A Constrained Design Methodology for VLSI

Mike Tucker, Hewlett-Packard Company
Lou Scheffer, Valid Logic Systems, Inc.

Hierarchical **CAD** tools hold promise for effectively managing the design of custom VLSI circuits. However, the implementation of these tools has suffered because of the common design practice of overlapping cells. Overlapping cells cannot be checked separately; they must be merged together before checking. This merging removes all the advantages of the hierarchical format. **A** solution used by some artwork checking systems (Whitney, 1981), is to take advantage of the hierarchy where it is present, but check overlapping cells in the usual way. **A** more effective approach is to preserve the hierarchy throughout the design process by severely limiting the amount of cell overlap. This can be done with no loss of chip area, and allows for much more effective verification techniques.

Cell overlap in IC design is like GOTOs in programming languages. These features are bad not only because they violate hierarchical design principles, but also because they impede verification and make human understanding difficult. However, overlaps (and GOTOs) are useful in some situations so it makes sense to limit the use of these features to cases that are easy to understand and verify.

An "Ideal" Methodology

Suppose we design a hierarchical IC layout so that no two cells overlap, and so that no primitives overlap the cells. This approach' provides several advantages:

1. Most operations can be done one cell at a time. These operations include design-rule checking (DRC), component extraction, cross-comparison between the schematic and ihe layout, and generation of new layers. This is a result of the no-overlap rule, which ensures that all instances of a given cell are identical.
2. Each cell may be verified when it is created. This feature distributes the verification effort throughout the design cycle. Errors are caught when they are easy to fix, rather than late in the design cycle, when changes are difficult.
3. Error reporting is much better. Because each cell is checked only once, multiple uses or "instances" of a cell which contains an error do not produce multiple diagnostics. Operations such as circuit-to-artwork comparison can pinpoint the locations of errors to a single cell.
4. Performance is greatly improved. The time it takes most **DRC** algorithms to run increases exponentially with the number of items checked. If overlap is not allowed, however, the time required to check all cells and cell interactions increases linearly (Scheffer, 1982). Furthermore, each cell need only be checked once, regardless of how many times it is used.
5. Changes in a design are much easier to check. Suppose a single cell is changed, and then verified by itself. If the boundary of the cell has not changed, then no higher-level cells need be checked. The lack of overlap means that a change to artwork that is not located on the edges of a cell does not affect any other cell. On the other hand, if overlap is allowed, then change in any cell will require rechecking the whole design.

Unfortunately, almost all of these advantages rely on a complete lack of overlap. For various reasons, however, most existing designs do have some overlap. This overlap may exist for innocuous reasons, such as the designers' desire to share buses, or because of a limited artwork system that only allows rectangular cells. Often, though, overlapping cells are a violation of hierarchical design principles; such cells often exist because there was no tangible advantage in maintaining a truly hierarchical design. Therefore, an ideal design methodology would accept "harmless" overlaps, but prohibit extensive violations of the hierarchy. This approach would preserve the advantages listed above, while causing minimum inconvenience to the designer.

Generalization Attempts

HP design groups have objected to a complete ban on overlaps primarily because of shared buses. In shared busing, two cells, each containing a single line adjacent to the cell boundary, are overlapped so that the lines coincide (Figure 1). The no-overlap constraint causes designers to include half of the bus width in each cell, thus creating two problems:

- Due to the half-width bus, neither cell definition honors the design rules;
- If either cell is instanced by itself, a half-width bus, adjacent to the cell, must be added manually.

Unfortunately, these problems cannot be dismissed on grounds of rarity; the use of shared buses to save area is far too common. Two possible solutions are:

1. Do not include shared buses in the cells which use them; put them outside the cells instead.
2. Let geometry overlap the cell boundary.

The first solution is the less desirable one from a designer's viewpoint, because it does not let transistors in the cell be put

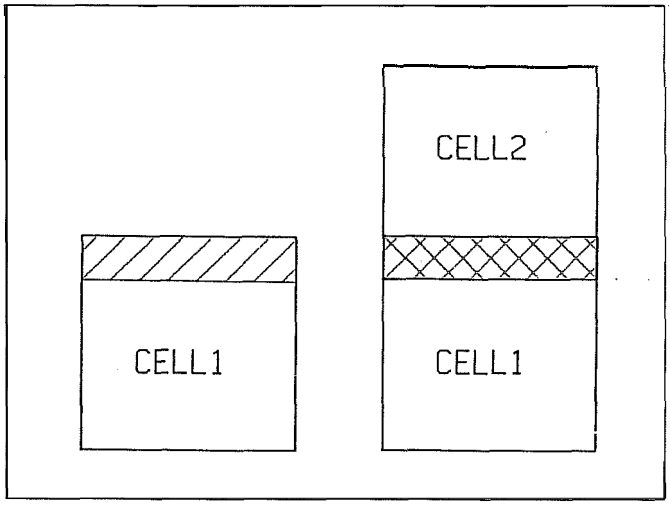


FIGURE 1. Shared bus.

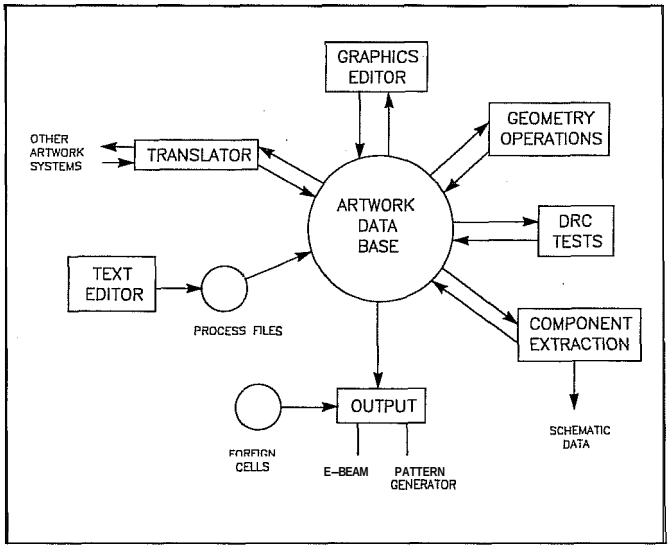


FIGURE 2. Artwork data flow.

under the bus. This problem is particularly acute in advanced double-layer metal MOS processes, in which the upper metal layer (used for busing) may have arelatively large minimum line width (Mikkelson *et* al., 1981). Therefore, the alternative is to let geometry overlap the cell boundary. The following overlap rules allow shared busing:

Rule 1: Artwork primitives in a cell definition may extend outside the user-specified cell boundary.
Rule 2: Boundaries of cell instances may not overlap.
Rule 3: Artwork primitives that overlap a cell instance boundary must extend into the cell by no more than a minimum line width; they must also be overlapped by existing cell geometry.

This modification of the methodology requires that only one more design rule test be performed: geometry that overlaps the cell boundary of an instanced cell must be flagged if it is not overlapped by geometry within the included cell.

Ports

Designers have long wanted to be able to compare schematics to artwork. The schematic drawing systems used to create machine-readable schematics usually allow two types of signals to be connected into a cell:

1. Signals which enter through pre-defined "ports";
2. Global signals, such as power and ground.

Both types of I/O signals must be named. Artwork and schematic data can be compared accurately by a hierarchical method if artwork I/O signals also have corresponding names. Providing these names is relatively easy, because all I/O signals in the artwork are ports (no global signals exist).

Thus, the following additional constraints make this comparison easy:

Rule 4: All cell definitions must be named.
Rule 5: In a cell definition, any geometric figure that touches or extends beyond the cell boundary is a port of the cell. All ports must have names.
Rule 6: All cell instances must be named.

Implementation

A new artwork design and verification system being developed at Hewlett-Packard uses these constraints to provide fast verification. The following questions are some of the ones that arise when such a system is implemented:

- How are ports recognized?
- How are overlap limitations enforced?
- How is the external abstraction of a cell created?
- How are DRC errors communicated to the designer?

Data Representation

We chose a single database containing all artwork data for the following two reasons:

1. The geometric operations traditionally used for DRC are also used in this system to create cell abstractions.
2. We wanted to use the graphics editor to look at DRC test reports.

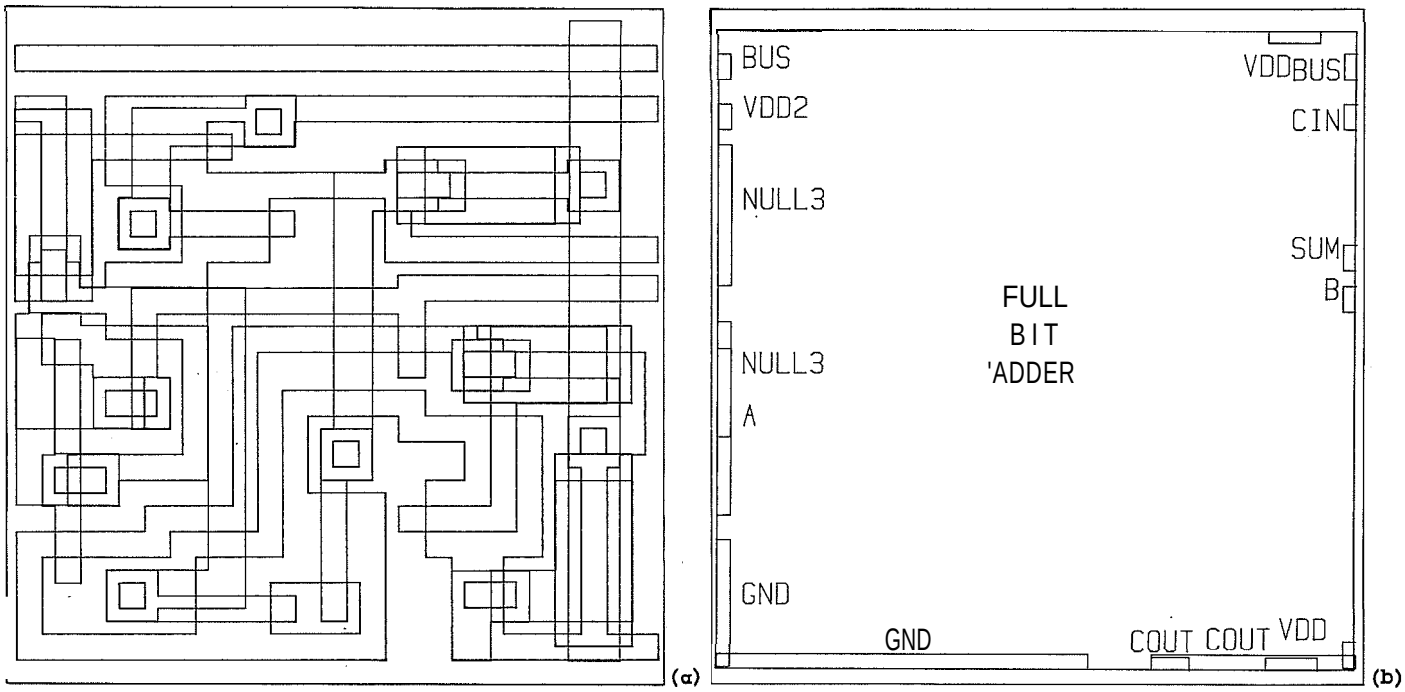
With such close coupling between traditional DRC functions and editing functions, data translation can be avoided by using a single database for both. Figure 2 shows an overall artwork data-flow diagram.

Within a cell, each line segment on a layer is ordered in two ways:

1. By means of a geometric X-Y ordered tree. This method provides sorted line segments, which are useful in algorithmic operations. Unlike a sorted sequential file, it allows incremental changes in the data.
2. By grouping into polygonal shapes. This method is preferred for graphics editing.

The data representation is complicated by the fact that cell definitions may contain geometry that exists outside of the cell boundary. When an instance of a cell is placed inside a "father-cell", each figure of the "son-cell" that is outside the cell boundary is entered as a figure in the father-cell. These figures are identified as being attached to the son-cell; they cannot be moved independently. They are treated as figures in the father-cell when artwork operations, component extraction, and artwork output functions are performed.

The database also contains a globalprocess file that specifies the sequence of operations and tests required to check design rules, extract the electrical components, and create cell



abstractions. Each user-specified command sequence is named, and can be invoked from the graphics editor, either as a foreground or a background task.

Artwork Editing

The artwork editor is functionally similar to a previous Hewlett-Packard graphics editor (Infante *et al.,* 1978). One significant modification is that angles are locked to multiples of 90 degrees ("Manhattan" angles). Another difference is that cell boundaries can be polygonal, as opposed to rectangular, and can be specified by the user. The use of polygonal boundaries prevents a decrease in density when cell overlap is constrained.

A unique characteristic of the editor is the interaction between it and the artwork operations. **A** new cell that has been drawn and saved ,is not instantly available to the editor for inclusion in other cells. It must first be abstracted; that is, the external view of the cell must be created by artwork operations such as AND, OR, NOT and OVERSIZE, as specified in the process file. A macro facility within the editor lets the operations and tests be performed whenever a cell is saved.

Artwork Operations and Modifications

Four artwork operations are provided:

1. Union or "OR"
2. Intersection or "AND"
3. Removal or "AND-NOT"
4. Polygon OVERSIZE (positive and negative)

These operations form the backbone of the following capabilities:

- Cell abstraction
  DRC feature isolation
  Component extraction (transistor and capacitor isolation)

Most of these operations are well known'for designing non-hierarchical artwork. The first three operations work normally

FIGURE 3. Different cell representations are used, depending on the needed level of abstraction. (a) Complete mask description of a full bit adder cell. (b) Bounded cell representation, with the external port connections labeled. (c) "Donut" of the cell, with all geometry within MAXRULE of the boundary shown.

in this hierarchical system, without any special cases, due to the limit on overlaps. The OVERSIZE operation algorithm must be modified for hierarchical operation, however, to prevent negative oversizing from causing port geometries to shrink away from the cell boundary.

Design-Rule Checking

Design-rule checking of a primitive cell (one which does not include other cells) is the same as it is in the non-hierarchical

case. No false errors need be handled as special cases, for the following reasons:

1. Shared buses are drawn so that half the bus width extends beyond the cell boundary.
2. No enclosure violations can exist, because of the overlap restriction.

After checking the cell, a "donut" of the cell is created (Figure 3c). This abstraction consists of

1. All geometry outside the cell boundary.
2. All geometry within MAXRULE distance of the cell boundary, where MAXRULE is the maximum external spacing rule in force.

Checking the design rules of a cell that contains other cells requires that "donuts" exist for all of the son cells. Before checking a father cell, the donut geometries of the son-cells are merged with the geometry of the father-cell. The boundaries of all son-cells are kept as a separate mask level. When the design rules of the father-cell are checked, any internal-width and enclosure violations that are detected must be ignored — if they are within a son-cell boundary. Spacing violations are reported unless both violating line segments are within a single son-cell's boundary.

All of the rules, including the complicated anti-reflection rules, can be checked using this method. This contrasts with other DRC methods that check for validity while the editing is in progress. These other techniques provide interactive feedback, but they cannot check some of the more difficult rules.

If the user asks that checking be performed automatically whenever a cell is saved, a "donut" of each cell is always available for checking higher-level cells. If automatic checking is not used, then checking of a given cell forces a DRC of any son-cell that has not been checked since the last time it was modified by the editor.

Component Extraction

Component extraction refers to the analysis of IC artwork to obtain the equivalent transistor-level schematic net-list. This transistor circiiit is usually post-processed to yield a logic-level circuit. Designers have long wanted to compare this logic-level schematic with a schematic previously entered into a drawing system. This comparison is difficult, at best, if it must be performed on the whole chip. It is especially difficult to pinpoint the area in which an error has occurred. However, with hierarchical component extraction the comparison is much simpler, and the error-reporting is far more precise.

Hierarchical extraction requires one additional constraint:

Rule 7: The active area of a transistor may not touch the cell boundary.

This rule prevents transistors from crossing cell boundaries, thereby letting each cell be extracted independently.

Component extraction of a cell is straightforward. Only the geometry within the cell boundary is extracted. All geometry outside the cell boundary is extracted with each cell instance. Extraction steps include the identification and sizing of transistors, area capacitance, sidewall capacitance (except at the cell boundary, where computation is deferred until the cell is used), and node-to-node capacitance. Node-to-node capacitance calculation can only be done between nodes within a cell. Connections to all son cells are also recorded.

The resultant transistor-level schematic can be converted to a logic-level schematic, which can be compared to a previously entered schematic. Comparison is simplified by the fact that each cell instance must be named. Because this constraint also exists in the logic-design system, mismatches can be reported precisely.

Artwork Modification

HP designers have been able to modify artwork algorithmically since 1978 (Tucker and Haydamack, 1978). They use this capability primarily to modify circuits so as to track process changes which are made to improve yield and performance. This capability requires no additional software in this system because the database for the editor is the same as that used for the artwork operations. Modified artwork is output the same way as other artwork.

Artwork Output

The output subsystem avoids the restriction to Manhattan geometries imposed on the rest of the system. Although Manhattan angles were strictly adhered to in the design of a recent 32-bit processor (Beyers *et al.*, 1981), the use of 45-degree angles for low-level cells can save a lot of area in some processes. Therefore; the output subsystem allows output of cells that have been designed using other artwork systems. The database contains only a hand-drawn abstraction for these cells along with a file name. This file contains the detailed cell artwork in a common interchange format.

Artwork can be output for either reticle pattern generators or e-beam exposure systems. Pattern generator output is especially fast in this system, because the overlap constraints allow hierarchical rectangle fracturing without double exposing any edges.

Performance

Early performance results are encouraging. A prototype system shows roughly the expected gains in performance. A 32-bit ALU and register stack, comprising 92,000 rectangles and 14,000 devices, is processed in 2.5 minutes on a DEC 2060. The processing includes generating four new layers, extracting a hierarchical schematic from the artwork, and converting the input layers and the generated layers into rectangles. The same task takes 60 times as long on a conventional system that expands the hierarchy before doing the analysis. The performance is eight times better because repeated cells are looked at only once. The rest of the performance gain is due to the smaller amount of data handled.

Furthermore, the cost of checking each cell as soon as it is entered is not prohibitive. Even for the largest cell, only 15 seconds of CPU time are needed to perform the operations which generate the new layers and extract the schematic. This procedure provides immediate feedback, and keeps errors from being propagated throughout a design.

Other Methodologies

We have presented a single methodology for developing hierarchical circuits. However, the same tools can be used for somewhat different methodologies. In particular, the overlap restrictions can be modified by specifying different operations in the process file. Loosening the overlap constraints will in-

evitably decrease the verification performance, because more interactions will have to be checked.

References

Beyers, J.W., L.J. Dohse, J.P. Fucetola, R.L. Kochis, C.G. Lob, G.L. Taylor, and E.R. Zeller. 1981. "A 32b VLSI CPU Chip," *ZSSCC Digest of Technical Papers.*

Infante, B., D. Bracken, B. McCalla, S. Yamakoshi, and E. Cohen. 1978. "An Interactive Graphics System for the Design of Integrated Circuits," *Proceedings of the 15th Design Automation Conference.*

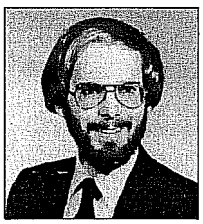
Mikkelson, J.M., L.A. Hall, A.K. Malhotra, S.D. Seccombe, and M.S. Wilson. 1981. "An nMOS Process for Fabrication of a 32b CPU Chip," *ZSSCC Digest of Technical Papers.*

Scheffer, L. 1982. "IC Design and Verification Using Strict Hierarchy and Programmability," Ph.D. thesis, Stanford University.

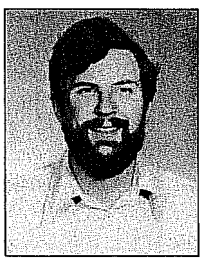
Tucker, M. and B. Haydamack. 1978. "A System for Modifying Integrated Circuit Artwork Through Geometric Operations," *Asilomar Conference on Circuits, Systems and Components.*

Whitney, T. First Quarter 1981. "A Hierarchical Design-Rule Checking Algorithm," *LAMBDA* (now *VLSI DESIGN).*

About the Authors



Mike Tucker received the BSEE degree (1974) from Iowa State University and the MSEE degree (1975) from Stanford University. He joined Hewlett-Packard in 1975. Since 1976, he has been with Hewlett-Packard Design Aids in Cupertino, California, where he is now Design Verification Project Manager.



Lou Scheffer received the BSEE degree (1974) and the MSEE (1975) from Caltech. At Hewlett-Packard in Loveland, Colorado, he helped design and lay out a digital filter chip for a spectrum analyzer. This experience prompted his interest in CAD tools, so he moved to California, to develop CAD tools for HP and to work toward the Ph.D. degree in the CAD area at Stanford. Lou now works for Valid Logic Systems, Inc. in Sunnyvale, California.