(14)

Measurements of a VLSI Design

Technical Report

R.S. Fabry
(415) 642-2714

"The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government."

Contract N00039-82-C-0235
November 15, 1981 - September 30, 1983

ARPA Order Number 4031

Measurements of a VLSI Design

John K. Ousterhout
David M. Ungar

Computer Science Division
Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720

Abstract

This paper presents data about three facets of a recently-completed VLSI design containing 45000 transistors. The first set of data describes the mask-level features of the circuit, from which it is seen that almost all features have at least one small dimension. The second set of data analyzes the hierarchical cell structure used by the designers to specify the circuit. The measurements show that composite cells have a different structure from primitive cells, and that, outside of arrays, cells are rarely re-used. The third set of data concerns the usage of an interactive layout program during the circuit's design. In spite of the circuit's size, the most frequently invoked commands were all simple.

1. Introduction

Although several large VLSI circuits have been designed during the past few years, there has been little publication of data concerning the structure of those circuits or the ways in which they were designed. Bentley et. al. [1] collected low-level mask data for several university projects, but did not include measurements of the logical structure of the circuits or of the tools used to design them. This paper presents data on the design of RISC I, an NMOS VLSI circuit containing about 45000 transistors [2,5]. Three different kinds of data are presented, pertaining to a) the low-level mask structure of the circuit, b) the hierarchical cell structure used by the designers to specify the layout, and c) the usage of Caesar [4], an interactive program with which the layout was entered. Our intent in presenting the data is twofold: first, to provide information for CAD engineers so that they can tailor their systems to the needs of designers; and second, to provide data for use in comparing VLSI designs.

The work described here was supported in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3803, monitored by the Naval Electronic System Command under Contract No. N00039-78-C-0013-0004.

2. Background

RISC I is a single-chip 32-bit NMOS microprocessor with estimated performance in the range of DEC's VAX-11/780 [5]. A team of graduate students at UC Berkeley designed and layed out the chip during the winter and spring of 1981. The chip has been fabricated, and is currently being tested. The design rules used for the chip are those defined by Mead and Conway [3]. All rules are based on a scale factor called *lambda* (λ). The Mead-Conway rules provide six mask layers including a single layer each of metal and polysilicon, and butting contacts (no buried contacts). Metal-to-metal pitch is 8λ under the Mead-Conway rules. The RISC I design is 5150λ wide, and 3850λ tall. It includes 180 cell definitions, and the complete mask pattern contains 530,000 rectangles. The fabricated version of the chip has λ = 2.0 microns.

The layout tool used for specifying the circuit was Caesar [4], an interactive program that utilizes color graphics workstations connected to a DEC VAX-11/780. Caesar is similar to most existing layout programs in that it allows designers to specify mask patterns (called *paint* in Caesar) and to group those patterns into hierarchical cell structures. Internally, Caesar represents paint as rectangles on the various mask layers, but designers do not deal directly with the rectangles. Designers specify the feature shapes and the cell hierarchy, and Caesar manages the implementation of the shapes with rectangles: the program automatically splits and joins rectangles to accomplish the designer's wishes while maintaining a minimized representation. (Because Caesar represents features with rectangles, the two terms will be used interchangeably in the remainder of the paper). Caesar does not contain any placement or routing aids. Caesar restricts designs to be *Manhattan*: all lines are parallel to the x- or y-axis.

To generate the statistics presented in Sections 3 and 4, we used Caesar's database subroutines to gather raw data about the rectangles and cells of the design. The data were then post-processed with UNIX utilities and special purpose statistical reduction programs.

In addition, Caesar's command interface was instrumented to save statistics on command usage. These statistics were gathered over many weeks of usage and post-processed to generate the condensed results of Section 5.

3. Mask Features

This section contains measurements of the low-level rectangle characteristics of the RISC chip. In gathering the data, the hierarchical structure of the design was ignored; only the resulting mask features were considered. The measurements show that most mask features are small, and that almost all features have at least one small dimension.

3.1. Feature Size and Density: Square Bins

Our purpose in measuring the mask features was to help us devise efficient data structures and algorithms for design tools. In particular, we wanted to understand the likely efficiency of *bin-based* data structures, whose purpose is to speed up database searches. One of the most frequent operations in VLSI design systems is to locate all of the features that overlap some small area of the whole chip. For example, an interactive display system must allow the user to "zoom in" to a small window on the chip, and design rule checkers must verify that features within an area can be fabricated correctly. For large chips, a linear search of all features is prohibitively expensive. Many systems implement special structures to permit efficient location of the features within a given area. The most common (non-hierarchical) structure divides the circuit up checkerboard-style into a two-dimensional array of square *bins*, and keeps pointers from each bin to all of the features overlapping that bin.

There are two conditions that must hold for a bin structure to be useful:

[1] Bins must be small enough to reduce the number of rectangles per bin to a tractable quantity, not merely for the average bin, but for almost all of them.

[2] Bins must be large enough so that rectangles rarely occupy multiple bins; space efficiency is important.

We chose several candidate bin sizes, then for each size placed 1000 bins at pseudo-random locations in the RISC I chip, and counted the rectangles that overlapped each bin. The cumulative distribution is shown in the first columns of Table 1. For example, out of the 1000 bins 16λ on a side, 10% of them had 3 or fewer rectangles, half of them had 13 or fewer, the mean was 14, and 90% of them had as many as 27 rectangles. Since the 90th percentile counts are only twice the 50th percentile counts and the means and medians are nearly equal, we conclude that the density of features is relatively uniform over the chip (this came as no suprise to us: a non-uniformity would have implied that some portions of the design are less space efficient than others).

The last column of Table 1 contains the average number of bins overlapped by each rectangle, assuming that the circuit is divided up into an array of bins of the given size. This information was computed by comparing the number of rectangles counted per bin to the total density of rectangles over the entire chip. The number of bins per rectangle is one measure of the storage overhead associated with a bin structure (if a rectangle overlaps two bins, then there must be a pointer to the rectangle from each bin).

Bin Size	Number of Rectangles per Bin				Bins per Rectangle
λ	10%	50%	90%	mean	mean
.5	0	1	2	1.0	150
1	0	1	3	1.2	45
4	1	2	5	2.7	6.3
16	3	13	27	14	2.0
32	3	36	77	39	1.4
64	17	130	250	140	1.3
256	580	1700	3500	2100	1.2

Table 1. Rectangle information for different bin sizes. The columns headed 10%, 50%, and 90% are percentiles: if a bin size of four is used, 90% of all bins will contain 5 or fewer rectangles, but only 10% of all bins will contain 0 or 1 rectangle.

Table 1 suggests that storage overheads become severe for bin sizes less than 16λ, while the number of rectangles per bin becomes large for bin sizes greater than 64λ. The best bin size appears, from the data, to be around 32λ.

3.2. Rectangle Shape Data

The results of Bentley et. al. indicate [illegible] large features tend to have at least one short dimension. This seems plausible since wires make up a substantial portion of VLSI circuits. Table 2 contains information about the dimensions of the rectangles in RISC I; the measurements are based on a pseudo-random sample of 53,000 of the 530,000 rectangles in the chip. The percentile information for each row is independent: 90% of the rectangles had a width of 9λ or less, and 90% of the rectangles had a height of 15λ or less, but these are *not* the same 90%. "Aspect ratio" refers to the ratio of height to width.

The data in Table 2 agree with the measurements of Bentley et al. and support the claim that most features have at least one small dimension: 95% of the rectan-

gles in RISC I have a short side of 6λ or less (in Bentley's data for 16 chips, approximately 96% of all features had at least one dimension less than or equal to 10λ). Caesar merges smaller rectangles into larger ones whenever possible, so we would expect features to have even smaller average sizes in other systems without automatic merging. Half of the rectangles have one dimension at least 2.5 times as great as the other.

One way to exploit the fact that features are skinny is to classify rectangles by orientation as either "logs" or "poles". Separate bins could be used for each orientation: a set of short fat bins for logs and a set of tall and narrow bins for poles. Such a bin structure would likely have less storage overhead than square bins with the same area per bin.

4. Measurements of the Cell Hierarchy

One of the most promising and popular techniques for coping with the complexity of a VLSI design is to organize the design as a hierarchy of cells. Each cell in a hierarchical design contains mask features (which we refer to as paint) and other cells, arranged in a tree structure (see Figure 1). Hierarchical design offers two potential advantages. First, it provides modularity: different cells can be designed independently by different designers, and designers can incorporate each other's cells without having to build them from scratch. The second potential advantage of hierarchical designs is efficiency. Popular cells may be replicated several times, but the design system need only store a single copy of the representation. The cell hierarchy can be used to reduce search time: when searching for all features overlapping a given area, it is unnecessary to search any cells whose bounding boxes do not overlap the area of interest.

To analyze the hierarchy of the RISC I design, we eliminated all non-functional cells such as logos, and then classified the remaining cells in several ways, as indicated by Tables 3 and 4. We gave separate consideration to three different classes of cells. Primitive cells are those containing only paint (no subcells); of the 180 total cell definitions in the RISC I design, 125 were primitive. Composite cells are those containing subcells. The top-most cell in the hierarchy is called the "floor" cell (when viewed in Caesar it looks like a chip floor plan); it is treated specially for one of the measurements because it contains an extraordinarily large amount of paint.

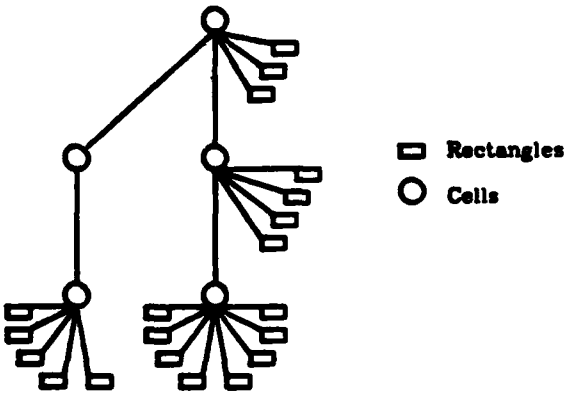

Figure 1. An example of a hierarchical layout, wherein each cell consists of mask features (rectangles) and subcells.

Class	Number of Cells
Primitive	125
Composite	55
Total	180

Table 3. Cell classes.

	min	10%	25%	50%	75%	90%	95%	max	mean
		Percentile Data							
Width (λ)	1	1	2	4	6	9	20	5100	6.4
Height (λ)	1	1	2	4	6	15	30	2300	7.6
Short Side (λ)	1	1	1	2	3	4	6	74	2.5
Long Side (λ)	1	2	4	5	10	24	42	5100	12
Area (λ^2)	1	4	5	14	24	78	110	100000	39
Aspect Ratio	0.0013	0.14	0.33	1.0	2.5	4.3	8.5	780	2.3

Table 2. Rectangle dimension characteristics. The percentages have the same interpretation as in Table 1: for example, 95% of all rectangles are no more than 15λ in height.

4.1. Cell Areas

The first statistic in Table 4 measures cell areas. Most primitive cells are relatively small — no more than about 80λ on a side — whereas composite cells are much larger. The large size of composite cells is expected, since they must (by definition) contain one or more smaller cells. For both primitive and composite cells there are a few cells that are far larger than the others, although variations in size were greater among composite cells than among primitive ones.

4.2. Mask Features in Cells

The second statistic in Table 4 counts the number of drawn rectangles in cells (rectangles defined in a subcell are not counted as part of the subcell's parent). The mean number of rectangles per cell is 120, and the 90th percentile falls at 390 rectangles. The rectangles are distributed relatively evenly between the polysilicon, diffusion, metal, and contact cut layers. The floor cell contains over 7000 drawn rectangles, or about 25% of all the drawn rectangles for the chip. These rectangles constitute most of the global wiring of the chip, and had to be put in the floor cell for ease of editing with Caesar. Recent changes to Caesar make this unnecessary: if the design were done today, the wiring would be split among several additional subcells. Composite cells tend to contain few drawn rectangles: more than half contain none at all. Because of the small number of rectangles in most cells, there seems to be no advantage to supplementing the hierarchical cell structure with a bin structure, except in a few rare cases such as the floor cell.

The paint density measurement was derived by dividing the total area of paint rectangles drawn in each cell by its total area. As with the drawn rectangle count, rectangles in subcells are ignored. Since paint may be present on any of the various mask layers, the density is occasionally greater than 1.0. The density of paint in composite cells is an order of magnitude smaller than in primitive cells. Our observation for RISC I is that primitive cells tend to be all paint and composite cells tend to contain almost nothing but subcells. Even the global wiring does not affect this judgement: wiring tends to be restricted to a few narrow channels.

4.3. Use of Subcells

The "total uses" line shows that the RISC I design is highly regular: on the average, each cell is used 21 times. Arrays were used frequently: 38 primitive and 4 composite cells, or about 25% of all cells, were used in arrays.

The line labeled "subcell count" in Table 4 counts an array as a single subcell: half of the composite cells contained 2 or fewer such subcells. Only a few cells contained many subcells; many of these were program-generated cells such as PLAs. It appears that if arrays are handled specially (as they are in Caesar), no other special techniques need be incorporated in DA systems to deal with the cell hierarchy: because cells contain so few subcells, a linear search is sufficient to find a desired subcell.

The last lines of Table 4 count the number of distinct places in the RISC I hierarchy where cell definitions are used, counting each array as a single use

	Cell type	Cumulative Data 10%	50%	90%	Mean
Area(λ^2)	primitive	270	4300	27K	10K
	composite	17K	150K	2.2M	1.5M
	all	350	11K	250K	450K
Drawn rects.	floor	7600	7600	7600	7600
	primitive	12	78	420	140
	composite w/o floor	0	0	270	65
	all w/o floor	0	46	390	120
Paint density	primitive	0.53	0.92	1.2	0.90
	composite	0	0	0.28	0.069
	all	0	0.74	1.2	0.65
Total uses	primitive	1	6	32	29
	composite	1	1	4	2.3
	all	1	3	30	21
Subcell count	composite	1	2	34	19
Distinct uses	primitive	1	1	2.7	6
	composite	1	1	1.1	2
	all	1	1	2.2	4

Table 4. Cell Characteristics.

of the arrayed cell. More than half of all cells are used in only a single place. Fewer than 10% of all cells are used in more than 4 places. This indicates that almost all of the regularity in the design is due to arrays. We can account for this in two ways. First, VLSI designers are willing to generate special purpose cells for each usage in order to minimize area and maximize speed. Second, Caesar does not permit parameterization of cells except through arrays and simple geometric transformations such as rotation; to make slight modifications to a cell, a designer must make a separate copy of the definition and then modify the copy. As more powerful design tools become available we hope cells will be re-used more frequently than in RISC I.

There are two overall conclusions to be drawn from the hierarchy data. First, composite cells have markedly different characteristics than primitive cells. Thus it may be appropriate to use different techniques to deal with the two kinds of cells. Rowson [6] has already proposed such a distinction. The second conclusion is that it is essential to represent arrays explicitly in design tools and languages. Artwork analysis tools can be speeded up enormously by taking advantage of the replication present in arrays, and may not need to consider any other kinds of regularity (the RISC I design suggests that arrays are the only source of regularity).

5. Usage of the Layout Program

Over the last month of development of RISC I, Caesar recorded usage data for each interactive session related to the project. During this month, the subcells were integrated into the final layout, several additional cells were created, and the global wiring was specified. In all, the sessions included about 480 hours of Caesar usage, and a total of 365,000 commands. The designers had nearly unlimited access to Caesar stations, so they tended to think in front of the terminals, rather than using the design stations only to enter pre-conceived designs.

Table 5 summarizes the usage of Caesar commands by group. In considering the data, it should be noted that the commands all execute in about the same time, so users were not biased in favor of certain commands by execution time. Overall, Caesar used only 5% of the available CPU time and commands completed in about 1 second each.

The most frequently-invoked commands do very simple things. Cursor positioning alone accounts for 84% of all commands. The large number of keystroke cursor commands is because they move the cursor by one lambda unit; it takes several such commands to accomplish the equivalent of a mouse positioning.

About once a minute, designers changed what was visible on the color graphics display. Almost half of these changes consisted of scrolling the picture left, right, up, or down.

On the average, modifications to the circuit occurred about once a minute; almost all of these modifications came from painting commands that made slight modifications to mask features. Commands that modified the cell structure of the circuit, such as adding a new subcell or moving an existing one, were rarely

Command		No. of Uses	Percent of all Commands	Average Time Between Uses
Cursor Positioning	(by keystrokes)	242000		7 sec
	(by mouse)	60600		28 sec
	(miscellaneous)	5606		5 min
	Total	308000	84%	6 sec
Viewing	(scroll)	10200		3 min
	(change)	6930		4 min
	(grid and misc.)	6120		4 min
	Total	23250	6%	74 sec
Selection	(cells)	2338		12 min
	(paint)	830		34 min
	Total	3259	1%	9 min
Painting		24645	7%	70 sec
Create Label		1204	.3%	24 min
Cell Manipulation	(move, copy, etc.)	1329	.4%	22 min
Generate CIF		284	.1%	100 min
Other		3667	1%	11 min
Total		365000	100%	5 sec

Table 5. Caesar usage statistics during RISC I design.

issued: on the average, such commands occurred only every 22 minutes.

The infrequency of commands that modify the circuit, particularly those that change the cell structure, is encouraging for the development of more powerful design systems. For example, suppose that design rules are to be verified as the designer makes changes to the circuit. Painting commands are issued about once a minute; these affect only small areas of the circuit so reverifying the design rules for those areas should be cheap. The cell manipulation commands may result in large amounts of work in reverifying design rules, but they are only invoked every 20 minutes or so. Thus, it should be feasible to build an incremental design rule checker that runs as a background process and keeps an up-to-date picture of violations as the circuit is edited[1].

6. Warnings and Conclusions

Caution must be exercised in interpreting these results, since they represent only a single large chip. Our low-level mask data agree with the measurements of Bentley et al, but until similar measurements of hierarchy and tool usage have been presented for other VLSI circuits, it will be difficult to know whether or not these results are generally applicable. Nonetheless, the RISC I design has the following characteristics:

[1] Features are generally small, and even those that are large have at least one small dimension.

[2] Cells fall into two general classes: primitive cells containing a small number of uniformly-distributed mask features and no subcells; and composite cells containing a small number of subcells but almost no mask features.

[3] Regularity occurs almost exclusively through the use of arrays. Cells are generally used in only a single instance or array.

[4] The most popular commands in the interactive layout editor were simple in nature; only rarely was the circuit modified and even then most of the modifications were local changes to mask features.

[1] We have also gathered Caesar usage data during a graduate-level course where students designed circuits somewhat smaller than RISC. The measurements for the course include almost 4000 runs and 1100 hours of data. The course data is quite similar to the data in Table 5 except that painting occurred about twice as often as in Table 5 and cell manipulation occurred about three times as often (about every six minutes). Although this is heavier usage than in Table 5, the modification commands are still sufficiently infrequent that it should be possible to build incremental design aids.

7. References

[1] Bentley, Jon L., Haken, Dorothea, and Hon, Robert W. *Statistics on VLSI Designs*. Computer Science Technical Report, Carnegie-Mellon University, Pittsburgh, PA, 1980.

[2] Fitzpatrick, Daniel T., *et al*. "A RISCy Approach to VLSI." *VLSI Design*, Vol. 2, No. 4, Fourth Quarter 1981.

[3] Mead, Carver, and Conway, Lynn. *Introduction to VLSI Systems*. Addison-Wesley, 1980.

[4] Ousterhout, John K. "Caesar: An Interactive Editor for VLSI Layouts." *VLSI Design*, Vol. 2, No. 4, Fourth Quarter 1981.

[5] Patterson, David A. and Sequin, Carlo H. "RISC I: A Reduced Instruction Set VLSI Computer." *Proc. Eight International Symposium on Computer Architecture*, May 1981, pp. 443-457.

[6] Rowson, James A., *Understanding Hierarchical Design*, Ph.D. Thesis, California Institute of Technology, 1980.