Inter-Office Memorandum

To	Chipmonk Users	Date	March 18, 1982
From	Rich Pasco	Location	Palo Alto
Subject	Nuts Manual	Organization	PARC VLSI Systems Design

XEROX

Filed on: [Indigo]<DA>Nuts>NutsManual.Bravo, .Press

Nuts are what Chipmonks eat.

Nuts.bcd is a D-machine program which translates integrated circuit layout data files from Caltech Intermediate Form (CIF) to Chipmonk Format. It runs on D-Machines under Alto/Mesa 6.0.

It accepts a limited subset of CIF commands, specifically:

Define Symbol
Box
Call Symbol
Mirror, Translate
Rotate (multiples of 90 degrees only)
UserExtension 9 = symbol names (compatible with SIF).
UserExtension 94 = node names.

Symbol hierarchy is preserved, although symbol definitions are re-ordered to eliminate forward references.

To use, retrieve [Indigo]<DA>Nuts>Nuts.bcd and type "Nuts<CR>" to your Alto Executive. A typical Mesa.typescript follows:

```
Alto/Mesa 6.0 of 13-Oct-80 11:47
13-Jan-82  2:10
>Nuts -- 146500B
NUTS of January 13, 1982
File Name (no extension): Example
Lambda (CIF units): 250
Initializing Parser, Interpreter
Parsing Example.cif...100...200...finished.
Writing Example.chip...finished.
```

Nuts is intended to translate any syntactically valid .CIF file into a valid .Chip file, with all exceptions (constructs Chipmonk cannot handle) flagged with an error message describing the difficulty and the fix taken.

Examples of exceptions and their fixes are:

Lambda = odd number of CIF units - not allowed, aborted
[Chipmonk internal dimensions are in terms of half-lambda]
Items not aligned on lambda grid - dimensions quantized
CIF dimensions too big to represent in Chipmonk's 16-bit-integers - set to 0
45-degree (and worse) boxes - deleted
User extension 94 = node names - placed on zero-by-zero metal rectangle (see below)
Undefined user commands - ignored
RoundFlash, Wire, Polygon - deleted

To answer in advance the most commonly asked questions:

Q. How does Nuts recognize transistors, contacts, etc?

A. It doesn't. Transistors are translated as red crossing green. If you want to use the Chipmonk transistor and contact primitives, you'll need to edit them in by hand.

Q. What about cut, implant, etc? I thought there was no way to draw rectangles on these layers in Chipmonk.

A. Right, there isn't any way to *draw* them in Chipmonk. But there is a way to *represent* them, and Nuts uses that.

It is interesting to note that contacts translated from CIF appear differently on a color display than contact primitives drawn in Chipmonk. Apparantly when drawing a contact primitive the black of the cut *replaces* the color of the other layers, while when drawing a naked contact superimposed on other layers its black is *added* to the color map by some other algorithm. This should help the designer visually locate such contacts.

Q. What about CMOS layers?

A. The current Nuts won't handle them; they were just recently added to Chipmonk. But Nuts has provisions for them, and they will be added once standardized CIF names for them are agreed upon.

Q. Will nuts be expanded to handle 45 degree angles?

A. It would be medium-easy, but there's no point, until Chipmonk can handle them. The Chipmonk file format allows 45's, but Chipmonk itself can't presently handle them.

Q. Will red-crossing-green transistors, naked cuts, etc. be flagged?

A. Regarding red-crossing-green transistors, naked cuts, etc. it is not really in my immediate plans to detect and flag these in Nuts, as they are structures which can be literally translated and will live happily in the Chipmonk environment. (You could even draw a red-crossing-green transistor in Chipmonk!) Indeed, for Nuts to replace red-crossing-green with a call on a Transistor primitive would require second-guessing the designer's intent, and would seriously complicate the present clean, simple code of Nuts. There are planned features of Chipmonk which will not properly handle them (e.g. circuit extractor). It is my feeling that all red-crossing-green transistors, naked cuts, etc., both those translated from CIF and those drawn in Chipmonk, should be detected and flagged within Chipmonk during Chipmonk's internal design-rule-checking phase.

Node Names. The next item on the Nuts development plan is node names. The best way to handle them is not clear. In Icarus, text items may be placed anywhere, but in Chipmonk, text must be a property of an underlying wire item (box on metal, poly, or dif). The Icarus designer may place a text item in the top level to label a node in a symbol nested several layers deep. The proposed approach is to find which layers lay under the loose text item. If there is exactly one, a one-lambda square of that layer is generated to hold the text. Otherwise, the text is flagged and ignored. Comments on this approach are solicited.

As an interim hack, Nuts now represents all node names as a text property of a zero-by-zero piece of Metal in the right place (and flags them appropriately). If the desired node was on some other layer, it is necessarily to manually edit the appropriate cell, but this should be easier than starting from scratch with node names.

Architecture / How it Works

Nuts is built around the same CIFParser and Interpreter used by Magic. The communication goes something like this:

Nuts is in control. There are two phases of operation. During the first phase CIF Parser and Interpreter build an internal representation of the design in the Interpreter's virtual memory. In the second phase, Nuts writes out the Chip file, interrogating the Interpreter to obtain the necessary information.

During the CIFParsing phase, Nuts calls ParserDefs.ParseStatement repeatedly. The parser returns to Nuts after each CIF statement, and Nuts increments the statement count and displays it as appropriate. When the Parser finds a User Command, it calls OutputDefs.OutputUserCommand. Nuts then parses the User Command, allocates a UserObject of the appropriate size and type, and passes it to the Interpreter (for inclusion in the Interpreter's data base) by calling IntDefs.IUserObject. After parsing each (non-User) statement, the Parser hands it (directly) to the Interpreter, which calls OutputDefs.BB* (where * is one of Wire, Flash, Polygon, Box, or UserObject). Nuts responds to these calls by returning the appropriate bounding box. Also, constructs not handled by Nuts (such as non-manhattan boxes) are flagged at this point.

Chipmonk files are so constructed that before each list of items there is a count of how many items follow. [See [Indigo]<DA>Nuts>ChipFileFormat.press]. In order to write out a chip file, multiple passes are made over the Interpreter data base, first to count and later to actually output the objects. In addition, Chipmonk requires that there be no forward symbol references, a constraint not imposed by CIF. To meet this constraint, Nuts maintains a dictionary of symbols in such an order that there are no forward references.

The first pass over the interpreter's data base simply counts the symbols and allocates the dictionary. The second pass enters the symbols in the dictionary in an order such that each symbol appears before it is called. This is achieved by the recursive procedure Mark, which enters a symbol in the dictionary only after first entering all the symbols it calls.

The contents of a symbol definition held by the Interpreter are examined by calling AuxIntDefs.IExpand. The Interpreter responds by calling AuxOutputDefs.Aux* for each object in the definition, where * is one of Wire, Flash, Polygon, Box, UserObject, or Call. How Nuts responds to these calls depends on its state Mode, which may be one of Idle, Marking, ScanningSymbol, CountingItems, PuttingDefn, or PuttingItems.

In Marking mode, Nuts ignores all items except Calls.

Once forward references have been resolved, Nuts may write out the definitions. For each symbol (in dictionary order), mode ScanningSymbol is invoked and the symbol definition is expanded, to search for the symbol name (user extension 9 anywhere in the definition) and to count the items inside. Then mode PuttingDefn is invoked and the symbol is again expanded, this time with the objects being written to the Chip file. This is repeated until all symbol definitions have been written.

Two more modes, CountingItems and PuttingItems, apply to top-level items. These correspond to ScanningSymbol and PuttingDefn except that instead of a real symbol the pseudo-symbol obtained by IGetRootID is used to signal the interpreter to use top-level items.