PLDS: PROTOTYPING IN LAGER USING DECOMPOSITION AND SYNTHESIS

by

Robert K. Yu

Memorandum No. UCB/ERL M91/53

11 May 1991

Contents

PLDS: PROTOTYPING IN LAGER USING DECOMPOSITION AND SYNTHESIS

by

Robert K. Yu

Memorandum No. UCB/ERL M91/53

11 May 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Contents

PLDS: PROTOTYPING IN LAGER USING DECOMPOSITION AND SYNTHESIS

by

Robert K. Yu

Memorandum No. UCB/ERL M91/53

11 May 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

PLDS: PROTOTYPING IN LAGER USING DECOMPOSITION AND SYNTHESIS

by

Robert K. Yu

Memorandum No. UCB/ERL M91/53

11 May 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objective

Prototyping a system is often the only way to test for correctness and to optimize its behavior with respect to the initial specifications. System prototyping has traditionally been an extremely tedious operation, often requiring the use of commodity components such as microprocessors and memories combined with random glue logic implemented in TTL parts.

Recent advances in technology have yielded various types of programmable devices that could reduce the prototyping time dramatically. Although these devices often use different processes and differ in architectural configurations, they typically have the equivalence of five to ten thousand gates [10] [7].

The objective of *PLDS*, or **P**rototyping in **L**ager using **D**ecomposition and **S**ynthesis, is to provide a solution to map efficiently a given high-level design description of arbitrary size and hierarchy into a set of programmable devices. This solution should optimize the logic and minimize the interconnect between the devices, and it should also be flexible, expandable, and automated both in algorithms incorporated and the targets chosen. So far, these implementation targets include field programmable gate-arrays (FPGA) from **Actel**, SRAM-based gate-arrays from **Xilinx**, electrically programmable logic devices (EPLD) from **Altera**, and full-custom integrated circuits using standard cells from Mississippi State University [1] [8] [11].

1.2 Solution

PLDS provides an interface between the Lager/Oct [6] environment and commercially available tools supplied by the target manufacturers. In the case of mapping to standard cells, *PLDS* generates the layout. With *PLDS*, high-level design descriptions are mapped into structural gate instances using *misII* and *espresso*. Typically, the input descriptions are written in **bds**, a Pascal-like behavioral language. But other input formats, including **blif**, **pla**, and **eqn**, are also acceptable as design entries into *PLDS*.

A fast clustering algorithm can next be used to partition the entire design into several groups while minimizing the connections between the groups. The clustering is performed under three constraints: The maximum number of groups allowed, the maximum number of elements within a group, and the maximum number of connections to a group. These constraints reflect the limits associated with the target device. The entire design is restructured into a new hierarchy according to the clustering results.

To map the final design into a set of devices, *PLDS* then generates a set of files in the netlist file format required by the tools provided by the target manufacturer. *PLDS* automatically inserts I/O buffers at the cluster boundaries and performs simple pin assignments.

Chapter 2

Overview of Design Method

This section provides an overview of the three programs that make up *PLDS*. The details of these tools will be discussed in Chapter 4.

Figure 2.1 shows the design strategy behind the *PLDS* system. *PLDS* consists of three separate programs integrated in the Lager/Oct environment: the structure-processor *x2oct*, the structure-processor *partition*, and the layout-generator *prototype*. *X2oct*, specified at the macrocell level, is used first to generate a structural mapping of a design described in bds. Once all behavioral descriptions in the design have been completely mapped, *partition*, specified at the topcell level, is used to perform physical partitioning of the entire design with the objective of minimizing interconnects between partitions. Lastly, *prototype* is used to generate the files in the netlist file format specified by the target manufacturer. These programs are typically invoked automatically by *DMoct* but may also be run stand-alone.

Since these three programs are modular, not all three need to be used for all applications. For example, *x2oct* may be skipped when the user instantiates cells directly from the target library and does not have any behavioral descriptions. Also, *partition* may be skipped altogether when the user wants to map the entire design into a single cluster. In addition, *partition* may be used by itself to obtain a minimal cut of edges to a graph, or to partition a full custom design into only a few packages.

2.1 Design Mapping

The design mapping from a behavioral description to a structural one is performed by the structure-processor *x2oct*. In Lager/Oct terminology, *x2oct* generates a structure-

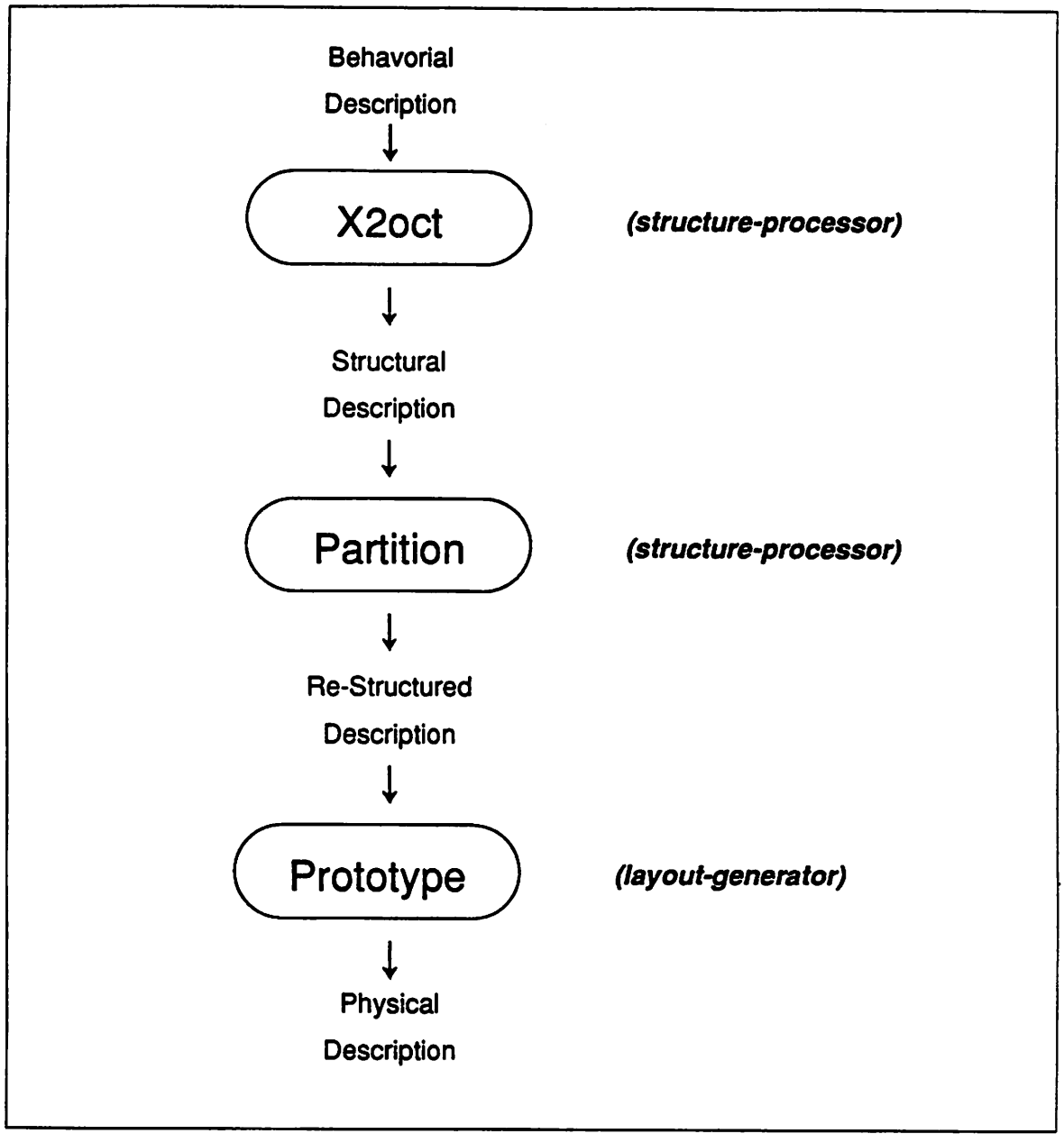

Figure 2.1: *PLDS* design strategy using two structure-processors and one layout-generator.

instance facet that contains only instances from a target library.

To achieve this end, *x2oct* takes as inputs the user's description either in **bds**, **blif**, **pla**, or **eqn** formats, the target device, and any optional constraints placed on the mapping. These parameters are specified as formal parameters in the .sdl file. *X2oct* then creates a *misII* script based on the target library and any constraints, and invokes *misII* to perform the mapping. The output of *x2oct* is an Oct facet representing the implementation of the description.

X2oct provides two main strategies to mapping designs, which I shall loosely refer to as "decomposition" and "synthesis". Decomposition is the technology mapping to find an implementation of a design given a target library. This is analogous to finding trees in a forest, given a set of particular trees to look for. Synthesis, on the other hand, does not require a library and makes use of hard-coded pattern graph routines in *misII* specifically written for the target device. (Currently, hard-coded routines for only **Actel**, **Xilinx**, **Altera** are supported.) It typically uses a tree-mapping approach to cover the subject graph with the pattern graph. This is also analogous to finding trees in a forest, but without using a set of particular tree templates.

2.2 Partitioning

Structural partitioning is performed by the structure-processor *partition. Partition* takes as input a design in Oct of arbitrary size and hierarchy, creates new structure-masters and structure-instances representing the newly formed clusters based on the given constraints, and outputs the design represented by two levels of hierarchy: the topmost level which interconnects the various clusters, and the cluster themselves which contains only leafcell instances.

A quick clustering algorithm performs the partitioning under three constraints: the maximum number of partitions allowed, the maximum number of nets cut by any partition (or the maximum number of I/O terminals), and the maximum number of elements within the partition. These constraints are specified as formal parameters in the **.sdl** file.

Here, the definition of "element" refers to some measure of logic capacity of the target devices, and differs from one manufacturer to another, depending of the equivalent logic granularity. For example, in the case of **Actel**, an "element" consists of three reconfigurable two-inputs multiplexers (**BASIC BLOCK**). In the case of **Xilinx**, an "element"

is a set of combinatorial logic, multiplexers, and registers ("Configurable Logic Block" or **CLB**). And in the case of **Altera**, an "element" contains a 30-input, 8-output PAL, a reconfigurable register, and an output buffer (**Macrocell**).

2.3 Design Targeting

The layout-generator *prototype* provides a design entry point into the various tools provided by the target manufacturers. *Prototype* automatically inserts the appropriate I/O buffers at the partition boundaries and generates files in a netlist format suitable for the target manufacturer's design environment. For example, in the case of **Actel**, *prototype* generates the required **.adl**, **.pin**, and **.crt** files. In the case of standard cell, *prototype* invokes the standard cell place and route program *Stdcell* to generate the layout.

2.4 Functional Verification

Functional verification in the Lager/Oct environment is performed by *THOR*, an event-driven simulator developed at Stanford University. *PLDS* also uses this approach. Associated with each primitive leafcell is a *THOR* model, which is a description of the leafcell's functionality encoded in the C language. A functional model of the user's entire design is obtained by combining the design's connectivity with the leafcell's functional models.

Chapter 3

Details of Organization

Files related to the *PLDS* package reside under the conventional subdirectories **src**, **man**, **examples**, **bin**, and **lib**. Figure 3.1 shows a diagram of the file structure. The **lib** directory, which contains the target libraries, default files, and *misII* scripts, are essential to *PLDS* operation and are discussed in detail here. Details of the *PLDS* Oct facet policy, consistent with the Oct symbolic policy, are provided in the Appendix A.

3.1 Abstraction Through Translation Libraries

Although *misII* will map directly to the particular target library, some designs may not use the *misII* approach and make direct use of library cells instead. Also, some targets lend themselves to more efficient implementations than other targets. These two reasons suggest the importance of supporting portable designs that may be retargetted.

To allow these designs to be easily retargetted from one manufacturer to another, the target libraries are organized with an added layer of hierarchy. This added layer of abstraction hides the different naming and structural details of the target manufacturer and also provides a translation from a generic library convention to the target library. That is, this translation layer provides a "wrapper" such that the target library appears identical to other libraries. Changing from one target to another requires the simple change of one formal parameter and the search order in the lager file before running *DMoct*. Figure 3.2 shows a design of a VME address decoder targetted towards **Actel**. This same design is shown in figure 3.3 but targetted towards **Xilinx**. The only difference between the two designs is the specification of the target manufacturer. For example, in the case of

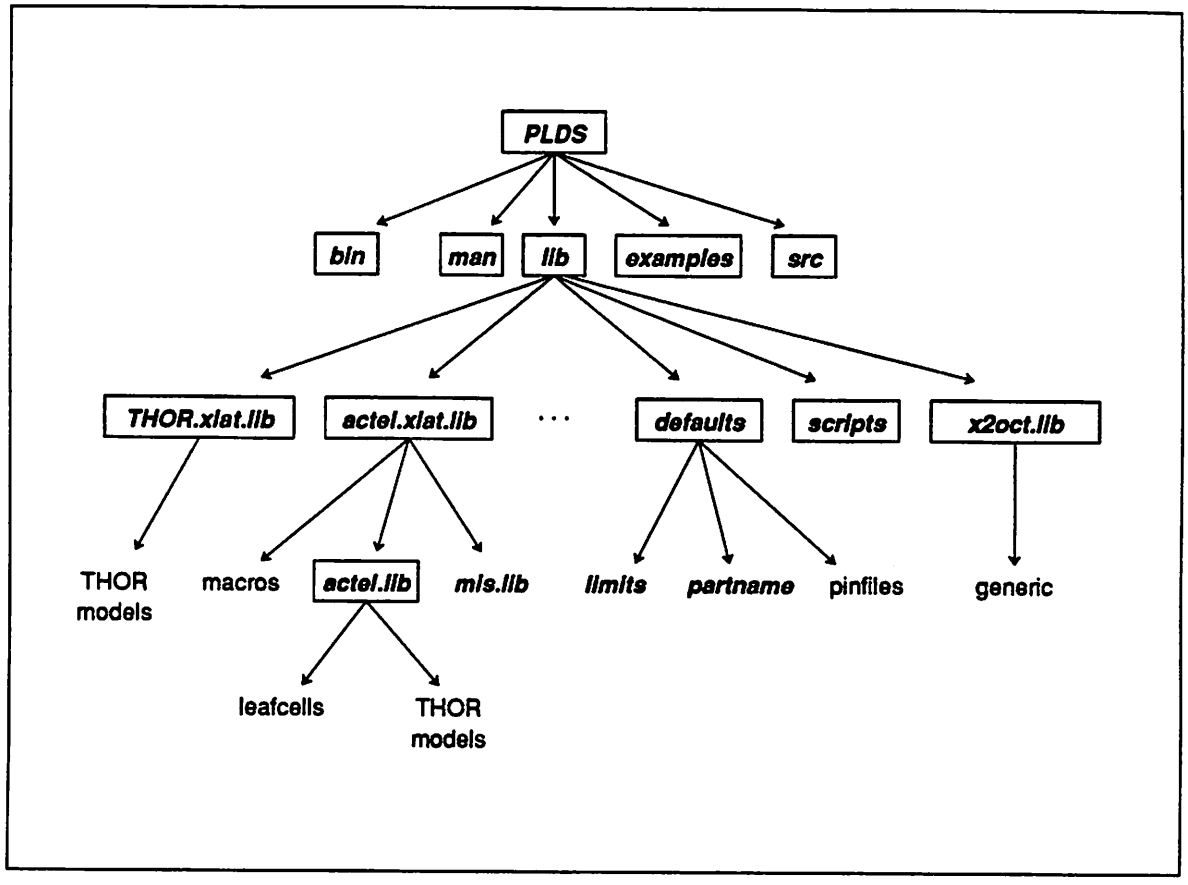

Figure 3.1: Structural organization of files used by *PLDS*.

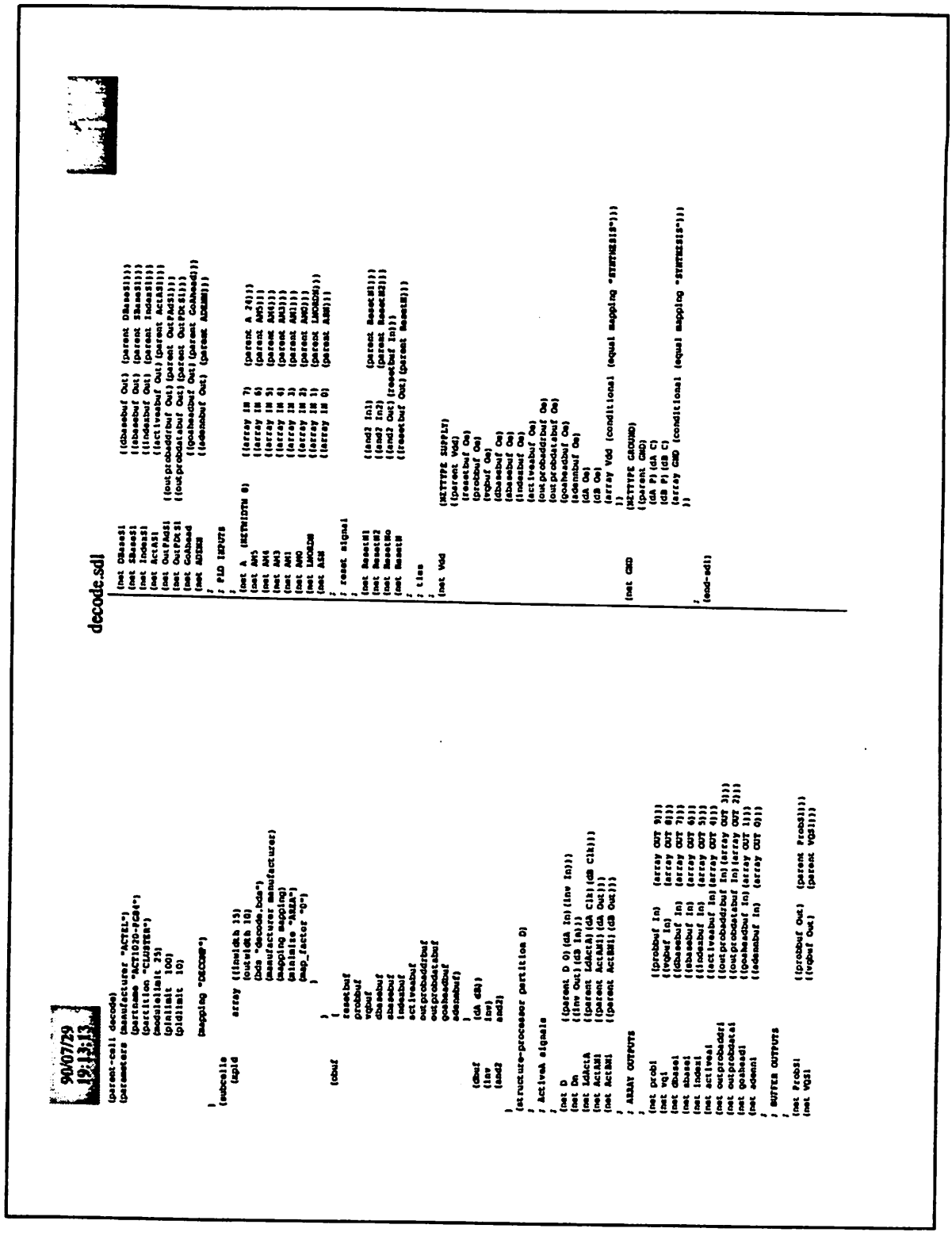

Figure 3.2: Design example using **Actel** target.

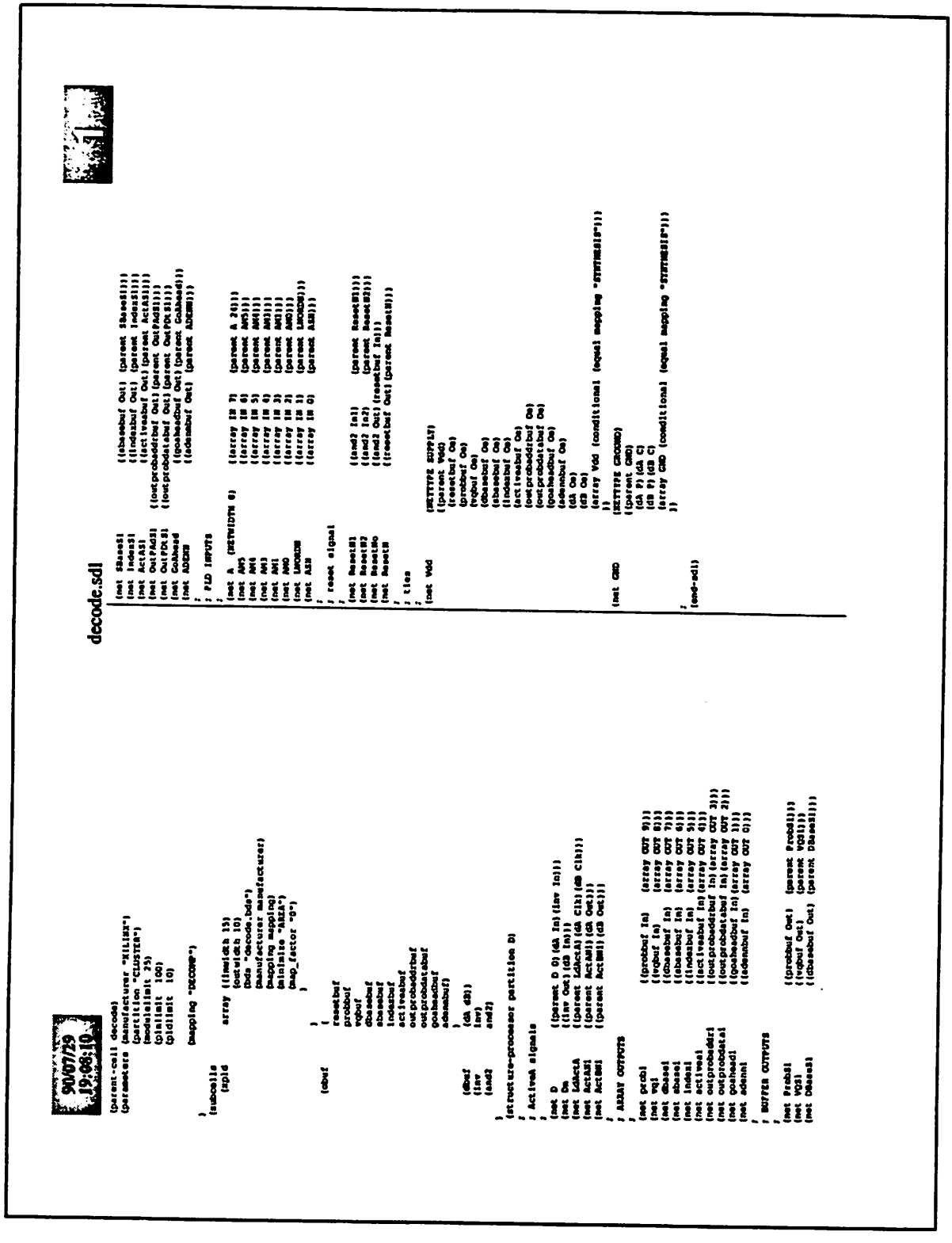

Figure 3.3: Design example using Xilinx target.

Actel, the translation macros reside under directory **lib/actel.xlat.lib**, and the particular actel primitive leafcells reside under the directory **lib/actel.xlat.lib/actel.lib**. Another potential advantage to using the translation library is that the same *THOR* models can be used across different targets, since the macros have the same terminal definition and functionality. But to allow for functional verification of designs that instantiate leafcells within the manufacturer's primitive library directly, *THOR* models should also exist for the primitive library as well.

To successfully hide the details of the different manufacturers' libraries, the translation library should be comprehensive enough to allow for the sole change of the target implementation without the need for modifying the design description. The trade-off for this design flexibility may be the efficiency of primitive usage in a particular library. But in typical applications, users will instantiate registers, buffers, and a few logic gates from the translation library, with the majority of complex logic functions described behaviorally rather than structurally. Since behavioral descriptions are mapped directly to the manufacturer's library and not the translation library, these designs will not suffer from inefficient primitive usage. Thus, the translation library should be made as comprehensive as possible to allow for design flexibility.

The libraries include both the structure-masters as well as the structure-instances since it is not necessary for the user to create local versions of the structure-instances, thereby saving disk space. The reason structure-instances need not be created locally is that the terminals within each library cell are always fixed in name and number.

3.2 MisII Scripts and Libraries

Under the directory **lib/scripts** is a set of *misII* template scripts used by *x2oct* and *misII*. Since *misII* does not currently support parametrized scripts, *x2oct* creates custom scripts by simple substituting of key string parameter names by parameter values into these templates. These parameter values are obtained through the formal parameters specified in the **.sdl** file. *X2oct* provides default parameter values if they are not specified. The set of template scripts provide different mapping strategies for the various targets. The details of these scripts are provided in the Appendix C.

The libraries used by *misII* to perform the technology mapping reside under the translation library. For example, in the case of **Actel**, the *misII* library resided under

lib/actel.xlat.lib/mis.lib. The primitives in the *misII* library are the same as the primitive leafcell library.

One other library used by *x2oct* deserves special mention. The directory name **lib/x2oct.lib** contains "generic" leafcells. These leafcells all have one output, a varying number of inputs, and no logic function associated with them. This library is needed by *misII* in some cases of mapping to **Xilinx** or **Altera**. *MisII* attaches logic functions to these generic cell instances when writing mapped oct facets.

3.3 Defaults

Under the directory **lib/defaults** are files used by *partition* and *prototype*. These default files are included in Appendix B as well.

Partname

The file **partname** specifies the default device to which to map the given target manufacturer name. The structure- processor *partition* uses this default to obtain the partitioning constraints. (Note that the structure- processor *x2oct* only needs the name of the manufacturer and not the partname in order to perform the technology mapping.) This file consists of two columns. The first column is the name of the manufacturer, and the second is the default part name.

Partitioning Constraints

The file **limits** specifies the default partitioning constraints to which to map given the part name. This file is also required by the structure-processor *partition*. The file **limits** consists of four columns: The first is the manufacturer's name. The second is the key partname. The third is a number representing the maximum number of elements that this part can implement. The fourth, also a number, represents the maximum number of signal pins on the part. This number represents the maximum number of nets that can be cut by any particular partition.

Another number used by the structure-processor *partition* is the maximum number of parts allowed to implement the design. The default number is set within *partition* itself.

Pin Allocation

In addition to the files **partname** and **limits** under directory **lib/defaults** are a set of files that specify the signal pins available on a particular part. The layout-generator *prototype* uses these files to make simple one-to-one pin assignments. *Prototype* assumes that each pin listed can be used as either input or output.

Chapter 4

Details of Tools

The previous sections discussed in general terms the *PLDS* design method and the organizational background. This section shall present in detail features and algorithms used in the programs *x2oct*, *partition*, and *prototype*. Further details of individual programs developed for these processors are provided in Appendices D, E, and F.

4.1 X2oct

Design Flow

Figure 4.1 shows a detailed design flow of the *x2oct*. Under the Lager environment, this structure processor is invoked after the structure has already been created. Here, the structure created by *DMoct* is a "footprint" Oct facet. That is, the facet does not contain any structural information relating to the implementation of the design. It is *x2oct* that orchestrates the various tools to replace this "footprint" facet with one that contains implementation details.

To do this, *x2oct* extracts all the formal parameters from the facet and creates a *misII* script based on their values. If the user is starting from a **bds** description, any bds parameters are replaced by their values. *X2oct* converts design inputs, specified in **bds**, **blif**, or **pla** formats, into the **eqn** format. During the translation process, details such as handling **DONT_CARES** and conforming to Lager/Oct policy naming conventions are performed.

MisII is then invoked with the design description, now in **eqn** format, and the

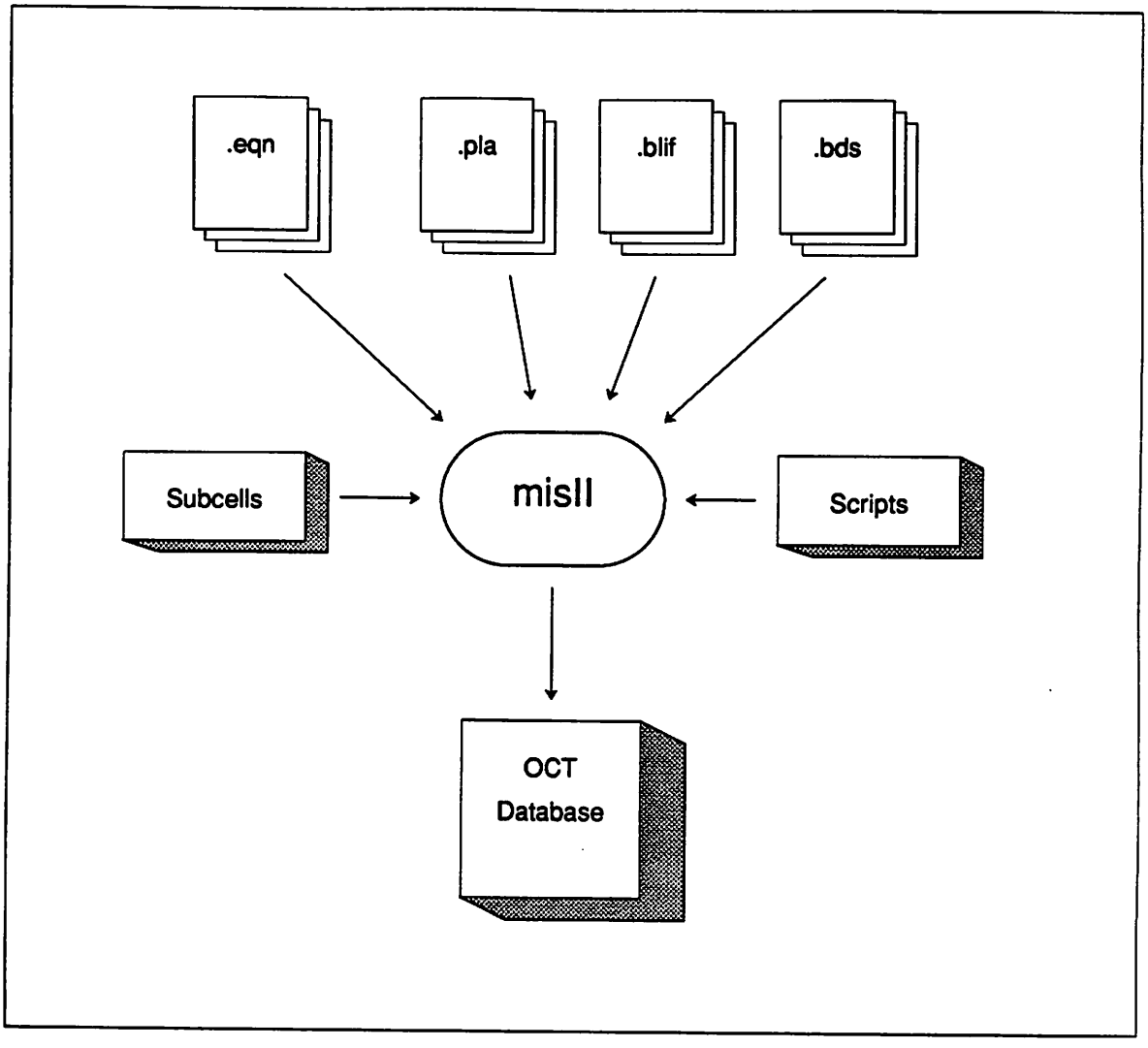

Figure 4.1: Design flow for structure-processor *x2oct*.

customized *misII* script as inputs. Depending on the type of mapping, some patchwork tasks are performed after *misII* is finished. For example, in the case of mapping to **Actel**, *misII* writes out a "bdnet-like" file. In order to create an Oct facet out of this file, this file is corrected before it is sent to "bdnet". In the case of **Xilinx** and **Altera**, if the mapping has been performed without the use of a particular library, the output from *misII* is a set of functionless blocks. Associated with each output terminal of a block is a **LOGICFUNCTION** property, which is a logic equation that expresses the block's output as a function of its inputs. *X2oct* replaces each functionless block by an equivalent generic block from the **x2oct.lib** and retains the **LOGICFUNCTION** as an instance property.

After the new oct facet has been created, *x2oct* checks for terminal consistency with the "footprint" facet. If any terminal present on the "footprint" facet but not present on the new facet is found, a warning message is issued and this missing terminal is created in the new facet. This occurs in the case where Vdd and GND terminals are required to generate the logic "0" and "1" within the facet. Also, properties and formal parameters from the "footprint" facet are copied over to the new facet. This new facet then replaces the "footprint" facet.

Input Styles

X2oct supports two input styles. One approach is similar to the structure-processor *plagen* and the other is similar to the structure-processor *Bds2stdcell.* In the *plagen* approach, a special cell master called **xpld** serves as a template and is instantiated for each **.bds** description. **Xpld** is nothing more than a cell that calls the structure-processor *x2oct* and that adds another layer of hierarchy into the design. The template is included in Appendix G. In the *Bds2stdcell* approach, the **xpld** template is not used, and users must create their own. Figure 4.2 show a design using the **xpld** template. Figure 4.3 shows the same design without using the template. Both styles are supported for compatibility with existing designs that use these different approaches. There are no real advantages to using one or the other, except that in the **plagen** approach, the terminal names are renamed **IN[i]** and **OUT[j]** due to the extra layer of hierarchy imposed. The user may, however, find this renaming undesirable especially when the number of inputs and outputs become large.

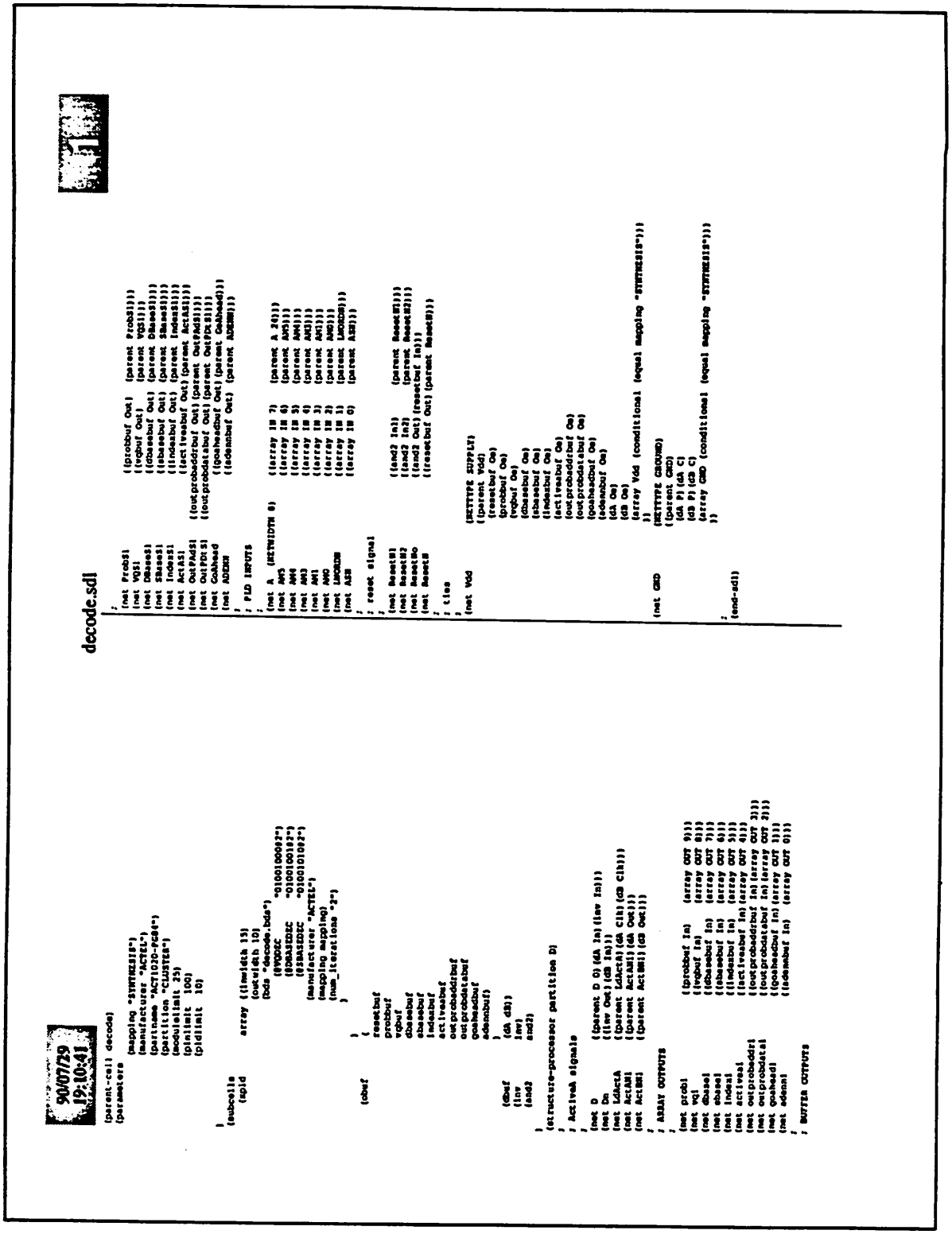

Figure 4.2: Example of design using **xpld** template.

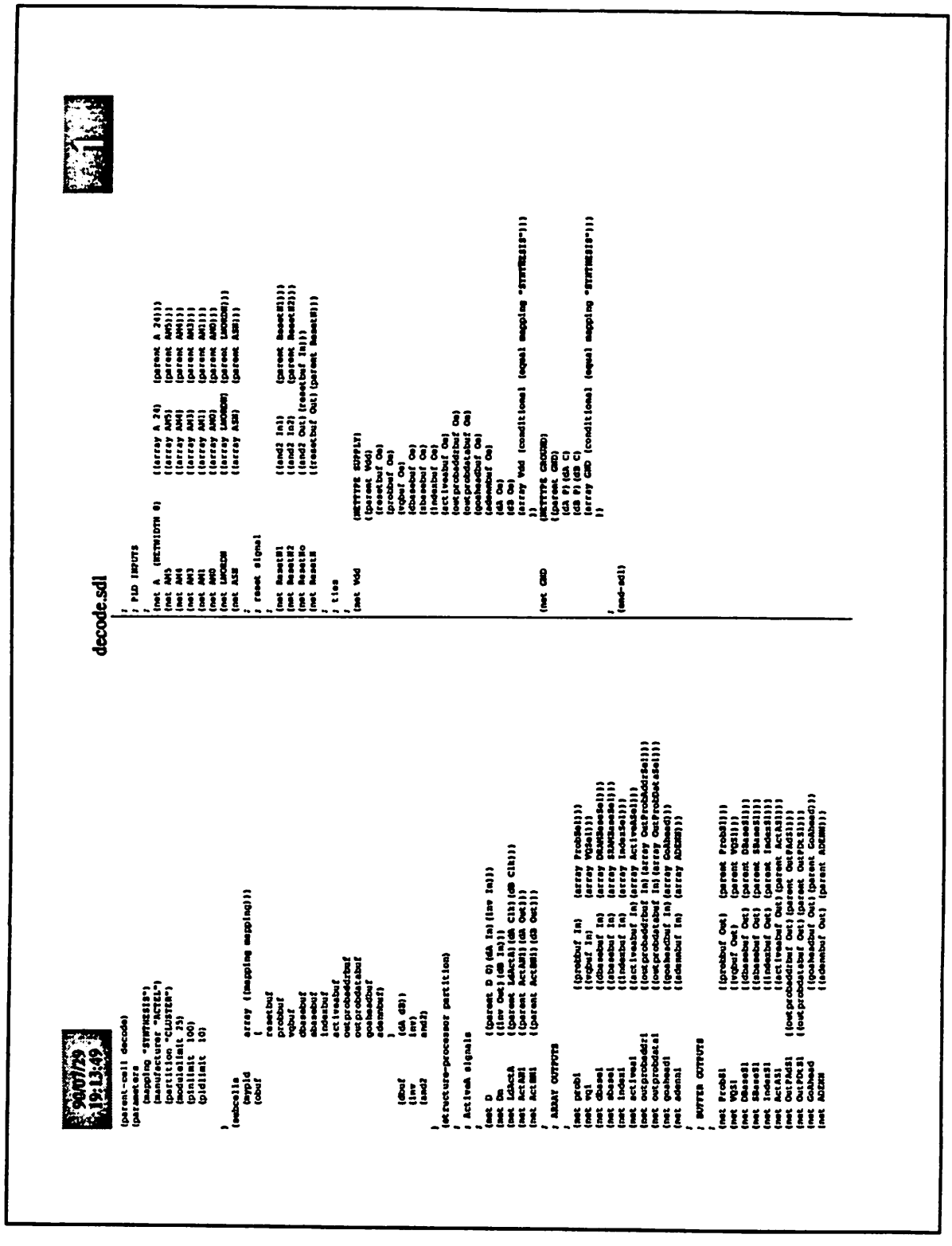

Figure 4.3: Example of design without using xpld template.

Formal Parameters

Table 4.1 shows the formal parameters used by *x2oct*. There are a total of 21 formal parameters supported by *x2oct*, two of which are required, and the other 19 optional. First, the user must indicate the target **manufacturer** to which to map. Valid manufacturer values are **ACTEL**, **XILINX**, **ALTERA**, and **STDCELL**. Second, the user must specify the type of input description and the file name which contains the description. The user specifies this by supplying the file name as the value to parameters **bds**, **blif**, **pla**, or **eqn**.

If the **xpld** template subcell is used, then the user must supply values to the parameters **inwidth** and **outwidth**. Otherwise, these may be omitted.

The rest of the optional formal parameters deal with the technology mapping performed by *misII*. The mapping strategy is specified as **DECOMP**, **SYNTHESIS**, **CUSTOM**, or **INTERACTIVE**.

In the case of **DECOMP**, where the mapping is performed using the manufacturer's primitive library, the user may further specify whether to minimize for **DELAY** or **AREA**. If **DELAY** is specified, then the design is collapsed into two levels of logic before mapping is performed. If **AREA** is specified, then the design is also factored before mapping is performed. A **map-factor** value between zero to one may be specified to control the minimization of area or delay, respectively. The **speed-up** factor specifies the number of levels of logic to equalize.

In the case of **SYNTHESIS**, the technology mapping is performed without the use of libraries, but with hard-coded routines within *misII*. Currently, only targets **ACTEL**, **XILINX**, and **ALTERA** are supported with this technique.

Under **SYNTHESIS**, if the target manufacturer is **ALTERA**, then there are two additional optional parameters. The **or-fanin** specifies the maximum number of fanins to the OR-gates, and the **and-fanin** specifies the maximum number of fanins to the AND-gates. If the target manufacturer is **ACTEL**, then there are five additional optional parameters. The **heuristic-num** specifies the subject graph selection, the **num-iterations** specifies the number of iterative improvements to make, the **collapse-fanin** specifies to collapse nodes with no more than this number of fanin, the **gain-factor** specifies to iterate only if the product of $cost \times gain\_factor$ is less than the gain, and the **decomp-fanin** specifies to perform a decomposition for nodes with greater than this value. If the manufacturer is **XILINX**, then there are two additional optional formal parameters supported. The

No.	Formal Parameter	Usage	Default Value	Description
1	manufacturer	required	none	target manufacturer (ACTEL, XILINX, ALTERA, STDCELL)
2	bds	one required	none	.bds file name
3	blif	one required	none	.blif file name
4	pla	one required	none	.pla file name
5	eqn	one required	none	.eqn file name
6	inwidth	required with **xpld**	none	number of inputs
7	outwidth	required with **xpld**	none	number of outputs
8	mapping	optional	DECOMP	mapping method (DECOMP, SYNTHESIS, CUSTOM, INTERACTIVE)
9	script	required if CUSTOM	none	custom script file name
10	minimize	optional	DELAY	type of minimization with DECOMP specified (DELAY, AREA)
11	map_factor	optional	0	area/delay weight 0 = minimize area 1 = minimize delay
12	speed_up	optional	4	number of levels to equalize
13	or_fanin	optional	8	maximum OR-gate fanin
14	and_fanin	optional	30	maximum AND-gate fanin
15	heuristic_num	optional	2	subject graph selection
16	num_iterations	optional	0	iterative improvements
17	collapse_fanin	optional	3	collapse nodes with this or less
18	gain_factor	optional	0.01	iterate if gain $\geq$ cost $\times$ gain_factor
19	decomp_fanin	optional	4	good_decomp nodes with this or greater
20	xl_fanin	optional	5	limit on fanins to a node
21	xl_cover	optional	3	heuristic number to solve cover problem

Table 4.1: Formal Parameters Used by X2oct.

xl_fanin specifies the limit to the fanin per node, and the **xl_cover** is a heuristic number used to solve the cover problem.

In the case of **CUSTOM**, *x2oct* will invoke *misII* with the script specified by the user. The user must provide the **filename** of the custom script.

In the case of **INTERACTIVE**, *x2oct* will invoke *misII* interactively. The user will be given a *misII* prompt.

Parametrized MisII Scripts

Since *misII* does not support parametrized scripts currently, *x2oct* gets around this problem by simple string substitution of key values into key words within template scripts. These scripts are provided in Appendix C. The keywords, shown in capitals, are replaced by the formal parameter value specified in the users' **.sdl** file.

Parametrized BDS Files

To allow for more flexibility, *x2oct* also supports the parametrization of key words in **.bds** files. Key words in the **.bds** file are specified by prepending the "@" character to the parameter name. This parameter is replaced by the valued specified as a formal parameter in the **.sdl** file.

Vdd, GND Details

Since *x2oct* replaces the oct facet created by *DMoct* by one created by *misII*, the terminals in both facets must match both in name and in number. In some cases, the oct facet created by *misII* contains Vdd and GND formal terminals. This occurs specifically in the case of mapping to **ACTEL** through the **SYNTHESIS** approach. Recall that in the case, all logic is implemented through reconfiguring the **BASIC_BLOCK** by tying certain inputs to Vdd or GND. In any event, the Vdd and GND terminals must be available for correct facet replacement. If users are using the **xpld** template subcell, then these terminals are created automatically for the cases that need them. Users must provide these terminals when they are not using the **xpld** template subcell.

4.2 Partition

Detail Flow

Figure 4.4 shows the flow for the structure-processor *partition*, which is typically run at the top-most level. *Partition* takes as input an oct structure of any size and hierarchy. The input is usually an oct structure that includes the results generated by *x2oct*, but any input is acceptable.

Given this oct structure, *partition* performs a physically partitioning under three constraints: The maximum number of groups allowed (**pldlimit**), the maximum number of elements (or nodes) within a group (**modulelimit**), and the maximum number of connections (edges cut) to a group (**pinlimit**). Currently, *partition* employs a fast-clustering algorithm to perform the partitioning, but future approaches can be easily integrated. The output of *partition* is a restructured oct facet of two levels of hierarchy functionally equivalent to the original facet.

Partition accomplishes the task of structural partitioning as follows. First, the entire oct structure is flattened down to the leafcell level. Associated with each leafcell in the primitive library is a property called **CELLCOUNT**, which specifies the number of elements required in the given target manufacturer's device to implement the given leafcell. After assigning node and net id's to the flattened structure, a simple pinlist is created representing the network. A clustering algorithm takes this pinlist and generates the partition. The intermediate pinlist input and partitioned output are simple text files that can be viewed and edited for more flexibility. In most cases, these files can be ignored, but are provided for experimentation. The results of the clustering algorithm are included in the *DMoct* log file.

Partition takes the cluster results and creates new oct facets called "cluster-i", where "i" is the cluster id. *Partition* also inserts the appropriate layout-generator in preparation for the generation of files for the manufacturer.

Formal Parameters

Table 4.2 shows the formal parameters used by *partition*. *Partition* supports six formal parameters, only one of which is required. The user must supply either the name of the target **manufacturer** or the name of the particular **partname** to which to map, or

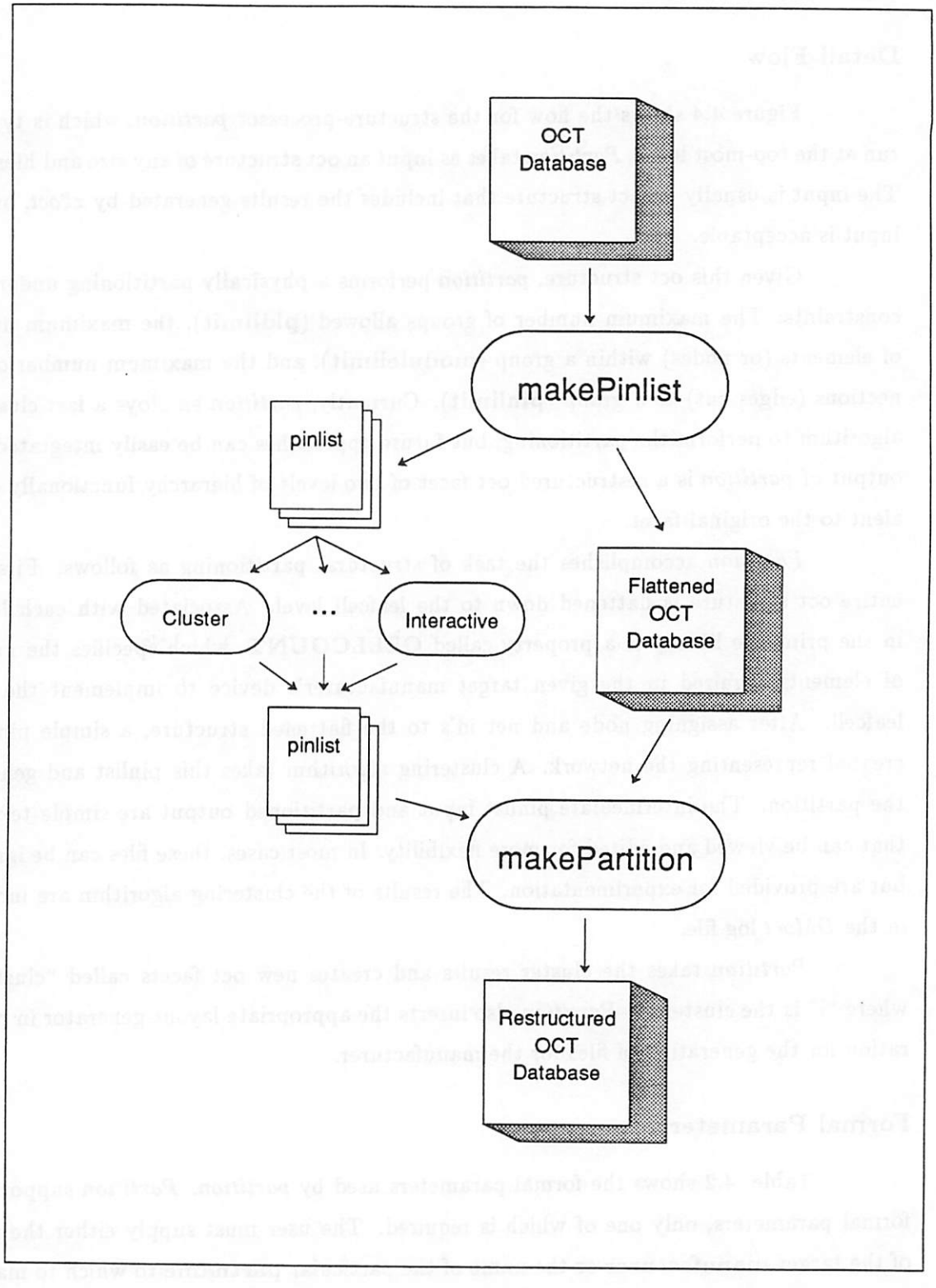

Figure 4.4: Design flow for structure-processor *partition*.

No.	Formal Parameter	Usage	Default Value	Description
1	manufacturer	required	none	target manufacturer (ACTEL, XILINX, ALTERA, STDCELL, NONE)
2	partname	optional	depends on manufacturer	target device
3	partition	optional	CLUSTER	partitioning algorithm (CLUSTER, INTERACTIVE)
4	modulelimit	optional	depends on partname	maximum elements per cluster
5	pinlimit	optional	depends on partname	maximum nets cut per cluster
6	pldlimit	optional	10000	maximum clusters allowed

Table 4.2: Formal Parameters Used by Partition.

both. Given only the name of the manufacturer, "partition" will assume default partname devices.

The formal parameter **partition**, which specifies the partitioning approach, can take values **CLUSTER**, **INTERACTIVE**, or **FLATTEN**. If **CLUSTER**, the default, is specified, then a quick clustering algorithm is used. If **INTERACTIVE** is specified, then users is given a csh prompt, allowing them to manipulate the partitioning by modifying the pinlist file manually or with some other approach. If **FLATTEN** is specified, then the entire structure is flattened and left as one large cluster.

The other optional formal parameters are the constraints **modulelimit**, **pinlimit**, and **pldlimit**, as discussed previously. If these constraints are not given, then *partition* will assume default constraints based on the partname.

Clustering Algorithm

The quick clustering algorithm used is based on the simple principle that the larger the number of nets exist between a pair of nodes, the more these nodes should be placed into the same cluster [3] [2] [4]. The algorithm looks at all node pairs with net(s) between them, and assigns a clustering value with each of these candidate pairs. The clustering value takes into account the conjunctivity and disjunctivity between these pairs. That is,

the cluster value CV between nodes i and j is:

$$CV(i,j) = \frac{I(i,j)}{E(i)} + \frac{I(i,j)}{E(j)} \tag{4.1}$$

where $I(i,j)$ is the total number "internal" nets between nodes i and j, and $E(i)$ is the total "external" number of nets on node i, including $I(i,j)$. From equation 4.1, the clustering value takes on a maximum value of 2 when two nodes have nets that go only from one to another, and values less than 2 when nets on either node i or j or both go to other nodes. Note that the clustering value is 0 when two nodes do not share any nets. However, the algorithm does not even consider these nodes.

After clustering values have been assigned to every candidate pair, these pairs are placed greedily into the same clusters according to their clustering values. After all pairs have been considered and merged, the process is repeated until the desired number of clusters remain. During each merger, the total number of nets external to the cluster is checked against the constraint on the maximum number of external nets allowed, or the **pinlimit**. Merger is not allowed if this maximum is exceeded with the merger.

A simple bin-packing algorithm is also used to prevent merging of pairs that would violate the constraint on the maximum number of clusters allowed. Let N (**modulelimit**) be the maximum number of nodes allowed in each cluster, and C (**pldlimit**) be the maximum number of clusters allowed. To ensure that these constraints N and C are followed, some packing rules are checked before a candidate-pair is merged. That is, to ensure that no modules become larger than some size N, each candidate in the pair must therefore be of size $N/2$ or less. By similar reasoning, these candidates of size $N/2$ must be merged from other candidates of size $N/4$ or less, and so forth. To keep track of the size of the clusters, each cluster is assigned to one of B buckets, where B is found by:

$$B = 1 + \lfloor log_2 N \rfloor \tag{4.2}$$

Each bucket contains a number of clusters, all of which have a certain size S. The value of S associated with the ith bucket is found by:

$$\frac{N}{2^{B-i}} \geq S(i) > \frac{N}{2^{B-i+1}} \tag{4.3}$$

Given the bucket structure to keep track of the cluster sizes, the constraint C can be checked by calculating the total number of clusters that would form should each cluster

in a given bucket be merged with another in the same bucket and be assigned to the next larger bucket size. That is, clustering into C clusters is feasible if:

$$T^{t+1}(B) \leq C \tag{4.4}$$

$$T^{t+1}(i+1) = T^t(i+1) + \lceil T^t(i)/2 \rceil, \quad for\ i = 1, \ldots, B \tag{4.5}$$

$$T^t(1) = 0 \tag{4.6}$$

where $T^t(i)$ is defined as the number of clusters in bucket i before merging, and $T^{t+1}(i)$ is defined as the number of clusters in bucket i after merging. Note that feasibility can be established by a single pass through the bucket structure from equation 4.4. Note also that equation 4.4 is an approximation, since it does not take into account that some mergers may not be performed because of pinlimit restriction.

In the actual check for feasibilility, the "slack" or the difference between $S(i)$ and the size of a cluster in the ith bucket in equation 4.3 is taken into account, since the "slack" would allow for the merger of a large cluster with a small cluster.

Clustering File Format

As mentioned earlier, the clustering program called by the structure-processor *partition* reads and writes in a simple pinlist format. This offers the advantages of allowing users to make quick changes to the constraints to obtain quick results, allowing users to make manual changes including initial conditions, and providing a file format for future partitioning tools.

This pinlist, which is generated by *partition* and found under the directory **PARTITION**, represents the user's design after it has been flattened and assigned node and net id numbers.

There may be six sections to the pinlist file. The first section of the pinlist lists the node id number and the instance name associated with it. Node 0 represents the external world. The second section of the pinlist lists the net id number and the net name associated with it. Net 0 represents GND, and net 1 represents Vdd. Both sections one and two exist as comments.

The **CONSTRAINT** section contains the constraints **modulelimit**, **pldlimit**, and **pinlimit**, and is obtained from the formal parameters or from the defaults file. The

SIZE section indicates the initial size of each cluster. (Note that node 0 is assigned the modulelimit so that no node will become merged with the external world.) The **PINLIST** section is a list relating each node to each net attached to them. And lastly, the **CLUSTER** section indicates that certains nodes should be placed into the same cluster. The user may specify this section to ensure that certain nodes will occur within the same cluster. It is this section that is generated by the cluster algorithm, which in turn controls the restructuring of the oct database.

Restructured Facet

Given the cluster results, *partition* creates new masters and instances of these clusters within the oct database. The restructured facet, functionally the same as the users original design, has at most three levels of hierarchy: The leafcell level, the cluster level which contain leafcell instances, and the topmost level which connects up the clusters.

Automatic Layout-Generator Property

Partition also inserts the appropriate layout generator *prototype* into each of these clusters to facilitate the generation of target design files. If the layout-generator is specified at the topmost facet, then the clusters will inherit the same generator as the parent. Otherwise, the default generator *prototype* is used for each cluster.

4.3 Prototype

Design Flow

Figure 4.5 shows the flow for the layout-generator *prototype*. *Prototype* generates files representing the user's design entry into the target manufacturer's mapping tools. The oct facet should contain only instances from the target manufacturer's library. The files generated by *prototype* are placed under directory **PROTOTYPE**.

Formal Parameters

Table 4.3 shows the formal parameters used by *prototype*. *Prototype* supports two formal parameters, both of which are required. If the user had used the structure-processor

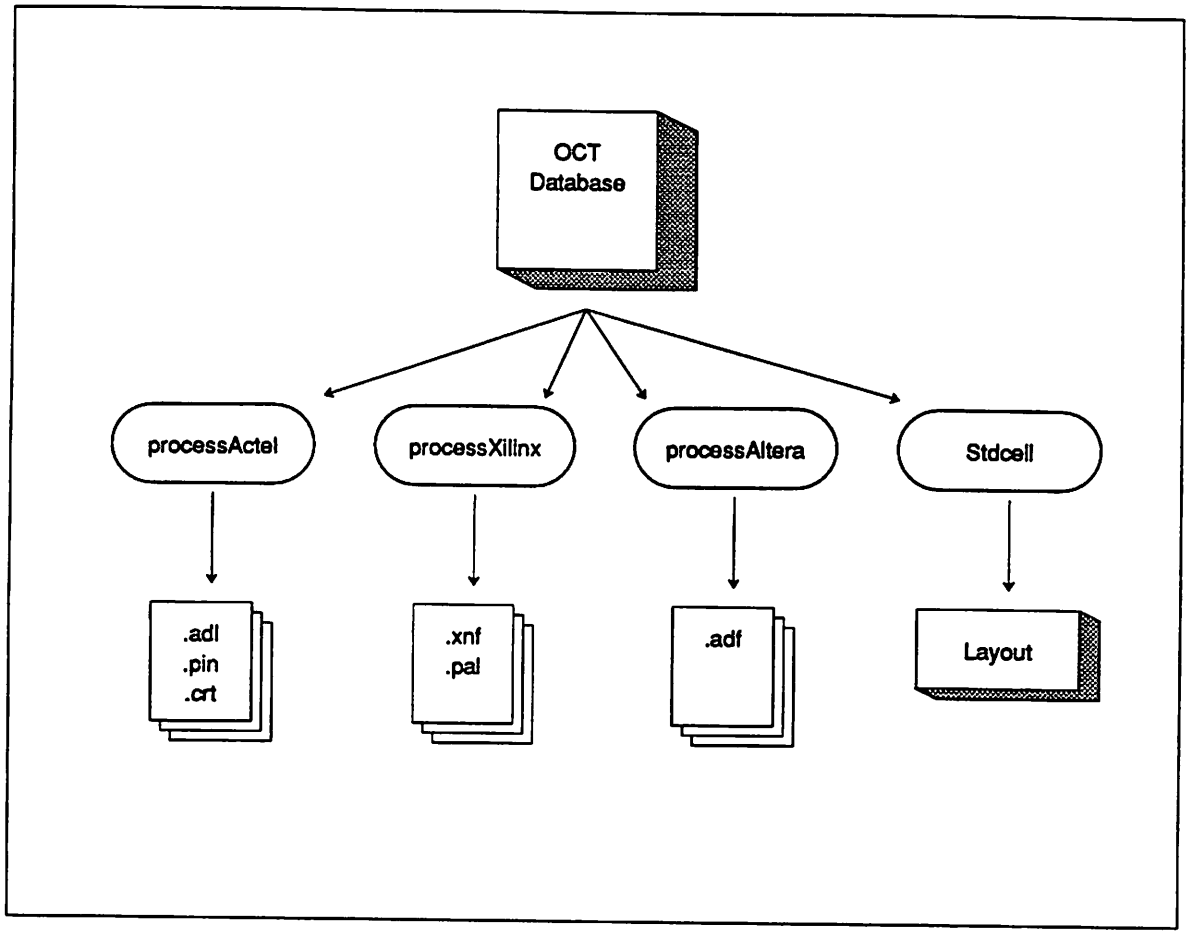

Figure 4.5: Design flow for layout-generator*prototype*.

No.	Formal Parameter	Usage	Default Value	Description
1	manufacturer	required	none	target manufacturer (ACTEL, XILINX, ALTERA, STDCELL, NONE)
2	partname	required	none	target device

Table 4.3: Formal Parameters Used by Prototype.

partition beforehand, then default formal parameters values would have been automatically introduced in the design.

The formal parameter **manufacturer** specifies the file format for the generated files. The formal parameter **partname** specifies which of the manufacturer's part to use.

Design Targets

Actel

In the case of **Actel**, *prototype* generates three files per cluster. The **.adl** file contains the netlist information, the **.pin** file contains the pin assignment, and the **.crt** file contains net criticality specifications. The **.adl** file is generated from the oct structure, and **.pin** file is generated using a simple pin assignment described below, and the **.crt** file is left empty.

Xilinx

In the case of Xilinx, *prototype* generates one or two files per clusters, depending on the mapping strategy used in *x2oct*. The **.xnf** file contains the netlist information, and the **.pal** file, in **PALASM** format, is also generated if the synthesis approach was used. The equations found in the **.pal** file comes from the **LOGICFUNCTION** property associated with the generic instances in the design. The pin assignment is also performed as described below.

Altera

In the case of **Altera**, *prototype* generates only one type of file per cluster. The **.adf** file contains the netlist information as well as any equations generated through the synthesis mapping approach in *x2oct*. Pin assignment is not done here since it is done automatically bye the **Altera** software.

Standard Cell

In the case of standard cell, *prototype* calls the layout-generator *Stdcell* to generate the layout.

Automatic I/O Buffer Insertion

Before any files are generated, *prototype* inserts any necessary I/O buffer cells at the boundaries of the newly created clusters. *Prototype* makes use of the **DIRECTION** and **TERMTYPE** properties associated with each terminal of the leafcells to determine the type of buffer to use. In general, users should include I/O buffers only at the toplevel of their designs or leave them out altogether and allow *prototype* to insert them automatically.

Automatic Pin Allocation

Associated with each part in the *PLDS* library is a list of pins available for use as either input or output. This allocation is performed starting from the top of the list. If the user wants to make particular pin assignment, then the file containing these assignments is used instead of the default.

Chapter 5

Evaluation

5.1 Examples

To examine the design flow of *PLDS*, I used a simple design example of an address decoder for a VME interface. (This decoder is an actual design for a speech recognition system.) The constraints were set to small values to check the performance of the clustering algorithm. Currently, the clustering algorithm will go as far as it can, given the constraints. In some cases, not all the clusters ended up the last bucket (the largest size) indicating that either the constraints were too stringent or that the resulting clusters were truly disjoint. In the case of disjoint clusters, the user may add a **CLUSTER** section in the pinlist to force the grouping.

5.2 Future Work

The *PLDS* system may be improved in the following areas. These improvements require only a minor amount of programming.

Partitioning Algorithms

The structure-processor *partition* was written so that it may support future partitioning algorithms. The pinlist format allows for developers to work with simple text files instead of with the oct database. Other approaches to the partitioning problem, including approaches using linear programming and simulated annealing, are certainly possible [5] [9].

Timing Constraints

Currently, circuit timing has not been considered in *PLDS*. Future improvements can be done through more accurate timing characterization of primitive cells in the *misII* library, leading to timing numbers of greater accuracy. A constraint on the critical path may be incorporated into the structure-processor *partition*, although this is not easily done with the current clustering algorithm.

Standard Cell Pads

Pads are not added to the clusters when the target implementation is full-custom standard cells.

Expand libraries

Although the primitive libraries are complete, the translation libraries may be expanded to include more commonly used macros.

PCB Interface

The netlist information at the topmost level after partitioning has been performed is useful for PCB (printed circuit board) tools such as that provided by Racal-Redac. This netlist shows how the clusters, now implemented in a *PLDS*, should be interconnected.

Backannotation Considerations

After a cluster has been successfully mapped into a target device, the pin assignment and part selected need to be back-annotated into the oct database so that future revisions, if any, can target the same device and footprint.

Target Comparisons

Some study into the mapping of different designs into different targets may be performed to determine the effectiveness and appropriateness of each target. Intuitively, PLA-like devices like those provided by **Altera** are appropriate for implementing controllers, logic arrays, and glue logic, whereas gate-array type devices like those provided by **Actel**

and **Xilinx** are appropriate for standard-cell and perhaps even datapath applications. This intuition requires further investigation.

Chapter 6

Conclusion

Prototyping in the Lager/Oct environment is possible through *PLDS*. This package consists of three structure-processors integrated with the Lager design methodology. *PLDS* maps and partitions designs using *misII* and a quick-clustering algorithm. *PLDS* was developed in an expandable and modular fashion, and makes use of translation libraries that hide many details from the user, allowing for simple retargetting. Future manufacturers can be supported by creating a library to represent their primitive gates and another to represent the translation from the particular library to a common library. Other partitioning algorithms can be easily included by adhering to the simple pinlist format. Given the *PLDS* framework, prototyping may become truly rapid.

Bibliography

[1] A. E. Gamal et al. "An Architecture For Electrically Configurable Gate Array,". *JSSC*, 24(2):394–398, 1989.

[2] S. B. Akers. "Clustering Technique For VLSI,". In *Design Automation Conference*, 1982.

[3] B. W. Kernighan, S. Lin. "An Efficient Heuristic Procedure For Partitioning Graphs,". *Bell Sys. Tech. J.*, 49:291–307, 1970.

[4] D. M. Schuler, E. G. Ulrich. "Clustering and Linear Placement,". In *Design Automation Conference*, 1975.

[5] E. R. Barnes. "An Algorithm For Partitioning The Nodes Of A Graph,". *SIAM J. Alg. Disc. Meth.*, 3(4), 1982.

[6] Electronic Research Laboratory, University of California, Berkeley. *LagerIV Silicon Assembly System Manual*, Distribution 1.0 edition, October 1985.

[7] H. C. Hsieh et al. "A 9000-gate User-Programmable Gate Array,". In *CICC*, pages 15.3.1–15.3.7, 1988.

[8] K. A. El-Ayat et al. "A CMOS Electrically Configurable Gate Array,". *JSSC*, 24(3):752–762, 1989.

[9] M. Beardslee et. al. "SLIP: A Software Environment For System Level Interactive Partitioning,". In *ICCAD*, pages 280–283, 1989.

[10] S. C. Wong et al. "A 5000-gate CMOS EPLD With Multiple Logic and Interconnect Arrays,". In *CICC*, pages 5.8.1–5.8.4, 1989.

[11] W. S. Carter et al. "A User-Programmable Reconfigurable Logic Array,". In *CICC*, pages 233–235, 1986.

Appendix A

PLDS Policy

The three oct policies used by the PLDS system are based on the Lager/Oct Symbolic Policy and differ in additional properties and bags. The details of these differences are presented here. They are the Top Cell Policy, the Cluster Cell Policy, and the Leaf Cell Policy.

A.1 Top Cell Policy

The Top Cell Policy introduces three additional properties and two additional bags to the Symbolic Policy. These additions are related to the partitioning of the topcell. The NODEID and NETID properties are integer-valued properties attached to the instances and nets, respectively. They are used to generate the pinlist for the clustering program. The NODEID and NETID bags contain these properties as well. The CUT property is a single property that is contained by all the nets that are cut as a result of the partitioning.

A.2 Cluster Cell Policy

The Cluster Cell Policy, used by the cluster facets one level below the topcell, introduces five new properties and two new bags to the Symbolic Policy. These additions are related to the generation of design files after the partitioning has completed. The INPUT and OUTPUT properties are contained by all the formal terminals that are either inputs or outputs to the cluster facet. There are only one of each property contained by the terminals. Each formal terminal and net may also have an name alias property called NEWTERM and

NEWNET, respectively. These properties are also contained by the NEW_TERM_NAME and NEW_NET_NAME bags, respectively.

In the case of a technology mapping using the SYNTHESIS approach towards Xilinx and Altera, the mapped instances contained by the facet are from the "generic" library. The logic function associated with each generic instance is found by the LOGICFUNCTION property attached to the instance. This string-valued property is a logic equation in pre-fix notation relating the generic output to the generic input(s).

A.3 Leaf Level Policy

The Leaf Cell Policy adds four new properties to the Symbolic Policy. In addition to the usual DIRECTION and TERMPLACE properties attached to the formal terminals, PLDS also requires a PINNUMBER property. This integer- valued property is used to determine the order of formal terminals in the file format used by the target manufacturer.

The FORMAL_PARAMETES bag has added CELLNAME, CELLCOUNT, and CELLTYPE to the usual CELLCLASS property. The CELLNAME is needed in the cases where the facet name cannot be the same as the name of the primitive it represents. (This occurs when the primitive name begins with a numeric character.) The CELLCOUNT property indicates the number of elements required to implement the leafcell using the target manufacturer's devices. The CELLTYPE property is required only by Altera. This property takes on the values "MACRO", "LOGIC", "OBUFFER", or "IBUFFER" to indicate that the leafcell represents a macrocell, a logic primitive, an output structure, or an input structure, respectively.

Appendix B

Defaults

B.1 Partname

```
;
; Default partname to use for each manufacturer.
; Search is done using grep on manufacturer to get
; partname.  Manufacturer must therefore be unique.
;
; Format:
; manufacturer partname
;

ACTEL ACT1020-PL84
XILINX XC3090
ALTERA EP1810
STDCELL STDCELL
NONE NONE
```

B.2 Limits

```
;
; Default modulelimit and pinlimit for each partname.
; Search is done using grep for partname to get associated
```

```
; modulelimit and pinlimit. The partname entries must therefore
; be unique.
;
; Format:
; manufacture partname modulelimit pinlimit
;

ACTEL ACT1010-PL44 295 34
ACTEL ACT1010-JQ44 295 34
ACTEL ACT1010-PL68 295 57
ACTEL ACT1010-JQ68 295 57
ACTEL ACT1010-PG84 295 57

ACTEL ACT1020-PL44 546 34
ACTEL ACT1020-JQ44 546 34
ACTEL ACT1020-PL68 546 57
ACTEL ACT1020-JQ68 546 57
ACTEL ACT1020-PL84 546 69
ACTEL ACT1020-JQ84 546 69
ACTEL ACT1020-PG84 546 69

XILINX XC3020 64 64
XILINX XC3030 100 80
XILINX XC3042 144 96
XILINX XC3064 224 120
XILINX XC3090 320 144

ALTERA EPM5016 16 20
ALTERA EPM5024 24 24
ALTERA EPM5032 32 28
ALTERA EPM5064 64 44
ALTERA EPM5127 128 44
ALTERA EPM5128 128 68
```

```
ALTERA EPM5130 128 100
ALTERA EPM5192 192 84

ALTERA EP1800 48 64
ALTERA EP1810 48 64
ALTERA EP900 24 36
ALTERA EP910 24 36
ALTERA EP600 16 20
ALTERA EP610 16 20
ALTERA EP320 8 18
ALTERA EP310 8 18
ALTERA AUTO 48 64

STDCELL STDCELL 1000 1000

NONE NONE 1000 1000
```

B.3 Pinlists

Associated with each target device is a file referred to by the "device-package" name which lists all the pins available for use as inputs or outputs. For example, the file ACT1010-JQ44 contains:

```
BIPUTS

 1  2  4  5  6  7  8  9 11 12
13 15 17 18 19 20 22 23 24 26
27 28 29 30 31 33 36 37 38 39
40 41 42 44
```

Appendix C

MisII Scripts

C.1 Actel

ACTEL.DECOMP.AREA

Note: Mapping using decomposition while minimizing area.

```
set autoexec ps
set OCT-CELL-VIEW structure_instance
set OCT-CELL-PATH LIBPATH/actel.xlat.lib/actel.lib
set OCT-TECHNOLOGY scmos
set OCT-EDITSTYLE SYMBOLIC
set OCT-VIEWTYPE SYMBOLIC
read_library LIBPATH/actel.xlat.lib/mis.lib/actel.lib.mis
collapse
source LIBPATH/scripts/script
map -m MAP_FACTOR -s
phase -g
write_bdnet
```

ACTEL.DECOMP.DELAY

Note: Mapping using decomposition while minimizing delay. This is done by collapsing the network before mapping.

```
set autoexec ps
set OCT-CELL-VIEW structure_instance
set OCT-CELL-PATH LIBPATH/actel.xlat.lib/actel.lib
set OCT-TECHNOLOGY scmos
set OCT-EDITSTYLE SYMBOLIC
set OCT-VIEWTYPE SYMBOLIC
read_library LIBPATH/actel.xlat.lib/mis.lib/actel.lib.mis
collapse
speed_up -d SPEED_UP
map -m MAP_FACTOR -s
phase -g
write_bdnet
```

ACTEL.SYNTHESIS

Note: Mapping using synthesis (hard-coded misII) routines.

```
set autoexec ps
collapse
source LIBPATH/scripts/script
act_map -h HEURISTIC_NUM -n NUM_ITERATIONS -f COLLAPSE_FANIN \
    -g GAIN_FACTOR -d DECOMP_FANIN
act_map -h HEURISTIC_NUM -r BDNET_FILE
```

C.2 Altera

ALTERA.DECOMP.AREA

Note: Mapping using decomposition while minimizing area.

```
set autoexec ps
set OCT-CELL-VIEW structure_instance
set OCT-CELL-PATH LIBPATH/altera.xlat.lib/altera.lib
set OCT-TECHNOLOGY scmos
set OCT-EDITSTYLE SYMBOLIC
```

```
set OCT-VIEWTYPE SYMBOLIC
read_library LIBPATH/altera.xlat.lib/mis.lib/altera.lib.mis
collapse
source LIBPATH/scripts/script
map -m MAP_FACTOR -s
phase -g
write_bdnet
```

ALTERA.DECOMP.DELAY

```
set autoexec ps
set OCT-CELL-VIEW structure_instance
set OCT-CELL-PATH LIBPATH/altera.xlat.lib/altera.lib
set OCT-TECHNOLOGY scmos
set OCT-EDITSTYLE SYMBOLIC
set OCT-VIEWTYPE SYMBOLIC
read_library LIBPATH/altera.xlat.lib/mis.lib/altera.lib.mis
collapse
speed_up -d SPEED_UP
map -m MAP_FACTOR -s
phase -g
write_bdnet
```

ALTERA.SYNTHESIS

Note: Mapping using synthesis while minimizing delay. This is done by collapsing the network, followed by tech_decomp into AND and OR gates with constraints on the fanins. eqn2oct used to store equations with generic instances.

```
set autoexec ps
set OCT-CELL-VIEW structure_instance
set OCT-CELL-PATH LIBPATH/altera.xlat.lib/altera.lib
set OCT-TECHNOLOGY scmos
set OCT-EDITSTYLE SYMBOLIC
set OCT-VIEWTYPE SYMBOLIC
```

```
read_library LIBPATH/altera.xlat.lib/mis.lib/altera.lib.mis
collapse
source LIBPATH/scripts/script
tech_decomp -a AND_FANIN -o OR_FANIN
write_eqn EQN_FILE
```

C.3 Xilinx

XILINX.DECOMP.AREA

```
set autoexec ps
set OCT-CELL-VIEW structure_instance
set OCT-CELL-PATH LIBPATH/xilinx.xlat.lib/xilinx.lib
set OCT-TECHNOLOGY scmos
set OCT-EDITSTYLE SYMBOLIC
set OCT-VIEWTYPE SYMBOLIC
read_library LIBPATH/xilinx.xlat.lib/mis.lib/xilinx.lib.mis
collapse
source LIBPATH/scripts/script
map -m MAP_FACTOR -s
phase -g
write_bdnet
```

XILINX.DECOMP.DELAY

```
set autoexec ps
set OCT-CELL-VIEW structure_instance
set OCT-CELL-PATH LIBPATH/xilinx.xlat.lib/xilinx.lib
set OCT-TECHNOLOGY scmos
set OCT-EDITSTYLE SYMBOLIC
set OCT-VIEWTYPE SYMBOLIC
read_library LIBPATH/xilinx.xlat.lib/mis.lib/xilinx.lib.mis
collapse
speed_up -d SPEED_UP
```

```
map -m MAP_FACTOR -s
phase -g
write_bdnet
```

XILINX.SYNTHESIS

Note: Mapping using synthesis (hard-coded misII) routines.

```
set autoexec ps
set OCT-CELL-VIEW structure_instance
set OCT-CELL-PATH LIBPATH/xilinx.xlat.lib/xilinx.lib
set OCT-TECHNOLOGY scmos
set OCT-EDITSTYLE SYMBOLIC
set OCT-VIEWTYPE SYMBOLIC
collapse
source LIBPATH/scripts/script
xl_split -n XL_FANIN
sweep
simplify
xl_partition -n XL_FANIN
sweep
simplify
xl_partition -n XL_FANIN
sweep
xl_k_decomp -n XL_FANIN
sweep
xl_cover -h XL_COVER -n XL_FANIN
time
xl_cover -h XL_COVER -n XL_FANIN
time
xl_merge
time
write_eqn EQN_FILE
```

C.4 Stdcell

Standard script for mapping into MSU Stdcell library.

script.msu

```
set autoexec ps
set OCT-CELL-VIEW structure_instance
set OCT-CELL-PATH LIBPATH/stdcell.xlat.lib/stdcell.lib
set OCT-TECHNOLOGY scmos
set OCT-EDITSTYLE SYMBOLIC
set OCT-VIEWTYPE SYMBOLIC
read_library LIBPATH/stdcell.xlat.lib/mis.lib/stdcell2_2.genlib
collapse
source LIBPATH/scripts/script
map -m MAP_FACTOR -s
phase -g
write_bdnet
```

C.5 General

Note: Standard script for multilevel logic optimization.

script

```
sweep; eliminate -1
simplify
eliminate -1
simplify
resub -a
gkx -abt 30
resub -a; sweep
gcx -bt 30
resub -a; sweep
gkx -abt 10
```

```
resub -a; sweep
gcx -bt 10
resub -a; sweep
gkx -ab
resub -a; sweep
gcx -b
resub -a; sweep
eliminate 0
decomp -g *
eliminate -1; sweep
```

Appendix D

X2oct

```
Usage: x2oct [-v] [-L logfile] [-T tempdir] cell[:view]
        -v:             verbose
        -L:             logfile
                        (default: x2oct.log)
        -T:             temp directory
                        (default: /usr/tmp)
        cell:           input facet
                        (default: structure_instance)
```

X2oct is a csh script that makes use of several custom programs. These programs are described below.

D.1 x2octparam

```
Usage: x2octparam [-c char] [-a] [-L logfile] -o sed_outfile cell[:view]
       -c:              char delimiting parameter name
                        (default: '@')
       -a:              get parameters from parent
                        smv ACTUAL_PARAMETERS bag instead
       -L:              logfile name
                        (default: stderr)
       -o:              sed output filename
```

```
cell[:view]:      input facet to extract
                  (default view: "structure_instance")
```

X2octparam extracts parameters from either the **FORMAL_PARAMETERS** bag or the **ACTUAL_PARAMETERS** bag, depending on whether the template *xpld* subcell is used or not. *X2octparam* uses the leading "@" character of a parameter to denote a variable in the **.bds** file, and creates a file for the string editor utility *sed* to perform the actual string substitution in the **.bds** file.

D.2 myMisII and fixbdnet

The current version of *misII* contained a few errors in the bdnet file generated in the case of mapping to Actel. *MyMisII* is a modified verion which fixes some of these errors. Other errors which I was not able to fix within *myMisII* were corrected by using an awk script *fixbdnet*.

D.3 myNle

The current verion of the *nle* package provided with the *octtools* distribution used the "." character in some of the net and instance names generated. This character often causes problems for some of the tools provided by the manufacturers. To overcome this problem, *myNle* simply uses the "_" character instead.

D.4 fixpla

```
Usage: fixpla [-g] -i pla_input -o sed_output [-L logfile] cell[:view]
       -g:              specifies generic 'xpld' used
                        (default: not used)
       -i:              source pla filename
       -o:              sed command filename
       -L:              logfile name
                        (default: stderr)
       cell[:view]:     reference facet
                        (default view: "structure_instance")
```

The Lager/Oct Symbolic policy expects brackets "[" and "]"to denote indexed signals within a bus. *Fixpla* changes any angle brackets "¡" and "¿" found in the **.pla** description before the description is sent to *misII* for mapping. Also, the **.pla** description is compared to the reference facet for terminal name consistency. If the template **xpld** is used, then the signals are mapped to "IN[i]" and "OUT[j]".

D.5 postSynthesis

```
Usage: postSynthesis [-L logfile] cell[:view]
       -L:           logfile name
                     (default: stderr)
       cell[:view]: input facet name
                     (default view: "structure_instance")
```

Postsynthesis is used in the cases of mapping towards Altera and Xilinx using the **SYNTHESIS** method. *Postsynthesis* attaches a property **LOGICFUNCTION** to each of the generic instances specified by *misII* after it has mapped the design. Also, each generic instance in the design is replaced by a generic cell from the **x2oct.lib**.

D.6 mis2siv

```
Usage: mis2siv -l lager_cell[:view] [-L logfile] mis_cell[:view]
       -l:              source lager facet name
                        (default view: "structure_instance")
       -L:              logfile name
                        (default: stderr)
       mis_cell[:view]: mis facet to modify
                        (default view: "structure_instance")
```

Mis2siv checks the newly created facet generated by *misII* with the original facet it is to replace. *Mis2siv* creates any terminals not found in the newly created facet, connects up Vdd and GND nets to the Vdd and GND terminals, if any, and attaches the NETTYPE property to these nets, if found.

Appendix E

Partition

```
Usage: partition [-v] [-d directory] [-L logfile] cell[:view]
        -v:             verbose
        -d:             directory to store output design files
                        (default: PARTITION)
        -L:             logfile name
                        (default: partition.log)
        cell:           input facet name
                        (default view: structure_instance)
```

The structure-processor *partition* is a csh script which makes use of several custom programs. These programs are described below.

E.1 makePinlist

```
Usage: makePinlist [-f cell[:view]] [-L logfile] [-o outfile] cell[:view]
      -f:           flattened output facet name
                    (default: "cell:flat")
      -L:           logfile name
                    (default: stderr)
      -o:           output pinlist file name
                    (default: "cell.pinlist")
      cell[:view]: input facet name
                    (default view: "structure_instance")
```

MakePinlist is the pre-processor for the partitioning program *cluster* and performs three functions. First, *makePinlist* flattens the design to the leafcell level. Second, *makePinlist* assigns NODEID and NETID numbers to each instance and net found in the flattened design. Third, *makePinlist* creates a pinlist file from the design using the assigned instance and net numbers.

E.2 cluster

```
Usage: cluster [-clstvw] [-L logfile] [-o output] input
       -c:     cleanup disjoint clusters at end
       -l:     print internal lists
       -s:     summary
       -t:     traverse clusters showing tree
       -v:     verbose
       -w:     turn ON any warnings
       -L:     logfile name
               (default: stderr)
       -o:     output pinlist filename
               (default: input.cluster)
       input:  input pinlist filename
```

Cluster is a partitioning program based on a quick-clustering algorithm. The input is a pinlist generated by *makePinlist*, and the output is a pinlist which shows the resultant partitioning. See section 4 for details of the algorithm.

E.3 makePartition

```
Usage: makePartition [-r cell[:view]] [-L logfile] -p pinlist cell[:view]
       -r:           restructured output facet name
                     (default view: restructure)
       -L:           logfile name
                     (default: stderr)
       -p:           input pinlist file name
       cell[:view]:  input facet name
```

```
(default view: "flat")
```

MakePartition is the post-processor to the partitioning program. It takes the resultant pinlist and the flattened Oct facet, and restructures the design by grouping each cluster into a new cell (or master). Each newly created cell is named ¡cell¿_cluster.i, where i refers to the cluster ID number assigned by the clustering program. *MakePartition* then creates a **CUT** property in the Oct cell and attaches all the nets cut to this property. *MakePartition* also copies over the **manufacturer** and **partname** properties from the topcell to the newly created cells.

E.4 insertLayGen

```
Usage: insertLayGen [-f] [-L logfile] cell[:view]
       -f:          insert layout generator at parent only
                    (default: layout generator inserted
                    at level below)
       -L:          logfile name
                    (default: stderr)
       cell[:view]: input facet name
                    (default view: "structure_instance")
```

InsertLayGen is a simple program that inserts the **layout-generator** property into the Oct design database. If the design is flat, then the default layout-generator inserted is "prototype". However, if the design is not flat (i.e. some partitioning has been performed), then the topcell is given the **layout-generator** "NONE", and the cluster cells are given the **layout-generator** "prototype".

Appendix F

Prototype

```
Usage: prototype [-v] [-m] [-p pinfile] [-d directory] [-L logfile] cell[:view]
        -v:             verbose
        -m:             generate magic files, where applicable
        -p:             pin assignment file
                        (default: depends on partname)
        -d:             directory to store output design files
                        (default: PROTOTYPE)
        -L:             logfile name
                        (default: prototype.log)
        cell:           input facet name
                        (default view: structure_instance)
```

The layout-generator *prototype* is another csh script that makes use of custrom programs. These programs are described here.

F.1 processActel

```
Usage: processActel [-h] [-p pinfile] [-d directory] [-L logfile] cell[:view]
       -h:          generate files for hierarchical cell
                    (default: input cell is flat)
       -p:          input file to read allowed pin numbers
                    (default: ordered assignment)
```

```
-d:           directory to put output files
              (default: "ACTEL")
-L:           logfile name
              (default: stderr)
cell[:view]: input facet name
              (default view: "structure_instance")
```

ProcessActel generates the **.adl, .crt, and .pin** files needed by the Actel software. *ProcessActel* uses the pin any number assignment specified and inserts the input and output buffers at the cluster boundaries. The INPUT and OUTPUT properties are also attached to the corresponding nets.

F.2 processXilinx

```
Usage: processXilinx [-h] [-p pinfile] [-d directory] [-L logfile] cell[:view]
       -h:           generate files for hierarchical cell
                     (default: input cell is flat)
       -p:           input file to read allowed pin numbers
                     (default: ordered assignment)
       -d:           directory to put output files
                     (default: XILINX")
       -L:           logfile name
                     (default: stderr)
       cell[:view]: input facet name
                     (default view: "structure_instance")
```

ProcessXilinx is similar to *processActel* except it generates the **.xnf and .pal** files needed by the Xilinx software. *ProcessXilinx* generates the equations in the **.pal** file using the LOGICFUNCTION properties attached to the generic cells.

F.3 processAltera

```
Usage: processAltera [-h] [-d directory] [-L logfile] cell[:view]
       -h:           generate files for hierarchical cell
```

```
                (default: input cell is flat)
-d:             directory to put output files
                (default: "ALTERA")
-L:             logfile name
                (default: stderr)
cell[:view]: input facet name
                (default view: "structure_instance")
```

ProcessAltera is similar to *processXilinx* except it generates the **.adf** file needed by the Altera software. *ProcessAltera* generates the equations using the LOGICFUNCTION properties attached to the generic cells, but these equations are placed in the same **.adf** file under the EQUATIONS section. *ProcessAltera* also assigns new net and terminal names since the Altera has a limit of eight characters in each name.

Appendix G

Xpld Template

```
;
; leafcell for pld prototyping
;
(parent-cell xpld)
(parameters
inwidth
outwidth
(generic "")
(bds "")
(blif "")
(pla "")
(eqn "")
(manufacturer "")
(mapping "")
(script "")
(minimize "")
(map_factor "")
(speed_up "")
(or_fanin "")
(and_fanin "")
(heuristic_num "")
(num_iterations "")
```

```
(collapse_fanin "")
(gain_factor "")
(decomp_fanin "")
)
;
(structure-processor x2oct)
;
(instance parent (
    ((terminal IN ) IN  (width inwidth ))
    ((terminal OUT) OUT (width outwidth))

    ; add Vdd and GND terminals only for ACTEL SYNTHESIS approach
    ; to allow for Vdd, GND ties within the BASIC_BLOCK to parent cell
    ((terminal Vdd) Vdd
     (conditional (and (equal mapping "SYNTHESIS")
       (equal manufacturer "ACTEL"))))
    ((terminal GND) GND
     (conditional (and (equal mapping "SYNTHESIS")
       (equal manufacturer "ACTEL"))))
    ))
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
;
(end-sdl)
```