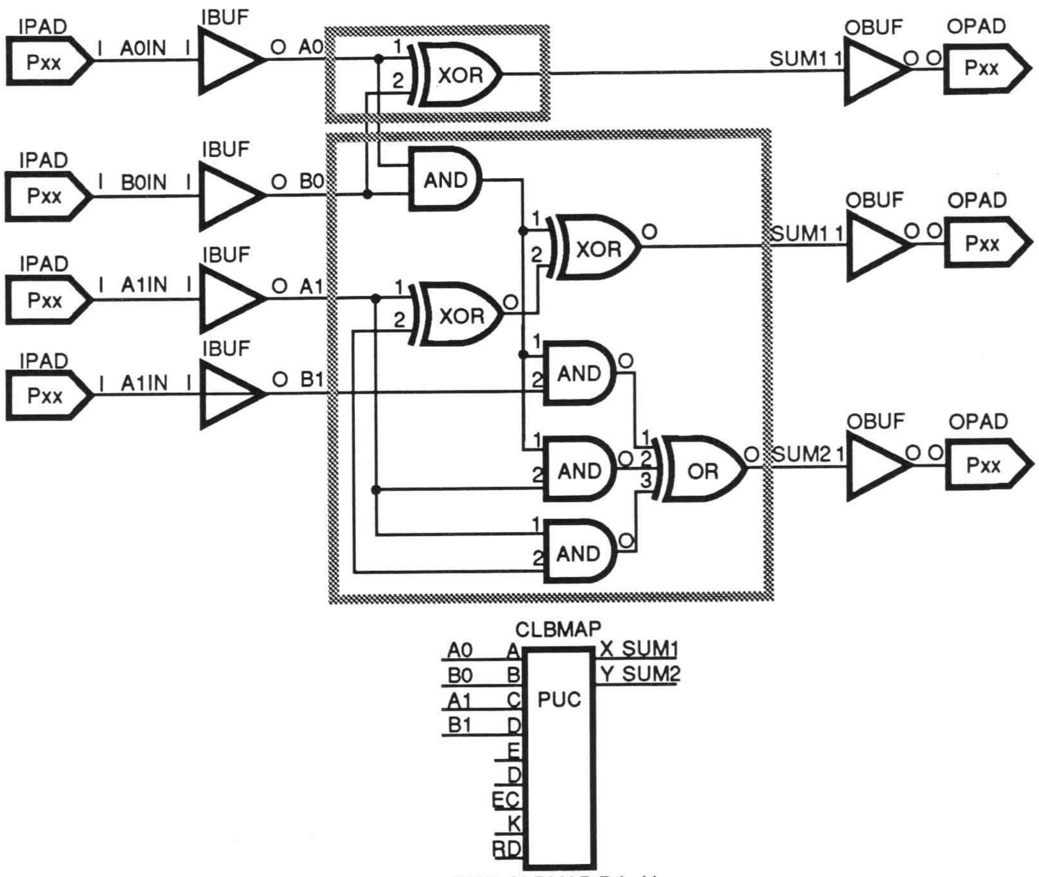
XC3000 CLBMAP Primitive

Use CLBMAP primitives to control the partitioning of logic into a CLB. Though the CLBMAP is not logic, you can use it to change the mapping of logic. To use a CLBMAP primitive, first define a logic group on the schematic that fits into one CLB. Determine the input and output signals for this logic group and assign them to the appropriate pins on the CLBMAP as shown in Figure 8. You can leave some pins unconnected; however, you must specify (on the CLBMAP) all signals associated with the logic group.

The PUC text string (attribute 84) inside the CLBMAP primitive stands for Pins Unlocked and Closed; this means that the CLBMAP pins can be swapped with other pins (unlocked) and no more additional logic can be added to the CLB (closed). The other option is PLC (Pins Locked and Closed), meaning that the CLBMAP pins are locked onto the specified CLB pins. You can use the P (Pinlock APR Constraint Flag) in conjunction with the PUC attribute to lock only the pins that have an attached pinlock flag.

Note: *Use CLBMAP primitives to control the partitioning of logic into a CLB.*

Figure 8 *CLBMAP Primitive and Example*



3000 CLBMAP Primitive

XC4000 Library Exceptions for Use with FutureNet

Representing Power and Ground Signals

VCC **Logical High**

The VCC symbol flag is used to tie a net to a logical HIGH state. Load the VCC flag (symbol name: VCC) and attach it to the desired net, using a junction dot if a T connection is formed. The actual VCC symbol appears as +5 on the schematic. A net tied to VCC cannot have any other source.

When the PPR program encounters a net tied to the VCC flag, it removes any logic that is disabled by the VCC signal. Note that a VCC signal will only be physically implemented in the LCA if the logic it sources is not removed by PPR.

Alternatively, a net may be tied to VCC by giving it the name VCC, using a standard SIG (5) attribute. Any net with the name VCC will be treated exactly as a net tied to a VCC flag.

GND **Ground**

The GND symbol flag is used to tie a net to a logical LOW state. Load the GND flag (symbol name: GND) and attach it to the desired net, using a junction dot if a T connection is formed. A net tied to GND cannot have any other source.

When the PPR program encounters a net tied to the GND flag, it removes any logic that is disabled by the ground signal. Note that a GND signal will only be physically implemented in the LCA if the logic it sources is not removed by PPR.

Alternatively, a net may be tied to ground by giving it the name GND, using a standard SIG (5) attribute. Any net with the name GND will be treated exactly as a net tied to a GND flag.

Representing IOBs

PAD **External LCA Pad**

The PAD primitive represents one of the external pads on the LCA that are electrically connected to the metal leads on the device package. Use this primitive to indicate a connection to an external pin. The PAD primitive must be connected to either an I/O buffer (such as IBUF or OBUF) or an I/O storage element (such as INLAT or OUTFF). The PAD has no directionality and should be used for LCA input signals, output signals, or bidirectional signals.

To assign a PAD primitive to a specific package pin, attach a label with attribute 80 to the pad symbol. As loaded from the Xilinx library, the PAD symbol has an attribute 80 label reading Pxx, that can be edited to represent an actual package pin, such as P17 or P6. For a PGA package, use the actual pin designation without the leading P character, for example A5 or F10. If the attribute 80 label is left as Pxx, the PPR program will choose an optimal package pin for that pad.

Note: *Locking pin locations restricts the ability of PPR to efficiently route your design. Xilinx recommends that you allow the PPR program to choose all pin locations.*

By default, every IOB configured as an output or bidirectional signal uses the slew-rate limited mode of the output buffer, which reduces noise generation and ground bounce. To switch an output buffer to the fast mode, attach an attribute 83 label reading FAST to the appropriate PAD symbol. The point of effect for this label should be inside the PAD symbol. The default condition is SLOW.

In keeping with the conventions of the XC2000 and XC3000 Xilinx libraries, the PAD symbol can also be loaded using the names IPAD and BPAD. The name OPAD loads a reflected version of the PAD primitive. Each of these names represent the same PAD primitive, which is stored under various names for convenience.

PADU **Unbonded Pad**

For packages where the LCA has more IOBs than can be bonded to external pins, the logic resources within each of the unbonded IOBs can still be used. This is done by using the PADU symbol to indicate an unbonded pad.

Since unbonded pads do not correspond to an external package pin, there is usually no need to assign them a specific IOB location. However, if you wish to do so, attach a label with attribute 80 to the PADU symbol. As loaded from the Xilinx library, the PADU symbol has an attribute 80 label reading Uxx that can be edited to represent an unbonded IOB location, such as U65. Use the XACT Design Editor to determine the appropriate IOB name. If the attribute 80 label is left as Uxx, the PPR program will choose an optimal unbonded IOB location.

Note: *Locking IOB locations restricts the ability of PPR to efficiently route your design. Xilinx recommends that you allow the PPR program to choose all IOB locations.*

In keeping with the conventions of the XC2000 and XC3000 Xilinx libraries, the PADU symbol can also be loaded using the name **UPAD**. Both names represent the same PADU primitive, which is stored under two names for convenience.

INFF Input Flip-Flop

The INFF symbol represents both an IBUF and a D-type, positive-edge triggered input flip-flop within a single IOB. The INFF symbol has both registered and direct outputs, representing the output of the flip-flop (Q) and the output of the IBUF (O). The D input of the INFF must be connected to a PAD or PADU symbol.

By default, the INFF is reset on powerup and when the global set/reset signal is asserted. To configure INFF to be set on powerup or on assertion of the global set/reset signal, attach an attribute 83 label reading INIT=S to the INFF symbol. The point of effect for this label should be inside the INFF symbol. The default condition is INIT=R.

By default, the input signal from the external pad passes through a delay buffer before reaching the D-input of INFF. This delay increases the setup time required for the INFF, but eliminates any hold time requirement. Removing the delay reduces setup time but increases hold time. To remove this delay buffer, attach an attribute 83 label reading NODELAY to the INFF symbol. The point of effect for this label should be inside the INFF symbol. The default condition is DELAY.

INLAT Input Latch

The INLAT symbol represents both an IBUF and a high-level transparent input latch within a single IOB. The INLAT symbol has both latched and direct outputs, representing the output of the latch (Q) and the output of the IBUF (O). The D input of the INLAT must be connected to a PAD or PADU symbol.

By default, the INLAT is reset on powerup and when the global set/reset signal is asserted. To configure INLAT to be set on powerup or on assertion of the global set/reset signal, attach an attribute 83 label reading INIT=S to the INLAT symbol. The point of effect for this label should be inside the INLAT symbol. The default condition is INIT=R.

By default, the input signal from the external pad passes through a delay buffer before reaching the D-input of INLAT. This delay increases the setup time required for the INLAT, but eliminates any hold time requirement. Removing the delay reduces setup time but increases hold time. To remove this delay buffer, attach an attribute 83 label reading NODELAY to the INLAT symbol. The point of effect for this label should be inside the INLAT symbol. The default condition is DELAY.

INREG **Input Latch and Flip-Flop**

The INREG symbol represents both a D-type, positive-edge triggered input flip-flop and a low-level transparent input latch, contained in a single IOB. The output of the latch appears at the QM pin and the output of the flip-flop appears at Q. The D input of the INLAT must be connected to a PAD or PADU symbol.

By default, the INREG is reset on powerup and when the global set/reset signal is asserted. To configure INREG to be set on powerup or on assertion of the global set/reset signal, attach an attribute 83 label reading INIT=S to the INREG symbol. The point of effect for this label should be inside the INREG symbol. The default condition is INIT=R.

By default, the input signal from the external pad passes through a delay buffer before reaching the D-input of INREG. This delay increases the setup time required for the INREG, but eliminates any hold time requirement. Removing the delay reduces setup time but increases hold time. To remove this delay buffer, attach an attribute 83 label reading NODELAY to the INREG symbol. The point of effect for this label should be inside the INREG symbol. The default condition is DELAY.

OUTFF **Output Flip-Flop**

The OUTFF symbol represents a D-type, positive edge-triggered output flip-flop in an IOB. The Q output of OUTFF does not require a separate OBUF symbol, and must be connected directly to a PAD or PADU symbol.

By default, the OUTFF is reset on powerup and when the global set/reset signal is asserted. To configure OUTFF to be set on powerup or on assertion of the global set/reset signal, attach an attribute 83 label reading INIT=S to the OUTFF symbol. The point of effect for this label should be inside the OUTFF symbol. The default condition is INIT=R.

OUTFFT **Output Flip-Flop with Three-State Output Buffer**

The OUTFFT symbol represents both a D-type, positive-edge triggered output flip-flop and a three-state output buffer, within a single IOB. The O output of OUTFFT does not require a separate OBUF symbol, and must be connected directly to a PAD or PADU symbol. The T pin that controls the IOB three-state buffer uses the same polarity as the TBUF symbol enable LOW and three-state HIGH. This polarity can be reversed inside the IOB by placing an inverter in front of the T pin.

By default, the OUTFFT is reset on powerup and when the global set/reset signal is asserted. To configure OUTFFT to be set on powerup or on assertion of the global set/reset signal, attach an attribute 83 label reading INIT=S to the OUTFFT symbol. The point of effect for this label should be inside the OUTFFT symbol. The default condition is INIT=R.

Using Constraint Flags

A constraint flag is used to identify the timing requirements of the attached net. Every net carries a routing priority or -net weight from 0 to 100. If no specific net weight is assigned using a constraint flag, for example, the default weight of 1 (one) is assumed.

C **Critical**

The C flag is used to identify a net as critical. A C flag assigns a weight of 100 to the attached net, which gives it the highest routing priority. Load the C flag (symbol name: C) and attach it to the desired net, using a junction dot at the T connection.

N **Non-Critical**

The N flag is used to identify a net as noncritical. An N flag assigns a weight of 0 to the attached net, which gives it the lowest routing priority. Load the N flag (symbol name: N) and attach it to the desired net, using a junction dot at the T connection.

SC **Skew-Critical**

The SC flag is used to identify nets that are skew-critical. A net that is skew-critical is routed by PPR to minimize differences between load delays. Load the SC flag (symbol name: SC) and attach it to the desired net, using a junction dot at the T connection. The unspecified field in the SC=xx label has no significance and need not be altered.

W **Weight (Relative Routing Priority)**

The W flag is used to assign a relative routing priority to a net. Load the W flag (symbol name: W) and attach it to the desired net, using a junction dot at the T connection. The W=xx label (attribute 3) must be edited to specify an integer net weight from 1 to 99. For example, editing the label to read W=34 will assign the attached net a weight of 34.

X **External**

The X flag is used to identify a net as external. An external net is one that exists at a CLB output, and will not be absorbed into a CLB. For example, an external net between a logic gate and a flip-flop will force PPR to place the combinatorial logic and the flip-flop in different CLBs. This may make the partitioning of the design less efficient, but will guarantee that the external net exists at a CLB output. Load the X flag (symbol name: X) and attach it to the desired net, using a junction dot at the T connection.

Controlling Block Placement — LOC= and LOC<> Constraints

The following statements are used to specify locations (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.

CLB Placement Examples — Primitives and Soft Macros

Soft macros and flip-flops can be assigned to a single CLB location, a list of CLB locations, or a rectangular block of CLB locations. The exact function generator or flip-flop within a CLB can also be specified. CLB locations are identified as CLB_RC. The upper left CLB is CLB_R1C1.

The following examples illustrate the format of CLB constraints. To attach a LOC constraint to a logic element, create an attribute 83 label with the desired LOC= or LOC<> statement. The point of effect for this label must be on the target symbol. If the target symbol represents a soft macro, the LOC constraint is applied to all flip-flops contained in that macro. If the indicated logic does not fit into the specified block(s), the constraint is ignored.

LOC=CLB_R1C1

Place logic in CLB_R1C1.

LOC<>CLB_R1C1

Do not place logic in CLB_R1C1.

LOC=CLB_R*C1

Place logic within the first column of CLBs. The asterisk (*) is a wildcard character.

LOC=CLB_R1C1;LOC=CLB_R1C2;LOC=CLB_R1C3

Place logic in CLB_R1C1, CLB_R1C2, and/or CLB_R1C3. There is no significance to the order of the LOC= statements.

LOC=CLB_R1C1;CLB_R8C5

Place logic within the rectangular block defined by CLB_R1C1 in the upper left corner and CLB_R8C5 in the lower right corner.

LOC=CLB_R2C2.FFX

Place logic in the X flip-flop of CLB_R2C2. For the Y flip-flop, use the tag FFY.

Note: *Any constraint of the form LOC=[blocks] can be replaced by an attribute 80 label that omits the LOC= prefix. For example, an attribute 80 label reading CLB_R1C1 is equivalent to an attribute 83 label reading LOC=CLB_R1C1. A LOC<> statement or a statement that includes multiple LOC= parameters (separated by semicolons) cannot be replaced by an attribute 80 label.*

IOB Placement Examples

I/O pads, buffers, and registers can be assigned to an individual IOB location, or to a specified die edge or half-edge. IOB locations are identified by the corresponding package pin designation.

The following examples illustrate the format of IOB constraints. To attach a LOC constraint to a I/O element, create an attribute 83 label with the desired LOC= or LOC<> statement. The point of effect for this label must be on the target symbol. If the target symbol represents a soft macro containing only I/O elements for example, INFF8 the LOC constraint is applied to all I/O elements contained in that macro. If the indicated I/O elements do not fit into the specified location(s), the constraint is ignored.

LOC=P13

Place I/O element in location P13. For PGA packages, the letternumber designation is used, for example B3.

LOC=T

Place I/O element(s) in IOBs along the top edge of the die. For the other three die edges, use B (bottom), L (left), or R (right).

LOC=LT

Place I/O element(s) in IOBs along the top half of the left edge of the die. The first letter in this code represents the die edge, and the second letter represents the desired half of that edge. Other legal half-edge values are LB, RT, RB, TL, TR, BL and BR.

LOC<>L

Do not place I/O element(s) on the left edge of the die.

TBUF Placement Examples

Internal three-state buffers (TBUFs) can be assigned to an individual TBUF location, a list of TBUF locations, or a rectangular block of TBUF locations. TBUF locations are identified by the adjacent CLB. Thus TBUF_R1C1.1 is just above CLB_R1C1, and TBUF_R1C1.2 is just below.

The following examples illustrate the format of TBUF constraints. To attach a LOC constraint to a TBUF, create an attribute 83 label with the desired LOC= or LOC<> statement. The point of effect for this label must be on the target TBUF.

LOC=TBUF_R1C1.1

Place TBUF in TBUF_R1C1.1.

LOC=TBUF_R*C1

Place TBUFs any location in the first column of TBUFs. The asterisk (*) is a wildcard character.

LOC=TBUF_R1C1;TBUF_R2C8

Place TBUFs within the rectangular block defined by TBUF_R1C1 in the upper left corner and TBUF_R2C8 in the lower right corner.

LOC<>TBUF_R1C*

Do not place TBUF in any location in the first row of TBUFs.

Decode Logic Placement Examples

Decode logic can be assigned to individual decoder locations, or to a specified die edge or half-edge. All elements of a single decode function must lie along the same edge. Edge decoders are identified as DEC_RC. The left edge is column zero and the top edge is row zero. Thus the three decoders at the top of the left edge are DEC_R1C0.1, DEC_R1C0.2 and DEC_R1C0.3. Each of these decoders is capable of implementing a single-input wired-AND function.

The following examples illustrate the format of decode constraints. To attach a LOC constraint to a decode logic symbol, create an attribute 83 label with the desired LOC= or LOC<> statement. The point of effect for this label must be on the target symbol. If the target symbol represents a soft macro containing only decode logic (for example, DECODE8), the LOC constraint is applied to all decode logic contained in that macro. If the indicated decode logic does not fit into the specified decoder(s), the constraint is ignored.

LOC=L

Place decoder logic along the left edge of the die. For the other three edges, use T (top), B (bottom), or R (right).

Global Buffer Placement Examples

Global buffers (BUFGP and BUFGS) can be assigned to one of the four corners of the die. The following example illustrates the format of global buffer constraints. To attach a LOC constraint to a BUFGP or BUFGS symbol, create an attribute 83 label with the desired LOC= or LOC<> statement. The point of effect for this label must be on the target global buffer symbol.

LOC=TL

Place global buffer in the top left corner of the die. For the other three corners, use TR (top right), BL (bottom left), and BR (bottom right).

BLKNM= Assignments

The BLKNM= parameter is used to assign LCA block names to CLB and IOB primitives, and to basic logic elements (such as gates or flip-flops).

To assign an LCA block name to a CLB or IOB primitives, attach an attribute 83 label reading BLKNM=<name> to the primitive symbol. The point of effect for this label must be inside the CLB or IOB symbol. The block name must obey LCA naming conventions.

To assign an LCA block name to logic gates or flip-flops, attach an attribute 83 label reading BLKNM=<name> to the desired symbol. The point of effect for this label must be inside the symbol. The block name must obey LCA naming conventions. If multiple symbols carry the same BLKNM= parameter, the PPR program attempts to partition these logic elements into the same block.

Using XC4000-specific Features

PULLUP

There are two uses of the PULLUP symbol. One use is to activate the internal IOB pullup resistor. For more information on IOB pullups and pulldowns, see the Macro Libraries entries on PULLUP and PULLDOWN.

The other use of the PULLUP symbol involves the three-state buffers, the wired-AND symbols and the decode logic. Every TBUF, WAND and DECODE symbol has an open-drain output that has only two possible states: logical LOW and three-state. In order for these outputs to create a logical HIGH, one or two pullup resistors must be attached to the output net.

To indicate a single pullup, connect the PULLUP symbol to the output net. To indicate a double pullup, connect the PULLUP symbol to the output net, and attach an attribute 83 label reading DOUBLE to this symbol. The point of effect for this label must be inside the PULLUP symbol. Note that two pullups will result in a faster rise time but slightly increased power dissipation, as compared to single pullup.

RAM16X1 and RAM32X1

The RAM16X1 symbol represents a 16-word by 1-bit static RAM. It has one data input (D), 4 address inputs (A0 through A3), and one data output (O). If the writeenable (WE) is LOW, the contents of the memory at the specified address appears at the data output. If write-enable (WE) is HIGH, the data on the D input is loaded into the specified address. When write-enable (WE) is HIGH, the data on the D input also appears at the data output.

The RAM32X1 symbol represents a 32-word by 1-bit version of the static RAM described above.

If desired, the contents of RAM16X1 or RAM32X1 can be assigned an initial value that is loaded into the RAM on configuration. To assign an initial value to one of these RAM symbols, attach an attribute 83 label reading INIT=<value> to the symbol. The point of effect for this label must be inside the RAM symbol. The <value> should consist of 4 (16x1 RAM) or 8 (32x1 RAM) hexadecimal digits, which are written into the RAM from the highest address (A<3:0>=1111) to the lowest address (A<3:0>=0000). For example, if a RAM16X1 carries the label INIT=10A7, the data is written as shown here.

```
0001 0000 1010 0111
|         |
MSB of RAM  LSB of RAM
```

If no INIT= parameter is specified, the RAM is initialized with zeros on configuration.

ROM16X1 and ROM32X1

The ROM16X1 symbol represents a 16-word by 1-bit ROM, having 4 address inputs (A0 through A3) and one data output (O). The ROM32X1 symbol represents a 32-word by 1-bit ROM, having 5 address inputs (A0 through A4) and one data output (O).

The contents of ROM16X1 or ROM32X1 are loaded into the ROM at configuration. To define the contents of a ROM, attach an attribute 83 label reading INIT=<value> to the symbol. The point of effect for this label must be inside the ROM symbol. The <value> should consist of 4 (16x1) or 8 (32x1) hexadecimal digits, which are stored in the ROM from the highest address (A<3:0>=1111) to the lowest address (A<3:0>=0000). For example, if a ROM16X1 carries the label INIT=9B3E, the data is written as shown here.

```
1001 1011 0011 1110
|         |
MSB of ROM  LSB of ROM
```

WAND1

The WAND1 symbol represents a single-input wired-AND gate that uses internal open-drain three-state buffers to perform a wired-AND function. The WAND1 symbol requires that either one or two internal pullup resistors be attached to its output for a logical HIGH state to be generated. For more information on attaching pullups, see the description of the PULLUP symbol in this section.

By default, a wired-AND function is implemented using horizontal longlines. To assign a wired-AND function to the dedicated decode logic on the edges of the device, attach an attribute 83 label reading DECODE to the WAND1 symbol. The point of effect for this label must be inside the symbol.

Using XC3000 Drawings for XC4000 Designs

If you wish to use existing XC3000 design drawn in FutureNet as the basis for a new XC4000 design, you should take into consideration the following.

Using XC3000 Symbols on an XC4000 Drawing

In moving from the XC3000 library to the XC4000 version, a few symbols have changed names, a few infrequently-used soft macros have been removed, and many new parts have been added to the library. With a few exceptions, the PIN2XNF translation program converts XC3000 symbols into the equivalent XC4000 versions. Those few soft macros that have been removed from the library are still supported by PIN2XNF, so that a drawing that uses an obsolete symbol is processed correctly. The symbols that cannot be converted are listed in the following section.

If you are adding logic to an existing XC3000 design, it is necessary to load the design in FutureNet with the XC4000 libraries installed, so that the new features of the XC4000 D such as RAMs and wide decoders can be specifed on the schematic. When an XC3000 design is loaded into FutureNet with the XC4000 libraries installed, all XC3000 symbols will appear on the drawing as they have always appeared.

In the XC4000 library, the some symbols were reduced in size to allow you to fit more logic on a single drawing. As a result, the XC4000 version of a symbol may not fit exactly in the space left by the XC3000 version. Since the PIN2XNF program correctly translates most XC3000 symbols, the existing symbols need not be replaced unless a symbol unique to the XC4000 library is desired.

If an existing XC3000 symbol is mistakenly deleted, it can be recalled from the XC3000 library even though FutureNet has the XC4000 library installed. This is accomplished with the normal FutureNet commands for loading a symbol (.L and *). These two commands can optionally specify an uninstalled library to load the symbol from. The format of the complete command is as follows.

```
.L symbol_name, library_path
```

or

```
* symbol_name, library_path
```

Replace *symbol_name* with the name of the symbol to load from the XC3000 library. Replace *library_path* with the complete path name for the XC3000 symbol library, which is called X3000.SYM and resides in the \XACT directory. As an example, if the XACT software is installed on drive C, the symbol OPAD would loaded from the XC3000 library with the following command.

```
.L OPAD, C:\XACT\X3000
```

Note: *If an XC3000 design is to be implemented in an XC4000 device without adding any XC4000-specific features D for example, memory elements D the original XNF file can be upgraded to an XC4000 device using the XNFUPD program. See the XNFUPD section in the XC4000 Design Implementation Reference Guide for more information on this process.*

Exceptions to The XC3000 Symbol Conversion Process

There are five symbols in the XC3000 library that PIN2XNF cannot convert to an XC4000 device. These are CLB, CLBMAP, IOB, OSC and GXTL. Since each of these symbols depends on the architecture of the XC3000 family, they must be removed from the schematic. The XC4000 library contains CLB and IOB primitives specifically for the XC4000 family. There is no CLBMAP in the XC4000 library, although LOC= constraints can be used to control partitioning. The XC4000 family does not have the on-chip crystal oscillator represented by OSC and GXTL, but the internal configuration oscillator can be accessed during operation by using the OSC4 symbol.

There are nine symbols in the XC3000 library that combine an I/O pad with a I/O buffer or register. These symbols (D PBUF, PDFF, PIN, PINQ, PIO, PIOQ, POUT, POUTZ and PREG D) are correctly translated for an XC4000 design, although the symbols do not exist in the XC4000 library. However, since these symbols do not allow you to name the net between the pad and the buffer, the resulting package pin (and IOB) will not have a meaningful name. The preferred method is to replace each of these symbols with separate pad and buffer symbols. The XC4000 library includes many I/O macros for this purpose.

If you do encounter problems with PIN2XNF when processing an XC3000 design for an XC4000 part, consult the troubleshooting guide in section 1.

Note: *The XC3000 TTL macros D contained in the former DS311 Optional TTL Library for FutureNet D are not supported in the XC4000 design implementation process and must be replaced. Most of these TTL macros are replaced by the XC4000 macros carrying the X74 prefix. The XC4000 TTL macros have been updated to be more consistent, and thus may not always be identical to the XC3000 versions.*

Section 2: Creating an XNF File from Your FutureNet Schematic

Contents

XNF Translation

Every schematic, memory compiler, state machine, Boolean equation, or other design entry format file must be converted to a Xilinx Netlist Format (XNF) file before it can be translated into an LCA file or a configuration bitstream file (BIT). An XNF file is a text file that describes each logic gate in a design, its associated pins, and the connections between the gates. With the XACT Design Manager, you can translate your design either automatically or manually.

- **Automatic Translation** — The XMAKE program (found in the Design Manager Translate menu) automatically translates schematics into an LCA and a BIT file. All of the programs needed to generate a completed design are run in the proper sequence by XMAKE. The programs used by XMAKE include the Schematic-to-XNF translators: DCM, PINC, and PIN2XNF; and the automatic design implementation tools: either PPR (for XC4000 designs) or APR (for XC2000/3000 designs). In addition, the MAKEBITS program generates a bitstream file if PPR or APR successfully implements an XC4000 design.
- **Manual Translation** — Translating schematics into an LCA file and then a BIT file is a fairly simple process. The design must first be converted to an XNF file and then translated into an LCA file. Finally, MAKEBITS must be run to generate a BIT file that can be downloaded to an LCA device. If you are familiar with the steps involved in converting your design to an LCA file, you may want to translate the schematics manually.

Automatic Translation Using XMAKE

Automatic translation of design files into XNF files is supported by the XACT Design Manager's XMAKE program. XMAKE (using the DCM, PINC, or PIN2XNF programs, and the appropriate design implementation tools in succession) automatically converts a design file into an LCA and a BIT file. XMAKE can use schematic files (*filename*.DWG), MAK files (*filename*.MAK), XNF files (*filename*.XNF), or a combination of all three as its input file format.

Manual Translation

The steps performed automatically by XMAKE can be executed manually. These steps include running the DCM, PINC, and PIN2XNF programs in sequence on your design file to create an XNF file.

- **DCM** — The Drawing Connectivity Model program is the FutureNet drawing pre-processor that establishes connection data for a FutureNet drawing. When translating a design manually, generate a DCM file (*filename*.DCM) for each drawing file (*filename*.DWG) in the design. Do not run DCM on Xilinx macro files, as they have already been processed.
- **PINC** — PINC is the FutureNet pin list generation program that creates a PIN file (*filename*.PIN) for each input DCM file. The PIN file contains a listing of the pins on all of the symbols in the drawing and the names of the signals connected to these pins. When translating a design manually, generate a PIN file for each DCM file you create. As with DCM, don't run PINC on Xilinx macro files.

- **PIN2XNF** — The PIN2XNF program converts a FutureNet PIN file into an XNF file (*filename*. XNF). After creating PIN files from each drawing file in your design, run PIN2XNF only on the PIN file corresponding to your top-level drawing file. The PIN2XNF program determines the hierarchical structure of your design. As a result, any PIN files representing drawing files beneath the top-level design file, including Xilinx macros, are also incorporated into the XNF output file.

Note: *Once XNF files are generated, they may be processed using ADI for XC2000 and XC3000 designs or using PPR for XC4000 designs.*

Note: *Design Update — In addition to supporting new designs for XC4000 LCAs, the XACT Development System supports an upgrade path that allows you to convert existing XC2000 and XC3000 designs. The XNFUPD program (XNF Update) allows you to upgrade XC2000 and XC3000 XNF design files to be compatible with XC4000 devices, provided they do not contain any XC2000 or XC3000 architecture-specific symbols. More information on XNFUPD can be found later on in this section and in the Xilinx Design Implementation Reference Manual.*

Using the XMAKE Program

If you are using XACT Design Manager-supported design entry software, the XACT Design Manager XMAKE program automatically runs the translation programs needed to convert a design into an LCA file. Even designs containing both schematic and non-schematic modules, such as memory modules and Boolean equations, can be automatically translated.

Running XMAKE

You can run XMAKE using the keyboard or mouse to select XMAKE and any options from the Design Manager Translate menu. Presented here are steps for running XMAKE using each of these methods.

Using Your Mouse

The procedure for using the XACT Design Manager menus to run XMAKE is outlined below.

1. Select XMAKE from the Translate menu.
2. Select any desired XMAKE options from the displayed menu.
3. Select DONE when you have selected any desired options.
4. Select the top-level schematic (*toplevel*.DWG) or MAK file from the displayed menu.

Using the Keyboard

The syntax for the XMAKE program, when run from the keyboard at the XACT Design Manager command prompt, is illustrated here.

```
Cmd:XMAKE -[options] toplevel.DWG
```

where *toplevel* is a FutureNet drawing file, or

```
Cmd:XMAKE -[options] toplevel.MAK
```

where *toplevel* is a "make" file.

XMAKE automatically runs the appropriate netlist translators to convert the top-level design into an LCA file. During the process, it scans each XNF file created to see if there are any subdesigns (schematic or XNF) that also need to be processed, and does so if required.

When run on a design file, XMAKE saves the translation commands into a file called *toplevel*.MAK.

Accepted File Formats

XMAKE can use two types of input files, and it can produce as many as five types of output files. These input and output file formats are described in detail in the two following sections.

Input Files

When translating a design from FutureNet, XMAKE requires one of two possible file formats as input. These include the FutureNet DWG file or an XMAKE-generated .MAK file.

toplevel.DWG	XMAKE accepts as its input the schematic design files created by the FutureNet schematic editor.
toplevel.MAK	As XMAKE runs, it uses, and records in the MAK file, the program options currently saved in the XACT Design Manager profile (XDM.PRO). If you use a MAK file as an input file, XMAKE uses the MAK file instead of creating a new one.

Output Files

The XMAKE program creates a number of output files. XMAKE produces an LCA design file and an output file (*toplevel*.OUT) containing the output from all programs run by XMAKE. When using a design file as input to XMAKE, a MAK file is created.

toplevel.XNF	For XC2000 and XC3000 designs only, XMAKE produces a completely flattened XNF file for the entire design. The file will be named *toplevel*.XNF
toplevel.OUT	As XMAKE runs various translation programs, it redirects their output to the file *toplevel*.OUT. This is an ASCII text file containing all the screen output from the programs run by XMAKE.

Note: *Since the OUT file contains all warning or error messages that occur during design processing, you should always review the OUT file after XMAKE has run to ensure your design is error-free.*

toplevel.LCA	XMAKE creates an LCA file that is partitioned, placed, and routed by either the APR (XC2000/3000 designs) or PPR (XC4000 designs) program, unless it is disabled with the XMAKE -N option.
toplevel.BIT	For all XC4000 designs that PPR successfully routes, or XC2000 or XC3000 designs that APR successfully routes, XMAKE creates a bitstream that can be downloaded to an LCA. The configuration options for the bitstream generator are determined by the options set in the XACT Design Manager profile (XDM.PRO).

toplevel.MAK	When you run XMAKE on a FutureNet drawing file, XMAKE creates a text make file (*toplevel*.MAK) that documents how each design submodule is processed, including the options used by the translation programs. The information in the MAK file serves three purposes.

- It is used as a script for XMAKE when first translating the design.
- It documents the commands used to translate the design into an LCA file. By examining the MAK file, you can determine exactly which programs and options XMAKE ran on each design submodule.
- It can be used as an input file the next time XMAKE is run on the same design. If you want to change a few program options used during translation, you can edit the MAK file to specify desired options and rerun XMAKE with your options instead of the default options XMAKE used originally.

Default XMAKE Operation Using FutureNet

The following information describes how to use XMAKE in the initial translation process and to reprocess your design after minor changes have been made.

Processing the Design the First Time

The first time you run XMAKE on your design, specify the design file, *toplevel*.DWG.

Run the XMAKE program from the Translate menu and specify the top-level design file. For example, for a top-level design called MYDESIGN.DWG, do the following:

1. Select XMAKE from the Translate menu.
2. Select DONE to indicate you don't want to change any XMAKE options.
3. Select MYDESIGN.DWG from the menu of designs that XMAKE displays.

To perform the same steps from the keyboard, enter the following on the Design Manager command line.

```
Cmd:XMAKE MYDESIGN.DWG
```

XMAKE then performs the following operations.

1. Scans the hierarchy of the design.
2. Creates a MAK file that describes which programs and options are to be used to process the design.
3. Translates the design to an XNF and an LCA file.

The design is then ready for partitioning, placement, and routing with PPR for XC4000 designs or with ADI for XC2000 and XC3000 designs.

Reprocessing the Design After Minor Changes

To reprocess the design after making logic changes, you can run XMAKE using the MAK file that XMAKE created in step 2 above, provided the changes do not affect the hierarchy of the design.

For example, if your top-level design is called MYDESIGN.DWG, you would follow these steps.

- Select XMAKE from the Translate menu.
- Select DONE to indicate you don't want to change any XMAKE options.
- Select MYDESIGN.MAK from the menu of designs that XMAKE displays: that is, tell XMAKE to use the MAK file to decide which programs and options to use to process the design.

Note: *As long as you do not add any new hierarchical blocks to your schematics, you can use the same sequence of commands to process the design by using the original MAK file as an input to XMAKE. If you do change the hierarchy of the design, you must use the modified design file as the input to XMAKE.*

Using an existing MAK file has two benefits.

- Design processing is faster since XMAKE does not have to reprocess the entire design: it only reprocesses those files that have changed since the design was last processed.
- You can edit the MAK file to contain different program options and reprocess the design using these.

Reprocessing the Design After Major Changes

If you make significant changes to your design, such as adding additional hierarchical blocks, you must recreate the MAK file so XMAKE will find the changes to the design structure.

The easiest way to do this is to run XMAKE on the top-level design file again, just as you did the first time you processed the design. XMAKE re-scans the design, recreates the MAK file, and then processes the new or changed design portions.

XMAKE Options

Other options control how MAK files are created. They also specify options for programs run automatically by XMAKE. Except for -R, these options are only valid when you specify a design file. They do not apply if you run XMAKE with a MAK file since the MAK file already describes which programs and options to use.

-A **Map All (XC2000 and XC3000)**

When this option is selected, XMAKE maps the logic in each user-created macro before merging it into the top-level design. Logic from unrelated macros will not be combined into the same CLB when invoking XMAKE with this command.

-F **Map -FILE=" (Default) (XC2000 and XC3000)**

When the -F option is selected, XMAKE maps the logic in each macro with a -FILE=" parameter before it is merged to the top-level design. You can use this option to prevent unrelated logic from being combined in the same CLB.

-G **Generate MAK File Without Performing all Design Processing**

This option tells XMAKE to create a MAK file without executing the actual commands. This option is generally used when you want to create a custom MAK file, but want XMAKE to generate an initial script for you to edit.

-K **Use XC4000 Family Flow (Default)**

When -All" is selected as the current family, this option indicates to XMAKE that the XC4000 design flow is to be used. This option should be selected when processing a design for use with an XC4000 LCA device.

-N **Don't Run PPR (or APR)**

This option allows you to disable the automatic invocation of either PPR or APR. The default is that XMAKE will automatically run PPR to create a routed LCA file.

Note: *If you do not have the appropriate installation software, either APR or PPR, installed on your system, XMAKE will discontinue operation after completing its normal process through the generation of a top-level XNF file. As a result, no LCA file will be created.*

-O **Screen Output**

This option tells XMAKE to direct all program output to the screen instead of generating OUT files.

-R **Execute All Commands to Translate the Design**

The force re-execution option guarantees that the entire design is reprocessed, even submodules that you have not changed since the last time you processed the design. Note that there is a difference between running XMAKE with the -R option on a MAK and a schematic file, as described below.

MAK file input with -R

XMAKE performs every step in the MAK file, regardless of whether the files have been changed since the design was last processed. If you do not use the -R option, XMAKE only reprocesses those parts of the design that have been changed.

Schematic file input with -R

XMAKE recreates the MAK file and reprocesses the entire design. If you do not use the -R option, XMAKE recreates the MAK file, but only reprocesses modules that have been changed or merged.

-T **Map Top Only**

The -T option flattens your design before it is mapped into CLBs and IOBs. Additionally, the design is reduced to one XNF file that contains all the logic in user-created macros. This option allows you to have unrelated logic in the LCA file grouped together in the same CLB. While this allows efficient use of CLBs, it can make routing difficult, especially with large designs. Use the -A (Map All) option to ensure that unrelated logic is not mapped in the same CLB.

-V **Verbose Mode**

When using XMAKE in its default configuration, this option is not selected. Selecting this option causes XMAKE to display more detailed information on its progress.

-X **XNF Only (Interface to Third-Party Schematics)**

Designers using design entry tools not directly supported by XMAKE can still use XMAKE to translate their design into a LCA file by using the -X option.

Each time you reprocess your design you should

- Manually (perhaps using a batch file) translate each design submodule into an XNF file using the interface program(s) for your design entry program.
- Run XMAKE with the -X option (plus any other desired options).

To compile the design into an LCA file, XMAKE bypasses the netlist translation (since the files are already in XNF format) and runs the required design implementation programs, XNFMAP and MAP2LCA for XC2000 and XC3000 designs or PPR for XC4000 designs. For XC2000 and XC3000 designs, submodules flagged with a FILE= parameter are mapped separately and merged into the top-level design as required. For XC4000 designs, any lower-level XNF files are merged into the design before PPR begins partitioning.

Combining Multiple-Page Drawings

For XMAKE to recognize a module that consists of multiple drawings, each drawing (or "page") must have the same base *filename*, with extensions -.D01, -.D02, instead of the default DWG extension. For example, if your top-level schematic, *toplevel*, contains three pages, name these drawings *toplevel*.D01, *toplevel*.D02, and *toplevel*.D03.

If a lower-level module called MYMAC consists of two pages, the name on the upper-level symbol, specified with a FILE (8) label, should be MYMAC and the two MYMAC schematics should be named MYMAC.D01 and MYMAC.D02.

When processing the MYMAC module, XMAKE recognizes from the .D01 and .D02 extensions that MYMAC is a multiple-page schematic. XMAKE then automatically runs DCM with the following command line.

```
Cmd:DCM MYMAC.D01 MYMAC.D02
```

The resulting DCM file is called MYMAC.DCM and contains the logic from both pages. XMAKE then treats the MYMAC module exactly like a single-page module.

Using the MAK File

The XMAKE program has three basic functions.

1. It scans a design to determine its structure.
2. It creates a MAK file that describes how the design's submodules need to be processed to translate the entire design.
3. It runs the programs and options specified in the MAK file.

When you run XMAKE on a design file, for example, a DWG file for FutureNet, it performs all three functions.

However, you can use the MAK file (created in step 2 above) as input into XMAKE instead of a design file. If XMAKE is run with a MAK file as its input, XMAKE checks each submodule described in the MAK file, translating it again if required. Therefore, editing the MAK file permits you to control which programs and options XMAKE uses to process your design. While this is not usually necessary, it does give you more control over the design process when needed.

MAK File Syntax

The MAK file contains an entry for each file required to produce an LCA file for the design. Each entry lists the target file, followed by the dependent files used to create the target file. The entry also tells which programs and options to use to create the target file from its dependent files.

Note: *Entries at the top of the file are performed last, while entries at the bottom of the file are performed first. Within each entry, the programs are executed from top to bottom.*

Each target file entry has the following format.

```
target_file: dependent_file dependent_file ...
   command argument argument ...
   command argument argument ...
```

Specific syntax requirements are listed here.

- The target filename must start in the first column of the line.
- Each field (e.g., filename, command name, command line argument) must be separated from any others by at least one space or tab character.
- Command lines must start with at least one space or tab character.
- Blank lines are ignored.
- Comment lines must start with a # in the first column.

A Simple MAK File Example

The simple example below is a MAK file that tells XMAKE how to convert the FutureNet schematic SELECT.DWG into an XC4000 LCA file. When manually performing this translation, the steps are as follows.

1. Run DCM to convert the DWG schematic into a DCM file.
2. Run PINC to convert the DCM file into a FutureNet PIN file.
3. Run PIN2XNF to convert the PIN file into an XNF file.
4. Run the Xilinx program PPR to convert the XNF file into an LCA file.
5. Run Makebits to create a bitstream that can be used to configure the LCA.

The MAK file that accomplishes these steps is highlighted in the following table with explanatory comments (the comments do not appear in the actual MAK file).

Note: *In the following table, the actual processing order is from bottom to top.*

MAK File Contents	Comments

`select.bit: select.lca`
`   makebits -o select.bit select.lca`
	Create the BIT file from the LCA file using this command.

`select.lca: select.xnf`
`   ppr select.xnf`
	Create the LCA file from the XNF file using this command.

`select.xnf: select.pin`
`   pin2xnf -p 4005pc84 select.pin`
	Create the XNF file from the PIN file using this command.

`select.pin: select.dcm`
`   pinc select.dcm`
	Create the PIN file from the DCM file using this command.

`select.dcm: select.dwg`
`   dcm select.dwg`
	Create the DCM file from the DWG file using this command.

When told to make SELECT.LCA and SELECT.BIT the first time, XMAKE automatically follows these steps (only the DWG schematic file exists at first).

1. SELECT.LCA depends on SELECT.XNF, which at this point does not exist, so XMAKE searches the MAK file to see how to make SELECT.XNF.
2. SELECT.XNF depends on SELECT.PIN, which at this point does not exist, so XMAKE searches the MAK file to see how to make SELECT.PIN.
3. SELECT.PIN depends on SELECT.DCM, which at this point does not exist, so XMAKE searches the MAK file to see how to make SELECT.DCM.
4. SELECT.DCM depends on SELECT.DWG, which does exist, so XMAKE runs the DCM program to create SELECT.DCM.
5. Now that SELECT.DCM exists, XMAKE makes SELECT.PIN by running the PINC program.
6. Now that SELECT.PIN exists, XMAKE makes SELECT.XNF by running the PIN2XNF program.
7. Now that SELECT.XNF exists, XMAKE makes SELECT.LCA by running the PPR program.
8. Now that SELECT.LCA exists, XMAKE makes SELECT.BIT by running the Makebits program.

A Complete MAK File Example

An example of a complete XC4000-type MAK file is provided below for reference. The DEMO design is a XC4005PG156 design.

Note: *In the following example, a \ denotes that in the MAK file the line following it is appended onto the line containing the \.*

```
# Created by XMAKE Version 2.00 on Thu Sep 27 13:50:28 1990
# The following options were used: -F
# The following is the hierarchy of the design 'DEMO.DWG'
# DEMO.DWG
#     timer.dwg
#        timeram.xnf
#     control.dwg
#        keycoder.dwg
#     frdec.xnf
#     toneram.xnf
#     synth.dwg
#        sinrom.xnf
#        acc16h.dwg
#     dispctl.dwg
#        disprom0.xnf
#        disprom1.xnf
#        disprom2.xnf
#        disprom3.xnf
#        disprom4.xnf
demo.bit0 : demo.lca
        makebits -0 demo.bit demo.lca
demo.lca : demo.xnf
        ppr demo.xnf improvecount=1
demo.xnf : disprom4.xnf disprom3.xnf disprom2.xnf \ disprom1.xnf
disprom0.xnf
dispctl.xnf acc16h.xnf sinrom.xnf \ synth.xnf toneram.xnf frdec.xnf
keycoder.xnf control.xnf \ timeram.xnf timer.xnf demo.pin
        pin2xnf demo.pin
demo.pin : demo.dcm
        pinc demo.dcm
demo.dcm : DEMO.DWG
        dcm DEMO.DWG
timer.xnf : timer.pin
        pin2xnf -x -p4005PG156 timer.pin
timer.pin : timer.dcm
        pinc timer.dcm
timer.dcm : timer.dwg
        dcm timer.dwg
control.xnf : control.pin
        pin2xnf -x -p4005PG156 control.pin
control.pin : control.dcm
        pinc control.dcm
control.dcm : control.dwg
        dcm control.dwg
keycoder.xnf : keycoder.pin
        pin2xnf -x -p4005PG156 keycoder.pin
```

```
keycoder.pin : keycoder.dcm
        pinc keycoder.dcm
keycoder.dcm : keycoder.dwg
        dcm keycoder.dwg
synth.xnf : synth.pin
        pin2xnf -x -p4005PG156 synth.pin
synth.pin : synth.dcm
        pinc synth.dcm
synth.dcm : synth.dwg
        dcm synth.dwg
acc16h.xnf : acc16h.pin
        pin2xnf -x -p4005PG156 acc16h.pin
acc16h.pin : acc16h.dcm
        pinc acc16h.dcm
acc16h.dcm : acc16h.dwg
        dcm acc16h.dwg
dispctl.xnf : dispctl.pin
        pin2xnf -x -p4005PG156 dispctl.pin
dispctl.pin : dispctl.dcm
        pinc dispctl.dcm
dispctl.dcm : dispctl.dwg
        dcm dispctl.dwg
```

The DCM Program

The DCM program converts a FutureNet DWG file into a Drawing Connectivity Model (DCM) file used by the PINC program. You must create a separate DCM file for each hierarchical module in the design. Do not use DCM to flatten the hierarchy of a design and do not run DCM on Xilinx macros since they have already been processed.

Note: *When converting an LCA design, you must use DCM as described in this section. The instructions given in the FutureNet manuals do not necessarily apply to Xilinx designs.*

Syntax

For each module that is represented by a single DWG file (a single-page drawing), you must use the DCM program to create a separate DCM file. If the module is contained in design.DWG, run DCM as follows:

```
Cmd:DCM design.DWG
```

The .DWG extension is optional. This command creates the file *design*.DCM. Repeat this step for every single-page drawing in the hierarchy.

If a module is represented by multiple DWG files (a multiple-page drawing), you must use the DCM program to combine those drawing files into a single DCM file. If the module consists of the files *page1*.DWG, *page2*.DWG, *page3*.DWG, run DCM as follows:

```
Cmd:DCM page1 page2 page3 ...
```

The output file takes the name of the first drawing specified, that is *page1*.DCM.

If the multiple page drawing in question represents a functional block on the schematic, the name of the DCM file must match the name specified on the functional block with a FILE (8) attribute. For the page1.DCM file created above, the corresponding functional block would specify *page1* (no *filename* extension) as the FILE (8) name.

Input File [*filename*].DWG

DCM operates on the schematic design files created by the FutureNet schematic editor.

Output File [*filename*].DCM

This an intermediate step between the FutureNet drawing file and the PIN file format.

Note: *If a FutureNet drawing contains a functional block with a bus pin, the DCM program issues the warning "Bus pin terminates at symbol." This warning may be ignored.*

The PINC Program

PINC, which creates a PIN file for each input DCM file, is the FutureNet Pin List Generator. This PIN file contains a listing of the pins on all of the symbols in the drawing and the names of the signals connected to these pins. When translating a design manually, you should generate a PIN file for each DCM file you create. Don't run PINC on Xilinx macros, since they have already been processed.

Note: *When converting an LCA design, you must use PINC as described in this section. The instructions given in the FutureNet manuals do not apply to Xilinx designs.*

Syntax Once you have converted a drawing file into a FutureNet DCM file, use the PINC program to create a FutureNet pinlist file (PIN). For each DCM file in the design, run PINC in the following manner.

```
Cmd:PINC design.DCM
```

The .DCM extension is optional. This command creates a PIN file used by the PIN2XNF program.

Input File *toplevel*.DCM

PINC requires a FutureNet-generated DCM file to be used as an input file. DCM is the FutureNet drawing pre-processor that establishes connection data for a FutureNet drawing.

Output File *toplevel*.PIN

The output file generated is a PIN file that contains FutureNet pin list information required for creating an XNF file.

The PIN2XNF Program

The PIN2XNF program creates a Xilinx Netlist Format (XNF) file from one or more FutureNet PIN files. Unless specified otherwise, PIN2XNF reads any lower-level PIN files and creates a single merged XNF file. This XNF file is used by the XACT design implementation software.

Syntax

```
Cmd:PIN2XNF -[options] toplevel.PIN toplevel.XNF
```

Input File *toplevel*.PIN

The PIN file created by the FutureNet programs DCM and PINC. The PIN extension is optional.

Output File *toplevel*.XNF

The equivalent Xilinx Netlist Format (XNF) file. The XNF extension is optional. If no output filename is specified, PIN2XNF uses the input design name with the addition of the XNF extension.

Options You can modify the operation of the PIN2XNF program using the following options.

-P **Part Type**

The -P option is used to specify the LCA part type. If a part type has been specified on the FutureNet schematic (using an attribute 81 label), the -P option takes precedence. If no part type is specified either on the schematic or with the -P option, PIN2XNF attempts to determine a part type based on the symbols used. If this fails, PIN2XNF assigns a default part type.

The -P option can also be used to specify the speed grade. The XC4000 family speed grades are -10 and -7, representing CLB combinatorial block delays of 10 and 7 ns, respectively. The XC3000 and XC2000 family speed grades are -50, -70, -100, and -125, representing the flip-flop toggle frequencies in megaHertz (MHz).

Here are two examples of a 100 MHz and 10 ns speed grade specification for XC2000/3000 and XC4000 devices, respectively:

```
...-P [parttype] -100
...-P [parttype] -10
```

-N **Don't Merge Lower-level PIN Files**

The -N option prevents PIN2XNF from merging in any lower-level PIN files, including those representing Xilinx macros. In the default mode, PIN2XNF automatically translates and absorbs the PIN files for any functional blocks or macros that are hierarchically below the input design file. If the -N option is used, the XNF file created represents only the input design file.

-X **Merge Xilinx Macros Only**

The -X option prevents PIN2XNF from merging in any lower-level PIN files that represent user-defined macros, although Xilinx macros are translated. In the default mode, PIN2XNF automatically translates and absorbs the PIN files for any functional blocks or macros that are hierarchically below the input design file. If the -X option is used, the XNF file created will represent only the input design file and any Xilinx macros it contains.

Error Messages and Techniques for Recovery

ERROR 1 Invalid PINLIST file. Bad or missing PINLIST Record.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 2 Incorrect PINLIST version= <number>, should be '2'.

PIN2XNF only supports version 2 PIN files. Use the DCM and PINC programs to generate a version 2 file.

ERROR 3 Expected DRAWING record.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N *<reference number>* to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 4 Missing *filename* on DRAWING record.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N *<reference number>* to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 5 Bad or missing path ref on DRAWING record.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N *<reference number>*" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 6 Bad or missing reference number on SYM record.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message: the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N *<reference number>*" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

Warning 7 Non-group SYM record. Ignored.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N *<reference number>*" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 8 Unbalanced groups. Missing ')' record.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N <*reference number*>" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 9 No type defined for symbol.

Check the attribute and point of effect for the name of this symbol. All user-defined functional blocks should carry a name with attribute FILE (8).

Warning 10 Invalid record type for SYM group.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N <*reference number*>" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 12 Invalid pin attribute number [number].

PIN2XNF allows only the PINI (23), PNBT (22), and PINO (21) attributes on symbol pins. Examine the PIN file at the specified line to find the symbol that has an invalid attribute, and correct this attribute on the drawing.

ERROR 13 Missing pin name.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N <*reference number*>" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

Warning 14 Bad or missing attribute on DATA record.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N <*reference number*>" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

Warning 15 Missing text on DATA record.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N *<reference number>*" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 16 Unknown reference [number].

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N *<reference number>*" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 17 Missing filename on PATH record.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N *<reference number>*" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 18 Bad or missing symbol reference number on PATH record.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N *<reference number>*" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 19 Record type '(SYM' not allowed to be a group.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N *<reference number>*" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 20 Unknown record type.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N <*reference number*>" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

Warning 21 nextrec: Input ignored: '[string]'.

Use DCM and PINC to regenerate the PIN file, and check for any errors or warnings reported by these programs. If the error persists, search the PIN file for the first SYM statement above the line specified in the error message; the number at the end of the SYM line is the reference number of the symbol causing the error. Examine this symbol on the drawing for any irregularities (use the FutureNet command N <*reference number*>" to locate the appropriate symbol). If the symbol appears correct, an invalid or corrupted PIN file may be indicated. Contact Data I/O Customer Resource Center.

ERROR 38 Error while writing XNF information to disk.

This probably indicates a disk full condition. Free some disk space and rerun PIN2XNF.

ERROR 39 Unable to rename temp '[source file]' to '[target file]'.

PIN2XNF could not copy the temporary workfile into the user-specified output file. This will occur if the target file already exists and is flagged read-only by DOS. Delete or rename the existing target file and rerun PIN2XNF.

ERROR 42 Unable to open pinlist file '[*filename*]'for reading.

Check that the PIN filename is specified correctly and exists in the current directory.

Warning 45 Unable to find file [*filename*] for symbol [symbol name].

The PIN file for the specified symbol was not found. If the symbol is a user-defined macro, the FILE (8) name on the symbol must match the name of the PIN file, and the PIN file must exist in the current directory. If this error appears for a Xilinx macro, verify that the PIN file for this macro exists in the proper directory (for example \XACT\PIN4 for XC4000 devices), and use the DOS SET command to verify that the XACT environment variable has been set to point to the \XACT directory. Any extra space characters in the SET XACT= statement may cause problems.

ERROR 49 Invalid pin name <name> on symbol <name>, type <type>.

The specified symbol has a pin that is not valid for that symbol type. This should only occur if a pin name is changed on a Xilinx primitive symbol (such as a gate or flip-flop). If this message appears, replace the corrupted symbol with the correct version from the Xilinx library.

ERROR 50 Unknown attribute [attribute] on pin [pin name] of symbol [symbol name].

PIN2XNF allows only the PINI (23), PNBT (22), and PINO (21) attributes on symbol pins.

ERROR 51	Pin [pin name] of symbol [symbol name] is not invertible. The indicated pin is not invertible, but is specified as inverted. This should only occur if a pin name on a Xilinx primitive symbol (such as a gate or flip-flop) has been inverted with an overbar. If this message appears, replace the corrupted symbol with the correct version from the Xilinx library, adding a separate inverter if necessary.
ERROR 52	Logic symbol [symbol name] has [number] inputs (max=5). PIN2XNF allows gates with a maximum of 5 inputs. All gates in the Xilinx libraries have 5 inputs or less. Use these gates to generate the desired function.
ERROR 53	Out of memory. Needed [byte size] bytes. PIN2XNF reports how much memory was needed. Check that the system meets the minimum memory requirements.
ERROR 68	The -x and -n options can not be used together. To flatten only Xilinx macros, use -x by itself. To prevent any flattening, use the -n option by itself.
Warning 69	Missing part type for -p option. Ignored. When the -p option is specified on the command line, the next parameter must be the LCA part type desired.
Warning 70	Unknown flag ignored. An unknown option flag was specified on the command line. Remember to include the leading dash (-) on all option flags. Type PIN2XNF with no parameters to see a listing of the valid command line options.
Warning 71	Extra argument [argument] ignored. An extra argument was found on the command line, following the input PIN filename and the output XNF filename. This extra parameter will be ignored.
Warning 72	Part type [part type] ignored. (Part type already set to [part type]). Part type already set. The target part type can be specified in the schematic (use the attribute 81), or on the command line using the -p option. If specified in both, the -p specification will take precedence, causing this warning to appear. This warning may also appear if there are two conflicting part types specified on the schematic.
Warning 73	Error messages found from the PINLIST program found in file. Error messages from the PINLIST program found in file. The PINLIST program is an ancestor of the current FutureNet programs DCM and PINC. If PINLIST is used to generate a PIN file, any error messages will be placed in the PIN file itself. PIN2XNF issues this message when it finds these errors in the PIN file. If this occurs, rerun the PINLIST program to locate errors. Note that the PINLIST program is not supported in the Xilinx design flow, and the use of DCM and PINC is recommended.
Warning 76	No substitution made for pin [pin name] on symbol [symbolname]. If PIN2XNF finds a symbol with an unknown symbol type, it searches for a PIN file that matches the FILE (8) name on the symbol. If any pin on the symbol does not correspond to a signal in the PIN file, this message is issued. Check that the symbol pin and the corresponding lower-level signal name match exactly.

Warning 79 No signals connected to symbol [symbolname].

The specified symbol has no signals connected to it.

Warning 80 No output pins found on symbol [symbolname].

The specified symbol has no signals connected to it.

Warning 81 Invalid name [name]. No non-digit characters used.

The specified name is not a valid LCA name. A valid LCA name should contain only letters, numbers and the characters: _ - $ < and >. Every name must also contain at least one non-numeric character. Note that the characters < and > should only be used for naming buses.

ERROR 82 Non-bus signal connected to bus pin on symbol [symbol name].

A bus pin on a symbol (created with the .= command) can only be connected to a bus signal (drawn with a type /2 line). If a normal wire is connected, this message will be issued.

Warning 86 EQU records found in file [*filename*.pin].

The PINC program places EQU records in the PIN file if a single net has multiple names; PIN2XNF reports this as an error. Search the PIN file for the string EQU to find which signal names have been equated. Edit the drawing and remove these names so the net has only one name.

ERROR 93 Conflicting part types determined for [part type]. Part type cannot be determined. Use the -p command line option to specify the part type.

This occurs if no part type is specified (using an attribute 81 label on the schematic or the -p option on the PIN2XNF command line) and if PIN2XNF finds symbols from different LCA family libraries that it cannot convert (see error 94).

ERROR 94 Conflicting symbol types found [symbol type]. Part type is [2000, 3000, or 4000] family and the symbol is from the [2000, 3000, or 4000] library.

PIN2XNF will issue this message if symbols from different LCA family libraries are found and if PIN2XNF cannot make a conversion to the specified part type.

ERROR 95 Symbol [symbolname] was found that is not unique.

Normally PIN2XNF will assign unique names to every symbol in the design. However, if symbols carry an attribute LOC (2) label, PIN2XNF will use that name instead; this message indicates that the LOC (2) labels are not unique. Either make each LOC (2) label unique, or remove them and allow PIN2XNF to assign symbol names.

ERROR 96 Unknown part type [part name] specified.

A list of valid part types is displayed after this message. PIN2XNF will choose a default part type; if this is not the desired part, use either an attribute 81 label on the drawing or the -p option in PIN2XNF to specify a valid part type.

ERROR 98 [delay name] has multiple delay starts and delay ends.

A set of delay flags with the same unique identifier can have either multiple starting points (DS) or multiple ending points (DE), but not both.

ERROR 99 [delay name] has no [start or end].

Both the starting point (DS) and ending point (DE) of a delay path must be specified.

ERROR 100 pin [pin name] has a delay start and a delay end.

Multiple delay starts (DS) or delay ends (DE) can be attached to the same pin, but a mixture of delay starts and ends cannot.

ERROR 101 Delay with no name.

All delay flags must have a unique identifier.

Warning 102 Mismatched Delay end and Delay start values on [*filename*] ignored.

If values are specified at both ends of a delay path, these values must be identical, or one end must be left blank (and they are assumed to be identical). If one end of the delay path has multiple flags and each delay has a different value, the opposite end should be left blank. If the end with multiple flags is left blank, the value at the opposite end is assumed for all paths.

Warning 103 Missing or invalid INIT value on [*filename*].

This message indicates that a flag (such as -W" or -SC") is not completely defined unless it carries a value. Similarly, the INIT= parameter on a ROM symbol must be defined.

Warning 105 Partname was not specified and unable to determine from PINLIST. Assuming default part.

If no part type is specified (using an attribute 81 label on the schematic or the -p option on the PIN2XNF command line), PIN2XNF tries to determine the part type based on the primitives used. If this fails, it assigns a default part type of 4005PG156.

ERROR 106 Delay [delay name] not separated from logic with buffer.

This error is caused when a delay is connected to a pad with no buffer.

Warning 107 Bus with no name in [*filename*].

This warning occurs when a bus is not named in the schematic. Buses must be renamed.

Warning 108 Unnamed signal [arbitrary net name] in design.

This warning occurs when a net was not named in the schematic. Xilinx recommends that all nets in a design be named since it makes design debugging simpler.

The XNFCVT Program

Overview of XNFCVT

In order to support the Xilinx XC4000 Logic Cell Array (LCA) family, Xilinx has changed some of its XNF file specifications. Earlier, Xilinx changed the XNF file specification to support logic partitioning control.

The changes from version 1 to version 2 are summarized below.

- New mapping symbols to control partitioning.
- A new pin lock signal flag, the P flag.
- A new save signal flag, the S flag.
- Each symbol and signal now has a full hierarchical path name.
- The / character is included in signal and symbol names.
- The LCANET version number changed from 1 to 2.

The changes from version 2 to version 4 are summarized below.

- New pin parameters to aid delay-driven routing.
- New symbols to support XC4000 architecture, e.g., WAND, BSCAN, BUFGP.
- Bus records were added to provide more flexibility in bus naming.
- OUTFFZ and OBUFZ were changed to OUTFFT and OBUFT.
- New parameters for logic placement.

These changes to the XNF file make necessary a program that converts the new XNF file format into the old XNF file formats so that the file is compatible with those simulator translators that do not yet support the new XNF formats. This program is called XNFCVT (XNF Convert). It enables Xilinx to make the required XNF specification changes without impacting existing simulator translation software.

XNFCVT translates a new version 4 XNF netlist into an old version 2 XNF netlist or version 1 XNF netlist. XNFCVT also converts a version 2 XNF netlist to a version 1 XNF netlist. The way to determine the XNF file version is to look at the first line of the XNF file. The number following LCANET is the version of the netlist

Before running an XNF-to-simulator translator (like XNF2SILO) which handles version 1 XNF files only, you would run XNFCVT to translate a version 4 or 2 netlist into a version 1 netlist.

The XNFCVT Command

Syntax XNFCVT [-A] input_file[.XNF] [output_file[.XNF]]

Options

-A Instructs XNFCVT not to use the existing AKA file for generating hierarchical prefixes. When this option is used, a new AKA file will be generated. If the -A option is not specified, the program automatically looks for an AKA file with the same name as the input file. If one is found, that file is used to generate shortened name prefixes. This allows prefixes used in previous runs to be maintained.

-V *<version>* The -V option specifies the XNF file version of the output XNF file. Valid versions are 1 and 2. If no version is specified, XNFCVT converts to the previous version. For example, a version 4 XNF file would convert to a version 2 XNF file, and a version 2 XNF file would convert to a version 1 XNF file.

Note: *If the part type in the input XNF file is from the XC4000 family, XNFCVT will not allow conversion back to a version 2 or version 1 netlist.*

Input File *design*.XNF

Version 2 XNF file.

Output File *output*.XNF

Version 1 XNF file.

XNFCVT Program Process

The XNFCVT program performs the following functions:

- Reads in a version 4, 2, or 1 XNF netlist file.
- Removes all mapping control (CLBMAP) symbols (version 2 to version 1).
- Removes all save signal flags (S flags) (version 2 to version 1).
- Removes all pin lock signal flags (P flags) (version 2 to version 1).
- Reads and updates the AKA alias names file (version 2 to version 1).
- Shortens hierarchical path names of symbols and signals (version 2 to version 1).
- Generates an XNF file, corresponding to the specified version.
- Removes delay flags (version 4 to version 2 or 1).
- Bus records are removed (version 4 to version 2 or 1).
- OUTFFT and OBUFT are changed to OUTFFZ and OBUFZ (version 4 to version 2 or 1).

The AKA File (Version 2 to Version 1 Only)

The first time XNFCVT is run on a file, it generates an AKA file containing a prefix name (automatically generated) and the corresponding path name that the prefix represents. In each successive run of the program (without the -A option), the AKA file is read along with the XNF file. The program uses existing prefixes from the AKA file for identical path names and only generates new prefixes when a new path name is encountered. Also, you can edit this file in order to make prefix names more meaningful. For example:

```
prefix          path name
   1            /TOP/U12
   2            /TOP/U28
   3            /TOP/U12/COUNTER
```

In the XNF file produced by XNFCVT, symbols and signals that are at the level /TOP/U12 have a shortened name; 1 replaces the path /TOP/U12. For example, if there is a signal in the input XNF file called /TOP/U12/SIG1 it is called 1-SIG1 in the output XNF file.

Note: *Since the current version of the XNF specification does not support the / character, the - character is used in its place as the character that separates the prefix from the symbol or signal name.*

Now suppose you run XNFCVT again after adding a new hierarchy level called U30, deleting the one called U28. The AKA file looks as follows.

```
prefix          path name
  1             /TOP/U12
  3             /TOP/U12/COUNTER
  4             /TOP/U30
```

If you run XNFCVT with the -A option, the existing AKA file is ignored and a new AKA file is generated that looks as follows.

prefix	path name
1	/TOP/U12
2	/TOP/U12/COUNTER
3	/TOP/U30

You can edit the prefixes to make them more meaningful. Prefix names should not contain the separator character and they should be limited to 16 characters.

XNFCVT Error Messages and Recovery Instructions

ERROR 200 Can only process files with the .XNF suffix.

XNFCVT only works on files with an XNF extension. Do not specify a *filename* that doesn't have an XNF extension.

ERROR 201 Unable to open file [*filename*] for [reading or writing].

If the file could not be opened for reading, it is because the input file does not exist. Check the *filename*. If the output file could not be opened for writing, check if the hard disk is full or check if the output file already exists and is write-protected.

ERROR 203 Illegal part [parttype].

Check the part type specified in the input XNF file and make sure it is a valid part type.

ERROR 204 Premature END OF FILE.

This error indicates a corrupted XNF. Check the bottom of the input XNF file and see if part of the file is missing. The file should end with an EOF statement. If the input file is corrupted, regenerate it. Check if there is enough hard disk space, since many premature end of file errors are caused by files written to full disks.

ERROR 205 No parttype specified in source file.

This error indicates that the input XNF file did not contain a part type. Part types must be specified, so regenerate the input XNF file with the part type.

ERROR 206 Invalid LCANET [version]. Valid types 2 and 4.

XNFCVT converts level 4 and level 2 XNF netlist versions only. Check that the input XNF file is LCANET version 1 or LCANET version 2. If it is an LCANET version other than 4 or 2, XNFCVT cannot convert it.

ERROR 207 Invalid conversion path from LCANET version [version] to [version].

XNFCVT converts from LCANET version 4 to version 2 or 1, and from LCANET version 2 to 1. Make sure that the input file is LCANET version 4 or 2 and that the output file LCANET version is either 2 or 1.

ERROR 208 Bad command line option.

XNFCVT has two options: -A and -V. Make sure that an invalid option was not entered. The -V option has two possible parameters: 2 or 1. Make sure that an invalid parameter was not entered.

DATA I/O