T638
T638

MIT/LCS/TR-378

MACE: A Multiprocessing Approach to Circuit Extraction

Samuel M. Levitin

October, 1986

MIT/LCS/TR-378 Levitin MACE: A MULTIPROCESSING APPROACH TO CIRCUIT EXTRACTION

P19173

MACE: A Multiprocessing Approach to Circuit Extraction

by

Samuel Mark Levitin
B.S., Massachusetts Institute of Technology (1985)

Submitted in partial fulfillment
of the requirements for the
degree of

Master of Science
in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 1986

Copyright © 1986 Samuel Mark Levitin

The author hereby grants to M.I.T. and to Digital Equipment Corporation permission to reproduce and to distribute copies of this thesis document in whole or in part.

Signature of Author Signature redacted

Department of Electrical Engineering and Computer Science

June 2, 1986

Certified by Signature redacted

Christopher J. Terman

Thesis Supervisor

Certified by Signature redacted

Kenneth H. Slater

Thesis Supervisor

Accepted by

Arthur C. Smith

Chairman, Departmental Committee on Graduate Students

MACE: A Multiprocessing Approach to Circuit Extraction

by

Samuel Mark Levitin

Submitted to the
Department of Electrical Engineering and Computer Science
on June 2, 1986 in partial fulfillment of the requirements
for the Degree of Master of Science

Abstract

The ever-increasing complexity of VLSI chips threatens to choke out all available computer power unless methods are devised to keep the CAD tasks conveniently sized. A review of the current methods of multiprocessing approaches in the domain of layout verification precedes the discussion of the current work. A loosely coupled coarse-grained multiprocessing version of the circuit extractor used by Digital Equipment Corporation was built to run in the VAXcluster environment. The technical issues of how to divide the work and how to combine partial results to make final results are discussed. Test results and performance measurements are given, accompanied by an open question about how to gauge performance. Finally, explanation of the pieces left out, suggestions for further work in the field, and suggestions for a coarse-grained multiprocessor for circuit extraction are given.

Thesis Supervisor: Christopher J. Terman

Assistant Professor of Computer Science and Engineering

Keywords: multiprocessing, circuit extraction, VLSI CAD

Acknowledgments

The author wishes to thank Ken Slater, for guidance and support over the last 3 years; Chris Terman, for assistance in defining the ideas and serving as a rational sounding board when all sanity seemed gone; the members of the IV support team, especially Alice DiPace and Pani, who helped explain IV's curious behavior and inner mechanisms; Randy Parker, for providing light relief and production support; and Josh Marantz, a developer of EPIC proportions, without whom and without which the MACE concept would be merely pie in the sky.

The following are trademarks of Digital Equipment Corporation:

DEC	VAX	VAXcluster
VMS	DECnet	MicroVAX

To Susan, for supporting me at all the right times.

Table of Contents

Table of Figures

Table of Tables

Chapter One

Introduction

1.1 An overview

As VLSI chips become more complex, the CAD tools essential to their design become more crucial. As chips cross the threshold of 1 million devices, however, the strain of the best tools on the fastest system computing on the densest chip is too great: some advantage must be gained to combat the exponential increase in the complexity involved in larger device counts. One of the many ways to achieve a speedup is to modify an existing algorithm to use a multiprocessing environment. This document describes MACE, a modification to IV (Interconnect Verifier), an existing circuit extractor used by Digital Equipment Corporation.

A circuit extractor is one tool that performs layout verification; it reconciles one representation of the VLSI chip, the layout, with another, the schematic representation. IV's purpose is to analyze the layout in a VLSI chip to see how closely it matches the schematic representation. IV translates the layout from a layout language into the equivalent wirelist. Although circuit extraction is not a geometrically local task, that is the approximation made in this research. The methods of setting up the supposedly separate tasks and the methods of merging together the separate results into a coherent unit that correctly represents the whole chip forms the crux of the explanation.

The organization of this document is as follows: Chapter 1 serves as an overview, to explain the imminent computation bottleneck in the VLSI CAD realm, to analyze IV, and to preview the approach taken in this research. Chapter 2 details the design rule checking problem (DRC) and the circuit extraction problem (CE), how DRC has been parallelized, and why CE is fundamentally different and more complex. It also gives a brief overview of

EPIC, the task scheduler/controller for the current research. Chapter 3 explains IV, the circuit extractor currently used by Digital Equipment Corporation, along with its input and output formats. It also traces the design decisions made in the current work. Chapter 4 discusses the merging process in full detail. Chapter 5 gives algorithm analysis, testing strategies, testing results, performance measurements, and predictions. Chapter 6 enumerates the features and facilities that were omitted from the current system, and contains suggestions for further research. A portion is devoted to design considerations for building a circuit extractor that is amenable to rapid prototyping, especially in computing environments ranging from medium to coarse-grained parallelism. Appendix I is the glossary of technical terms. Appendix II reports the test results in a more verbose fashion than Chapter 5. Appendix III contains the cases that are not handled, along with a set of features that should be handled by a tool similar to MACE. In order to maximize universality of this document while not boring or insulting the more sophisticated reader, technical terms will appear in italics, and are explained in the footnotes.

1.2 IV: the starting point

1.2.1 IV overview

IV works on hierarchically organized layout. A root cell, usually corresponding to the entire chip, contains instances of other smaller cells, loose layout, or some mixture of the two. Those cells, in turn, may call other cells or contain layout. At the bottom level of the hierarchy are leaf cells, containing only layout. IV typically extracts the connectivity of the layout, identifies the devices, and records their sizes. It can also calculate parasitic capacitance. Its input is a layout file in a language like CIF [9]. Its output includes a wirelist for a circuit simulator, such as SPICE, or for a wirelist comparator, such as WLC [7, 10].

1.2.2 The scanline algorithm

IV uses a scanline algorithm to establish connectivity [2, 15]. IV scans in units called *swaths*, starting at the top of the chip and working down. Within a swath, it scans from left to right. The swaths share horizontal edges. A swath is defined by the highest feature remaining, where a feature is defined as the upper edge of a new polygon, the lower edge of a polygon already under analysis, or the bottom of the chip.

1.2.3 Current and active swaths

For a given swath, IV uses the swath previously extracted, which lies above the current one, as a source of state information. The state information includes which electrical nodes and devices are active in the previous swath and touch the current swath's upper border and which edges of polygons touch the border. The term *active* refers to structures within the previous swath; the term *current* refers to structures within the swath under analysis. Structures are active because they can still have an effect on structures under scrutiny. Current polygons are reconciled with the active ones. When a connection is made from a current structure to an active structure, the two structures are merged into the same structure. When a structure is encountered that does not connect with anything in the active swath, a new record is allocated for that structure, and a new name is assigned to it. A device is defined by a description of the technology used to fabricate the chip and by the various layers present in an area. For example, in an NMOS process, a device is defined by the presence of overlapping polysilicon and diffusion and the absence of the buried contact layer. The set of layers present is compared to the layers needed to establish a device. A device is recognized if the two sets agree exactly.

1.2.4 IV bookkeeping

IV measures the devices and stores the geometry of their *channels*.[1] At the end of

[1] A channel is the overlap area of polysilicon and diffusion for an NMOS device.

the extraction of each swath, IV calculates the size and area of the devices within the swath. After all swaths have been extracted, the structures are checked for consistency. For example, a device record contains pointers to the nodes that form the gate and source/drains of the device. These nodes may be renamed when they merge with other nodes in a lower swath. In the output phase, the comprehensive, coherent results are output into the wirelist format, and the errors in the layout are announced to the user.

1.3 MACE: the prototype

MACE, a Multiprocessing Approach to Circuit Extraction, is the system built out of IV with modifications to support multiprocessing. It runs under the control of the EPIC system [8]. MACE essentially imposes the notion of a task upon the serial, interlocking nature of the swath extraction of IV. Because it decomposes layout where layout is not amenable to decomposition, MACE must take steps at some point to compensate for the division.

MACE divides the layout into horizontal sections called *slices*, comprising many swaths, and pretends that the layout in each slice can be extracted independently.[2] This preparatory step is handled by the SPLIT verb in MACE. The extraction in MACE is a bit of bookkeeping surrounding a normal IV extraction, and is handled by the EXTRACT verb. The compensation phase takes all the individual extraction results and attempts to convert them into a circuit equivalent to what normal IV would have derived. This occurs by several stages of merging two adjacent slices together, using the MERGE verb. As the slices combine to make larger and larger slices, the results become more and more final, until there is only one slice the size of the whole chip, whose merged output resembles that produced by normal IV.

In IV, the previous swath serves as the context for the extraction of the current swath,

[2] A slice is a rectangular portion of a chip. Also called strip.

with the exception of the topmost swath in the chip, which has no predecessor. In MACE, there is no previous swath for the first swath in each slice of layout. (Typically there are many swaths within a slice.) Because MACE does not have this context from which to start the extraction process, it must perform further analysis of the temporary results produced by the EXTRACT verb. Later chapters will reveal in greater detail the nature of the problems encountered with this approach, the methods to solving them, and the performance and applicability of those solutions.

1.4 An overview of multiprocessing

To make the motivation for a project such as MACE more concrete, consider that in the last 10 years, chip complexity has increased 5000-fold, while computer speed has increased only 100-fold [17]. The problem of speedup has received some attention before, but there is no universal solution to all CAD problems [1]. Hardware multiprocessors or problem-specific hardware engines are one method that promises large speedup at the expense of the high cost and the time required to integrate such an engine into the CAD environment [14, 18]. Converting existing hardware systems to serve as a multiprocessing environment depends on the support of the operating system and hardware for the necessary operations [6, 12, 19]. It is this latter approach that was taken in the current research and in the related EPIC project [8].

Chapter Two

LV and extant multiprocessing methods

Before jumping into the details of IV and MACE, some analysis of the layout verification problems will put the current work into perspective. This chapter examines the problem of layout verification in greater detail. It explains and contrasts the two problems of circuit extraction and of design rule checking. It reviews the relevant work done on developing multiprocessing systems for DRC. It contains some insight as to why the circuit extraction problem is significantly harder than the design rule checking problem to modify into a multiprocessing environment. It concludes with a brief look at the system for controlling the parallel execution of a set of interrelated tasks on a multiprocessor, used by the current system.

2.1 Layout verification tasks

In the layout verification phase of VLSI design, the layout undergoes several tests to ensure that it fulfills the specifications for the chip. Additionally, it may be passed under the scrutiny of a set of check tools, which ensure that the layout is consistent with the other forms of representation. One assertion vital to establish in the verification process is that the layout actually embodies the functionality specified by the schematic description of the chip. Another is that the shapes of the physical metal layers are such that they are consistent with the size, shape, and separation that can be reliably produced by the fabrication process. The former task is the domain of a circuit extractor; the latter, of a design rule checker.

2.1.1 Circuit extraction

The circuit extractor is a translator of layout into circuit. It uses a description of the fabrication process to define which layers interact with others by merely crossing, and which require contacts to be electrically connected. The description also defines which layers constitute different types of devices. Several descriptions of circuit extraction and various implementations have appeared [5, 13, 15].

The most common approach to extracting connectivity is the bitmap approach [2]. The term bitmap is somewhat misleading because individual bits are not used to store useful data. Rather, the layout is broken into small tiles, each of which is likened to a bit. As the scanline passes from top to bottom and from left to right, the connectivity of a given bit is a function of its left and top neighbors, it they exist. If either neighboring tile is of the same material as the tile currently under analysis, they are the same electrical node; otherwise, the tile is a distinct electrical node. For a given tile, the various layers are handled by examining the neighbors. The layers can also connect within a tile, if two layers present connect without a contact. Recognition of devices comprises detecting the presence of the required layers and the absence of the forbidden layers within a tile.

2.1.2 Design rule checking

Design rule checking examines the chip layout with respect to a set of design rules. Typical design rules relate separation between physical shapes required to ensure that the shapes function as distinct electrical nodes on the fabricated chip. Design rule checking in general includes some electrical rules in addition to the geometric ones in the rule set, but DEC concentrates on the geometric checks. Other rules specify the minimum size that is able to be reliably drawn. In the VLSI chips of 1986, a minimum dimension of 1 micron is typical.

2.1.3 Comparison of the two tasks

Design rule checking is entirely a local operation because all the data necessary to generate a design rule error, or conversely, to prove that a given area is error-free, are in the area plus its immediate border. Ousterhout and Taylor call this border the *halo* [16]. Geometry in one corner of a large chip cannot cause design rule errors in another corner. Circuit extraction, on the other hand, depends on the generation and maintenance of a context essential for the correct extraction of a given area. Without this context, results are only partially valid. Judgments of electrical characteristics are impossible to make given a limited area, because two nodes that look distinct within the area could join outside the area.

Because of its locality, the task of DRC can be segmented with much less care than can the task of circuit extraction. As the subsequent sections show, DRC can be segmented, and in some cases, with no conversion of temporary results necessary. Circuit extraction, because of its nonlocal nature, has not lent itself to arbitrary segmentation. Several methods for breaking down the DRC of a large chip have been developed.

2.2 DRC parallelization methods

Consider the chip as a three dimensional entity. The x and y dimensions are the ones apparent when looking at the plot of the chip from above. The z dimension is the view from a side of the chip, in which one can see the various layers, one on top of the other. The approaches that are discussed in turn are the x-y method, the z-method, and the rule-method.

2.2.1 The x-y method

In this method of layout segmentation, the cuts are in the x-y plane. The majority of the work is done by performing DRC on the individual segments. After each sector of the chip is checked, all that remains to guarantee equivalence to a conventional whole-chip

DRC is to perform a check on the haloes of each sector. I call this final operation sewing, since it is similar to sewing together patches to make a quilt. Such an approach to parallel design rule checking was recently made [3].

2.2.2 The z-method

This method considers the interaction between the various sets of layers. One rule might refer to the minimum size of a contact, while another might relate the minimum size of one layer overlapping another to form a transistor. Performing several checks with partial rule sets, which collectively encompass the entire rule set, is equivalent to checking the whole chip over the entire rule set. There is no sewing step needed. This approach has also recently been reported by Nielson [11]. The rule set is separated in this fashion into separate jobs that can be scheduled independently. To gain a speedup, this approach uses the generic queue facility available in the VAX/VMS environment of a VAXcluster [19]. The queue server holds jobs in a generic queue, which feeds into several queues on the processors that compose the cluster.

2.2.3 A novel approach

A slightly different tack was take by Marantz [8]. This approach goes deeper into the heart of the DRC problem. The process of checking a single rule is not an atomic action; there are smaller sub-parts, which generate intermediate results. Marantz notices that these intermediate results are the same for the computation of several rule checks. This means that in the separate task for separate rule approach, as above, some computation is duplicated by two different processors. The potential exists that even the same processor would recompute intermediate results in two different rule checks. The approach taken is to divide the 40 or so rules into 120 steps. As many as 14 distinct rules depend directly on a given intermediate result. If communications costs, measured in time and disk space needed to store the intermediate results, are small compared to the computation costs, there will be a speedup of the aggregate computation.

2.3 EPIC

To facilitate this method of problem partitioning, Marantz designed and built the EPIC system [8]. EPIC serves a variety of functions involved in multiprocessing. Initially, it serves as the task scheduler. During the computation it serves as a computation controller, providing a front end, the *monitor*, with which the user can view the computation. It also handles the communication of status messages and the file transfer. Input to EPIC describes what tasks there are to be performed. EPIC reads an *execution control file* (ECF), which contains the tasks descriptions. A task description consists of the task name, its inputs, its outputs, and the single statement that is given to the computer to perform the task. All the file transfer is hidden from the user. In some preliminary measurements of EPIC on tasks with low communications overhead, he has measured speedup of 3.2 on 4 processors compared to sequential processing of DRC.

Chapter Three

MACE: A new weapon

MACE, a Multiprocessing Approach to Circuit Extraction, is the body of work that attempts to solve the question of how circuit extraction with overlapping layout can be performed using the coarse-grain parallelism provided by the EPIC system. MACE is a set of modifications and enhancements to IV (Interconnect Verifier), the circuit extractor currently used by Digital Equipment Corporation. The version of IV used in this research contains over 16,000 lines of source code, and includes some object modules. The MACE system contains all of IV's code, all IV object modules, and other code specific to MACE's functions. This additional code consists of 6400 lines of PL/I code, half of which are MACE's private copies of IV routines. MACE executable images are fully 50% larger than IV images.

This chapter is organized as follows: Section 3.1 gives an overview of IV. Section 3.2 gives the MACE mindset and shows how it motivated design decisions. The explanation of the verbs in MACE starts in section 3.3.

3.1 IV: the starting point

To get a better understanding of the question it is necessary to elaborate on IV [15]. The input is in a variant of CIF [9]. There are statements that represent geometric shapes, such as boxes, wires and polygons, and there are other statements such as *labels*, comments, and calls to other blocks of layout.[3] The output is a wirelist for a simulator like SPICE

[3] Labels are descriptive text associated with a node in a layout file that helps to identify for the designers' use the function of the node. Sample labels might be PHI, WRITE-ENABLE, DATA<3>, to indicate a clock signal, a control signal, or a slice of data, respectively.

containing the devices and their sizes, attached nodes, and so on [10]. IV always extracts the devices and names of the structures. Additionally, if the user wishes, IV calculates parasitic capacitance, the stray capacitance found between a node and ground.

The forms of IV output are:

cap file — a capacitance list. Capacitance is of two types: parasitic capacitance, between a node and ground, and coupling capacitance, between two nodes.

edge file — a list of polygon edges that touch the MBB of the chip.

XND file — eXtended Network Description file, which contains information about the chip; all the node names and their locations; all the device names, their sizes, locations, and type; and capacitance information if it is calculated.

device geometry file — contains the individual coordinates of the points that define the devices' channels. Also called DGR file, from Device Geometry Record.

wirelist — the input for the circuit simulator, which contains the device names, sizes, and its terminal names. The wirelist also contains the capacitance data if they are calculated.

log file — the record of the IV run, which contains information about cells encountered, CPU time taken in the various phases, errors in the layout, and status messages.

One initial task was to find appropriate points in the layout-to-circuit pipeline to break the pipe, in order to define a *task* in the sense of EPIC's tasks. The phases in an IV extraction, without capacitance, are the following:

- read the layout file into working memory
- flatten the layout, which may be at various levels in a hierarchy, especially for large chips, into a consistent, globally-defined grid.
- start the scan-line algorithm, as detailed in [2].

- after all layout has been processed, make the output formats from the internal data structures that remain when scanning is finished.

Flattening the layout, and why it's done

Large chips, which might have a multilevel organization such as the one in figure 3-1, may include instances of smaller cells, as from a library of standard cells, or not so small cells that correspond to the logical blocks of the chip such as datapath, mux, input pad, and so on.

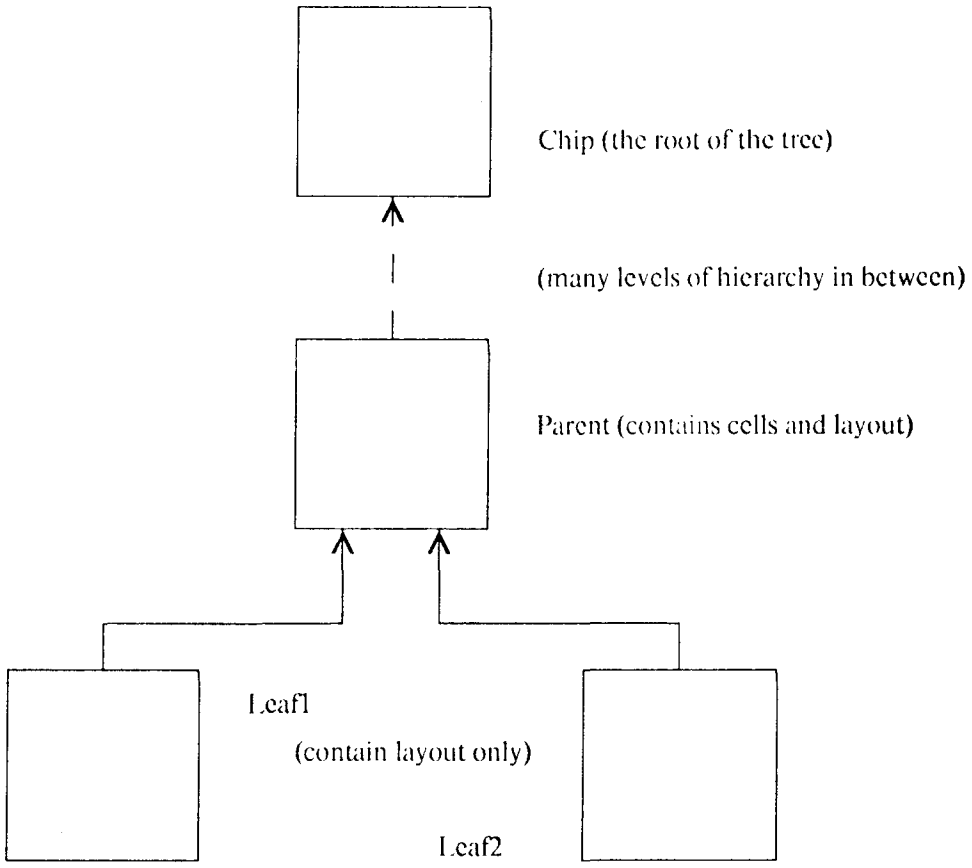


Figure 3-1: Hierarchy in a composite cell

They may in addition contain loose layout. In the flattening stage, all the individual elements of all the cells at the various levels of the chip hierarchy are brought into a

common representation.

From the viewpoint of the scanline algorithm, it is more efficient if all the layout within a given area can be identified directly, without making reference to the whole hierarchy. In order to enable convenient layout retrieval during the scanline algorithm, the layout is flattened.

3.2 MACE: the philosophy

MACE revolves around a very simple philosophy: pretend that extracting a segment is a local operation, and then make up for what was missed by this oversimplification later. It attempts to treat as many of IV's functions as possible as black boxes. MACE attempts in several places to create a state of the machine that is equivalent to an IV state, then to execute the same procedure that IV does, in the belief that this is one way to achieve equivalent results.

Error propagation was one such issue, in which MACE attempts to recreate the state of the machine. IV reports errors in the layout to the user in the log file. Some of the various error types are shown in table 3-1 . IV generates errors as a side effect of scrutinizing a data structure; MACE does likewise. The messages report structures IV believes to be of questionable validity. For example, an MOS capacitor is a valid electronic structure that could be introduced under some circumstances. However, more often than an intentionally introduced structure of this kind is the occurrence of a device whose two source-drains are shorted together, making a two-terminal device. This device functions as a parallel plate capacitor, and is flagged by IV as a questionable structure the user should scrutinize. More frequent than this are errors that result when unusual electrical properties are deduced, such as two nodes with different names that are shorted together.

Fortunately for MACE, IV generates the errors when it creates the wirelist. The process of writing out a wirelist involves examining each device, each node, and all the

Table 3-1: Some error messages IV reports

```
MOS capacitor detected
                As this is an unusual structure, IV
                notifies the user that it encountered one.

dangling node   a label unattached to any node

a shorted node  two nodes with unequal labels

well node not connected to VDD
                for CMOS process

surface plug node connected to VSS
                also for CMOS

no source-drains
                a device without any source-drain nodes

extra source-drain
                a device with too many source-drain nodes
```

structures derived from the extraction. If MACE were able to create an identical environment at this point, the errors would follow directly. This was the approach taken.

3.3 The MACE verb set

MACE was built as a separate subsystem under the IV system. IV commands did not apply in the MACE subsystem, and MACE commands were irrelevant in the IV system. The basic flow of data is illustrated in figure 3-2. The involved flowchart is illustrated in figure 3-6 on page 32. The following sections describe the MACE verbs in the order they were built. The order is also their order in the flow of a MACE invocation.

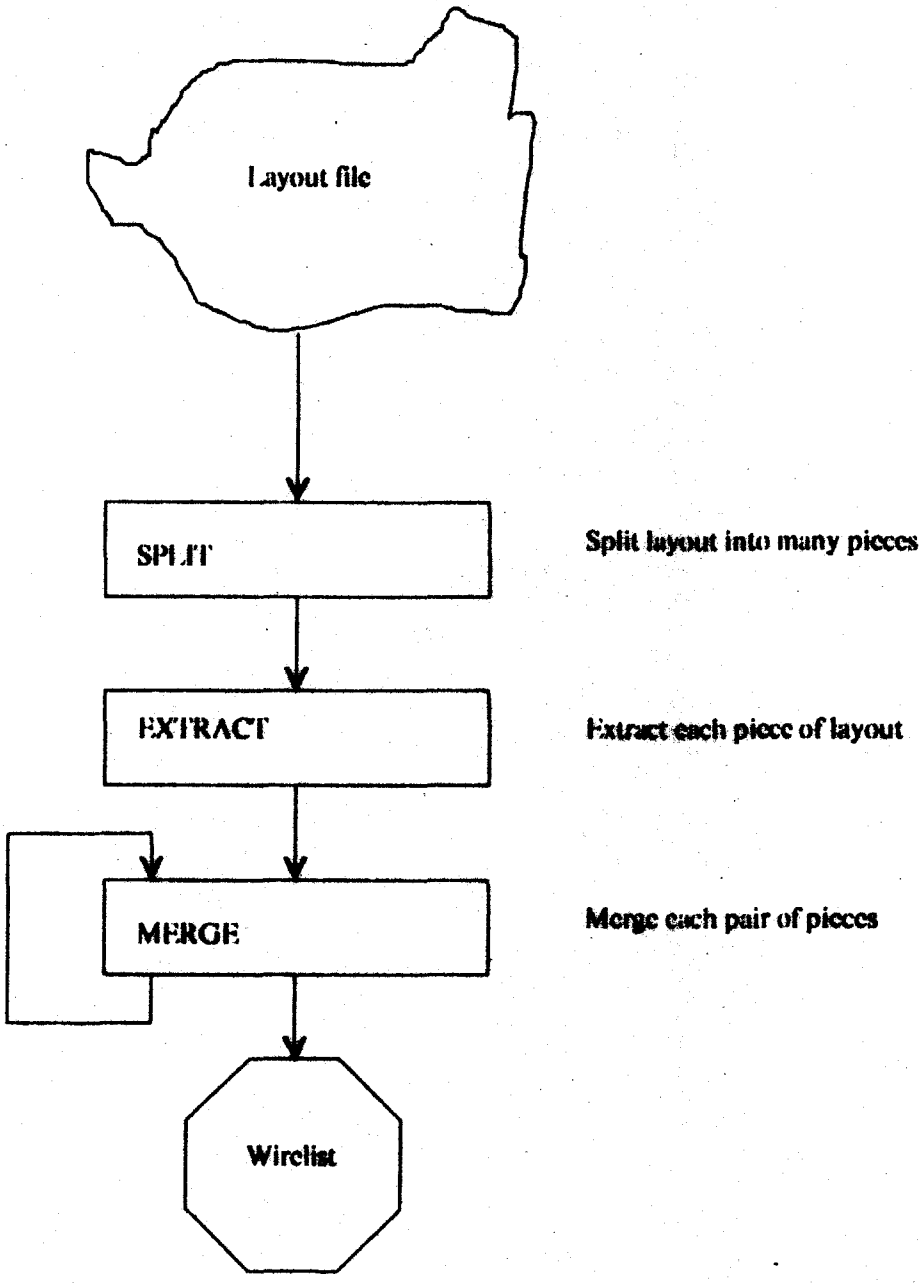


Figure 3-2: MACE data flow

3.3.1 The SPLIT verb

I chose to place the first break in the pipe at the point when the layout had been flattened and was all available for comparison on a single grid. This is accomplished with the MACE verb SPLIT. SPLIT was the first MACE verb implemented, and was accomplished by reordering existing IV code. SPLIT reads in the layout file and filters it into bins corresponding to different regions of the chip. Each bin is given a number or index, which is used by the other verbs to identify the data belonging to the segment. In general, if the chip name is CHIP and the IV file type is .TYP, files will be called CHIP−SEG1.TYP, CHIP−SEG2.TYP, and so on.

3.3.1.1 Fixed splitting

Splitting can be done in two ways: fixed or variable. In fixed splitting, the user enters the number of *slices* into which he wishes the layout partitioned. Fixed splitting into two slices is equivalent to halving the chip; with four slices it is equivalent to quartering it and so on. However, because all *cutlines* are drawn horizontally, quartering the chip creates four thin slices, not almost-square sections like a window pane.[4] While the facility existed to filter layout into arbitrary shapes whose borders are horizontal and vertical line segments, it seemed difficult in the early stages of MACE development to merge two staircase shaped regions. It also seemed unnecessarily complicated. In the absence of a clear benefit to handling arbitrarily cut slices, I opted for simply shaped ones.

Since all the cutlines are horizontal, the slices are regularly shaped, and the cutting process is well-defined for numbers that are not integral powers of 2. Merging must only be done on the tops and bottoms of the slices, and not on the left or right sides, since the left and right sides of the slices directly correspond to the left and right borders of the chip. Typically the number of slices is low, and corresponds to the number of processors one has available to run MACE.

---

[4] A cutline is an imaginary line imposed on a layout by which SPLITTING occurs.

3.3.1.2 Variable splitting

In variable splitting, the user enters a sequence of y values at which he wishes to split the layout. Currently, MACE prompts the user for a descending sequence of values, discarding values that are out of order. Also, the value 0 is used as a token to specify the last value. However, not all chips have their MBB's low y value equal to 0. For reasons of symmetry, often the point (0,0) falls somewhere inside the chip. IV places no constraint on placement, so MACE should not have done so either. A version of MACE for production use would have embodied a cleaner user interface that would have solved this problem. See section 6.1 for the features that did not make it into the MACE described here.

Rather than try to convert the internal memory to readable information, I decided merely to dump IV's geometric data unceremoniously to a disk file. I called the atomic unit of data transfer an iotrap, for Input/Output TRAPezoid, based on name of the IV data structure. IV's working memory contains trapezoids rather than rectangles because IV allows *non-Manhattan* geometry.[5] The iotraps were written out with the SPLIT verb and read by the EXTRACT verb.

3.3.2 The EXTRACT verb

EXTRACT was the second MACE verb implemented. It consists of a memory reconstruction phase, the extraction phase, and the output phase. It was the goal of the EXTRACT implementation that, except for a few status flags, that the state of the machine after the memory reconstruction phase be identical to the normal IV run after layout is flattened. The memory reconstruction phase locates and reads into memory the file full of iotraps. The extraction processes normally, as if the internal data structures had been derived from flattening the circuit and not from reading in the iotraps. In order not to corrupt the working of IV, MACE uses a copy of the IV code, except with different procedure names. The only other modifications that were necessary were the setting of a

[5] Manhattan layout contains only horizontal and vertical lines.

few status flags in order to force the generation of some data formats that are optional in a normal IV run. The EXTRACT verb produces the files CHIP–SEG1.EDG, CHIP–SEG1.XND, and CHIP–SEG1.DEV as output of extracting slice 1.

I chose to use existing IV data formats rather than to create my own for several reasons. I judged that using the existing routines to read and write formats would be better than to have to design a file format to contain all the various data types. This approach kept the code size down, and also used the existing IV support team's expertise on the code that was assumed to function correctly from the start. Consult section 6.3.2 for an analysis of this decision. It should be noted that the implementation of these two verbs was completed within the first month of coding; making the MERGE verb successively more functional took all the remaining time, roughly 4 months of work.

3.3.3 How to paste two slices together: the MERGE verb

Given several processors working individually at extracting slices of a layout, there must be a way to get their combined output to look cohesive. That is, we must transform the output so that it is functionally indistinguishable from what normal IV would have made. This duty was relegated to the MERGE verb. The MERGE verb requests the numbers of two adjacent slices, and complains if the numbers are not consecutive. It is not possible or meaningful to attempt to merge slices 1 and 4 of a chip that was quartered by the SPLIT command. The MERGE facility reads in the output files from the two extractions and constructs output files that represent what would have been derived from the extraction of the combined slice.

I call the MERGE facility a binary merge because the slices are pieced together like a binary tree. Originally I wanted to make the merging process computable at run time, after the extraction has taken place. It seemed intuitively better to perform a merge on any two adjacent segments than to specify the order of the merge operations. Each segment must be merged with both of its neighbors eventually, so why not let it be merged with which

ever neighbor is ready to be merged first?

Limitations in EPIC, which preclude runtime decisions of precedence, coerced me into opting for a binary merge. Additionally, I found a set of tasks and executions times that make a dynamic decision process worse than a rigid, binary merge. The comparison of a dynamic decision process to a static decision process is shown in figures 3-3 and 3-4.

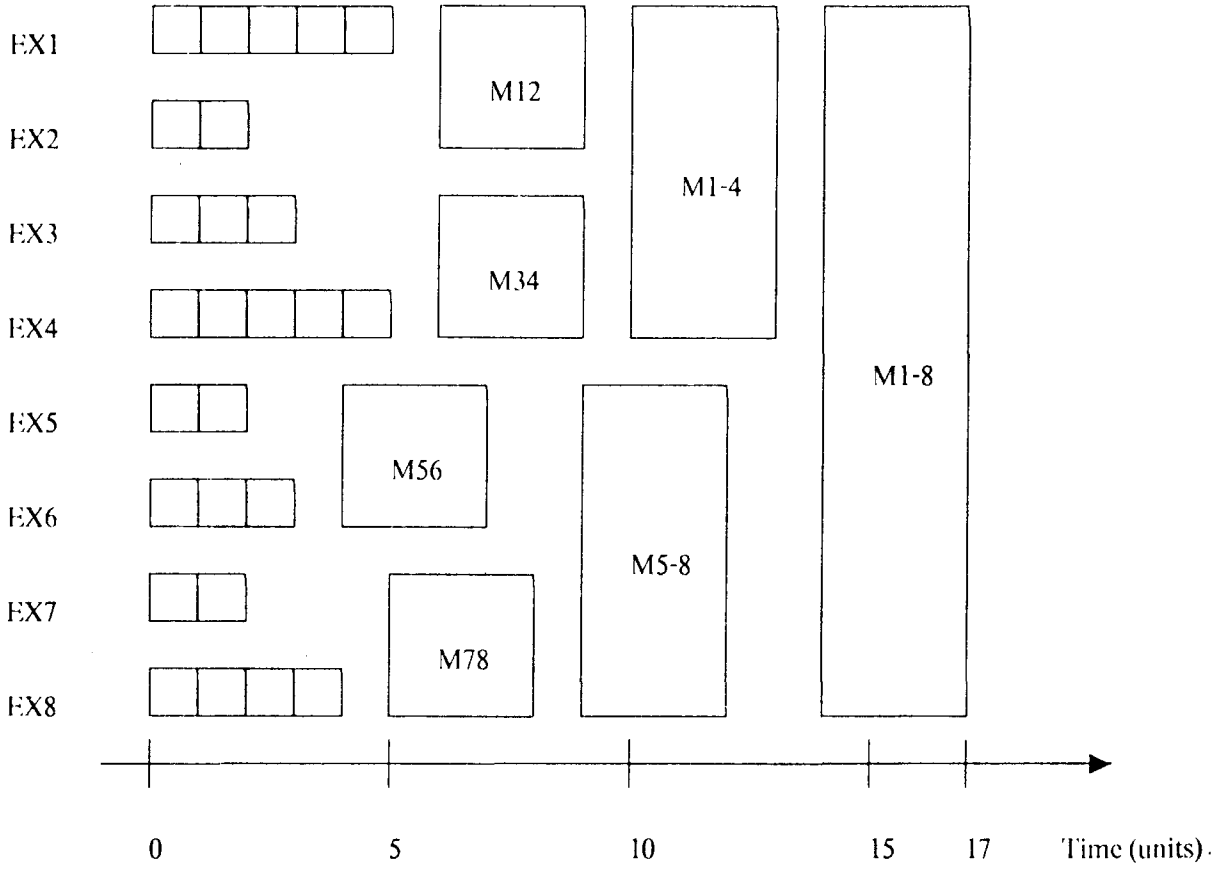


Figure 3-3: The merge ordering is made before the run.

In a chip split into a power of 2 slices, the merge functions exactly like a single-elimination tournament. In a chip split into some other number of segments, the odd segments are merged together where possible, and are unprocessed until needed (figure

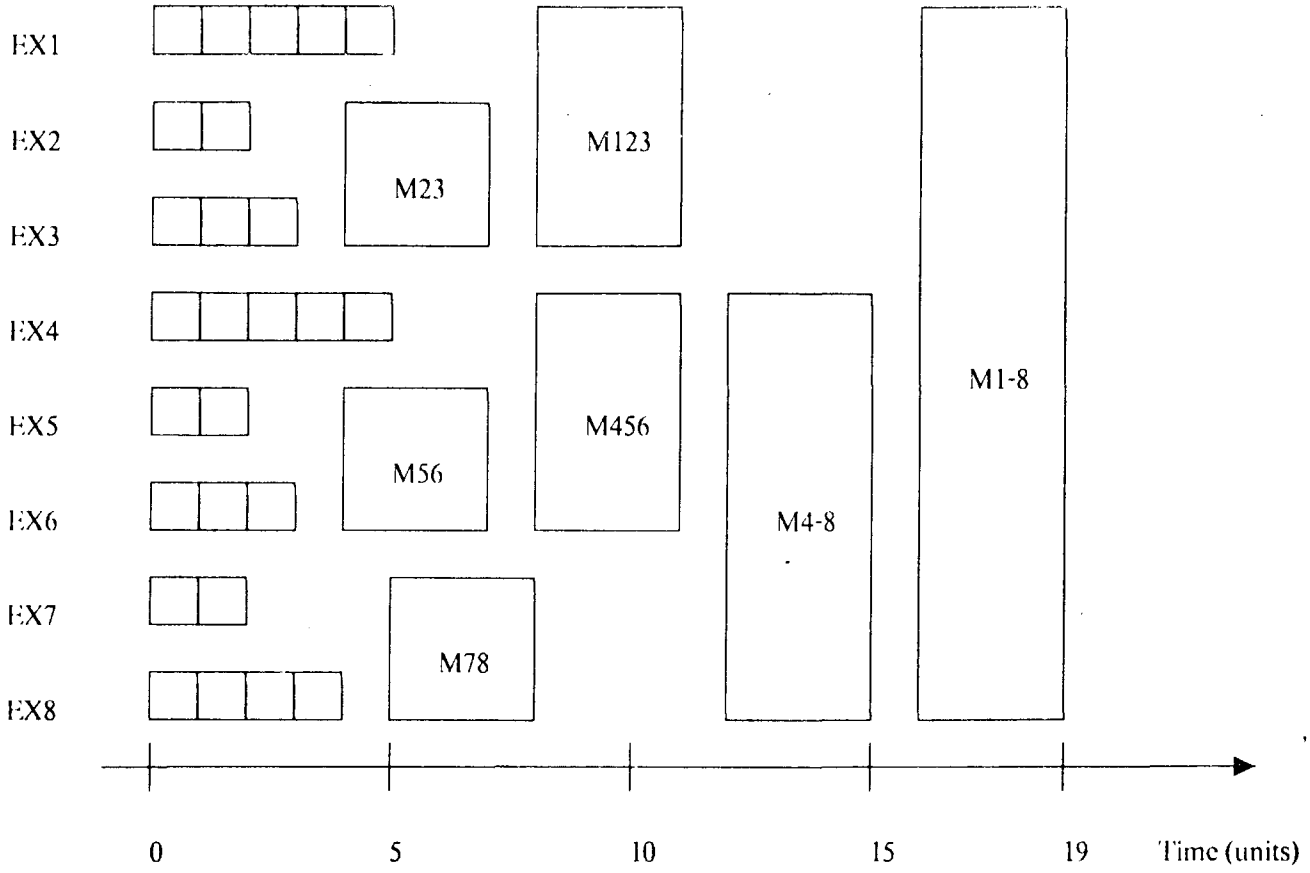


Figure 3-4: The decision to merge is made during the run.

3-5). This is similar to the wild-card playoff in the National Football League. The order or rules used to arrange the merges would only have mattered in the case that I had written a routine to generate the ECF from the MACE command line. Since the command line interface was not written, the ECF generator was not written either. The test cases I ran under EPIC used hand-written ECFs. Consult section 6.2.3 for the details.

3.3.4 The aborted OUTPUT verb

Originally I had the EXTRACT verb output a wirelist, but I later decided to avoid the creation of the wirelist file when MACE was extracting a slice. The reason the wirelist

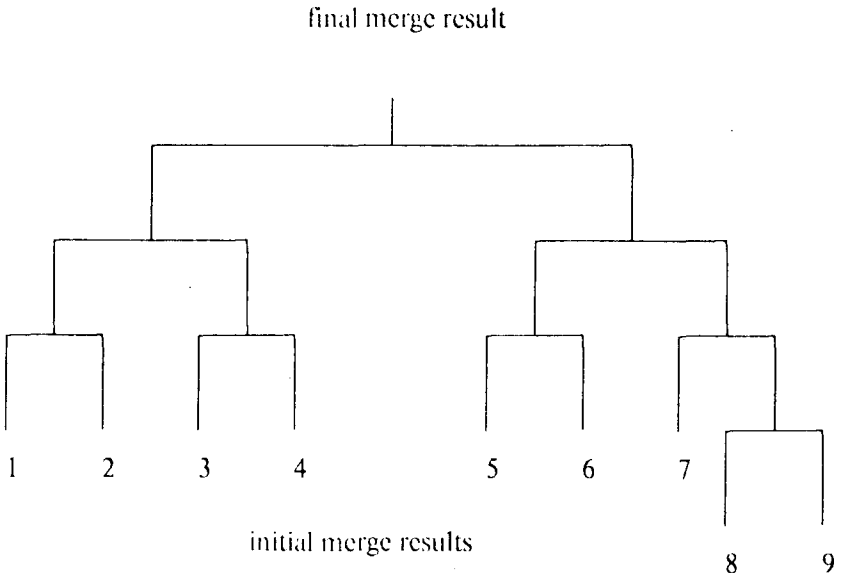


Figure 3-5: How the merge phase works with an odd number of slices.

is unneeded is that all the information contained in the wirelist is in the XND file. In fact, both the wirelist and XND file are generated from the same data structure. At first I had a separate MACE verb OUTPUT, which reads in the XND file corresponding to the whole chip and creates the wirelist. Later I realized that this verb was unnecessary. At some point I realized that the MBB information must be propagated throughout the process, and in any case would certainly be available at merge time. If the *window* information about the size of the slice is available also, the MERGE facility can decide when a wirelist is called for. The window data for a slice is the MBB of the slice. The window data for a slice merged from two adjacent smaller slices is the MBB of the combined slice. When the window data of the slice being merged is equal to the MBB of the whole chip, the MERGE facility knows it is performing the final merge.

The design decisions made during implementation of the MERGE verb are sufficiently complex that to explain them fully requires an entire chapter.

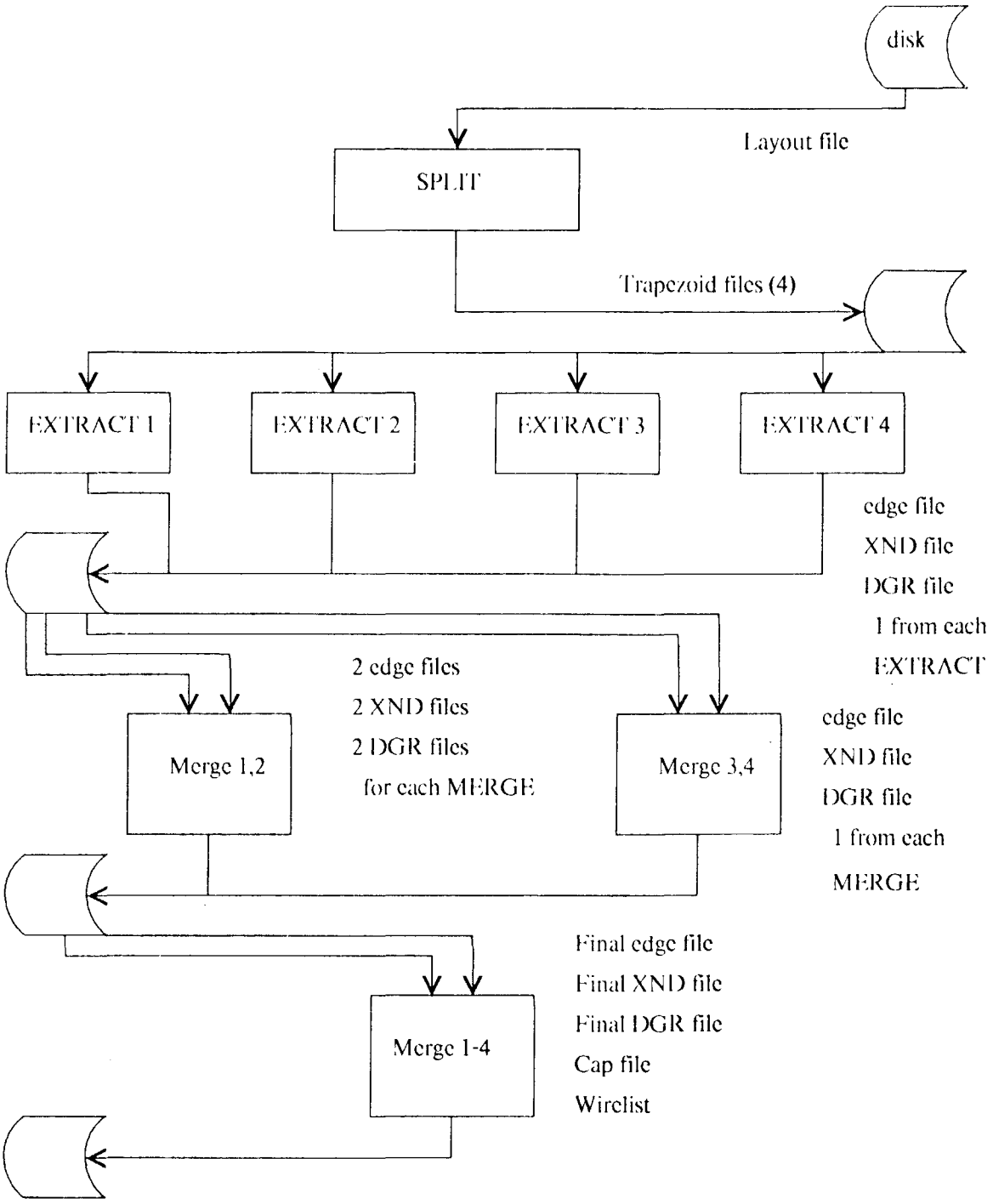


Figure 3-6: MACE data flow, in exact detail, for layout split into 4 pieces

Chapter Four

The merging process and the MERGE verb

The merging process was not well understood at the beginning. Its charter was ill-defined, roughly "do whatever is necessary to transform two slices' output into one meaningful whole." The concept of "whatever is necessary" changed as my understanding of IV's output formats grew. At first I believed only the edge and XND files necessary. However, at some point I realized that the DGR file was also needed because device geometry was required to compute device size properly. Device size is an important datum to process correctly because

- the simulator that reads IV's output uses device size to compute rise and fall times of the devices, and
- the wirelist compare facility reports device size mismatch errors when the device size in the schematic wirelist does not agree with the device size that IV renders.

The rest of the chapter is organized as follows. A high level description of the MERGE algorithm comprises section 4.1. The major features from the algorithm are discussed in the subsequent sections. The different ways a device can be victimized by a cutline are illustrated in section 4.5.

4.1 The MERGE algorithm

The purpose of the merge phase is to paste two slices together. At a very high level, the steps involved are to:

1. compute the connectivity between the two slices,
2. rename the layout structures to approximate IV's output, and

3. merge those structures judged to be split by the cutline.

Connectivity is discussed in section 4.2, renaming in section 4.3, and structure merging in section 4.4. The parts of section 4.4 deal with the individual types of data that must be merged.

4.2 Connectivity

Connectivity was computed by analyzing the edge files. The polygons abutting the cutline have edges written to the edge file. Although the slices have four sides each, corresponding to top, bottom, left, and right, only the bottom edge of the top and the top edge of the bottom are involved in the sewing process. For each edge record of the bottom border of the top slice, the list of edge records of the top border of the bottom slice is scanned. A match is made if there are two edges made on the same layer that have a nonzero overlap, signifying that the two edges belong to the same electrical node. The match records are used to rename and to merge the set of nodes and devices in the two slices, as detailed in following two sections.

4.3 Renaming

4.3.1 Motivation

Before explaining how the various structures in the slices are renamed, it is best to explain why structures are renamed. Nodes can be named by the user with labels in the layout. If a node is not labeled by the user, it is given an internal identifier for IV's use. The user cannot label devices, so each device is given an internal identifier. The labels are assigned during the extract phase. Thus, in a normal IV run, the unnamed nodes are numbered consecutively starting at 1; in MACE, the unnamed nodes are numbered starting from 1 at each cutline. The node names are very probably overlapping, since there are many cutlines.

If during an extraction IV discovers two unnamed nodes that connect, for example, nodes 7 and 9, node 9 would become node 7, and the number 9 would not be reused. It was an early goal of the project to preserve the general node naming order, including these "holes". I was initially unclear of the importance of the holes and attempted to emulate as closely as possible the IV node naming scheme. An alternative would have been to construct a node name by having a slice-dependent prefix plus the internal node number. This would make all node names unique, with the loss of only a small portion of information. Any tool that deals with node names symbolically, such as WLC, would not be affected by this naming scheme, as long as the node names did not contain illegal characters [7]. However, I judged these names to be unpleasant to see in an output file, and did not opt for this route. Another possible alternative would have been to assign these slice-unique names, but at the end to renumber the unnamed nodes consecutively from 1 for all nodes in the chip. This renumbering operation could be done very conveniently at the end of the merging, when the wirelist and final output files are being created. However, this approach was rejected because it destroyed the information about the holes.

4.3.2 Mechanics

With the set of matches derived from the previous stage, the collection of nodes from both slices can be named appropriately. The precedence by which IV joins together two nodes that are electrically the same is illustrated in table 4-1. MACE follows the same rules for conflict resolution in node names. Thus, not only must node numbers propagate from top to bottom, but labels must cross the cutline in both directions.

In order to approximate IV's node naming strategy, the node names of the unnamed (local) nodes from the bottom slice are adjusted. The unique naming routine finds the highest internal node identifier, and uses it to change the bottom slice's local node names. If the highest numbered node from the top is 44, local nodes in the bottom numbered 3, 25, and 29 become 47, 69, and 73 respectively. Since IV always assigns positive integers to nodes, the nodes named by MACE will be in an order similar to what IV would compute.

Table 4-1: Rules for resolution of node name conflicts

```
Unlabeled node on top + unlabeled node on bottom:
        bottom node takes top node's label

Unlabeled node on top + labeled node on bottom:
        top node takes bottom node's label

Labeled node on top + unlabeled node on bottom:
        bottom node takes top node's label

Labeled nodes on top and on bottom:
        If the labels are the same, ok
        Else it is a short.
```

The result of renaming is that the nodes from the two slices, considered as one group, are correctly named in the context of the merged slice. The holes are preserved with this approach.

Device naming is handled similarly, but more simply. When two device records are judged to be different perspectives of the same device, the device names are resolved. Since the user cannot assign a name to a device, all names are given by IV. The resolution strategy is one rule: the device on the bottom assumes the name of the device on the top. Compared to determining when a node crosses a cutline, determining when a device that touches the cutline is the same as one touching the cutline from the other side is a bit trickier. A node is single polygon; a device is a set of overlapping polygons. A node only has one name and one layer; a device has several nodes and several layers. To rename a device, one must check the constituent nodes with the set of node names in the merged slice. This set of node names is possibly different than the names of the device's nodes. Then, the devices themselves can be renamed with increasing indices to reflect the larger context of the merged slice.

4.4 Merging the structures

The structures that are to be merged consist of edges, nodes, devices, and, although I did not know it at the beginning of the MERGE implementation, the DGR records.

Edges

Edge merging is a problem of symbol manipulation. Edges from the two interior borders vanish when the two slices become one. Edges on the left and right borders of the slices must be checked to see if they connect to edges on the same layer on the other side of the cutline. After the edges are merged, the names of the nodes to which they are connected are checked to make sure they are consistent with the node names determined previously. All edges that appear in the merged edge file need the name of the associated node checked with the node names of the merged slice.

Nodes

Nodes can have status flags associated with them. These must be propagated into the node records for the merged slice. This minor adjustment, along with the connectivity computation and the renaming, above, suffices for merging nodes.

Devices

Some of the attributes of a device are easy to compute: names, types, and terminals are some of these. There is, however, one minor difficulty. Devices have a flag called "touches-boundary" (TB), which IV sets when it notices that the channel touches the boundary of the chip (or slice in MACE). If no device has the TB flag set, the whole DGR phase is avoided, since all devices are local to their respective slice. Thus, correct processing of the TB flag is essential.

However, each slice has four edges in the current MACE. For the device whose channel lies in the corner of the slice, touching two edges, the TB flag is set as if it only touches one edge. The TB flag alone cannot reliably encode the channel's position relative to the boundary. The problem I identified was that a device that is large in the y-

dimension could run from top to bottom of the slice, yet when it is merged on one side, the TB flag is cleared. The fact that it still touches a boundary is lost. I did not correct this deficiency, because it resulted from IV's inability to encode the information. With suitably thick slices this problem could be avoided.

There are other attributes that make merging complex. The merged device's area, length, and width, for instance, are not immediately evident from the two component device records. In order to examine the device merging more fully, we must first examine the different scenarios of device splitting.

Device geometry

Device geometry merging was extremely complex, and was not solved in the general case. Over 800 lines of source code were devoted to merging device geometry alone. DGR data are stored in non-ASCII format. The DGR files contain records for each device. Within the record there may be one or more polygon sections. A polygon section contains a number that indicates how many points follow, and then that many points, stored in ordered pairs.

I first attended to channels that

- had only one polygon in the top DGR and only one in the bottom DGR;
- had arbitrary numbers of points in the polygon;
- had no embedded holes in the channel;
- had the polygon touch the cutline in exactly two points.

. Some varieties of split channels are shown in figure 4-1. Note that not all shapes are meaningful in the physical realm. From a physical standpoint there are problems if the different pieces of the channel are not all the same length. My solutions were strictly geometric. Devices with holes in them are used to generate a high Width to Length ratio within a small area.

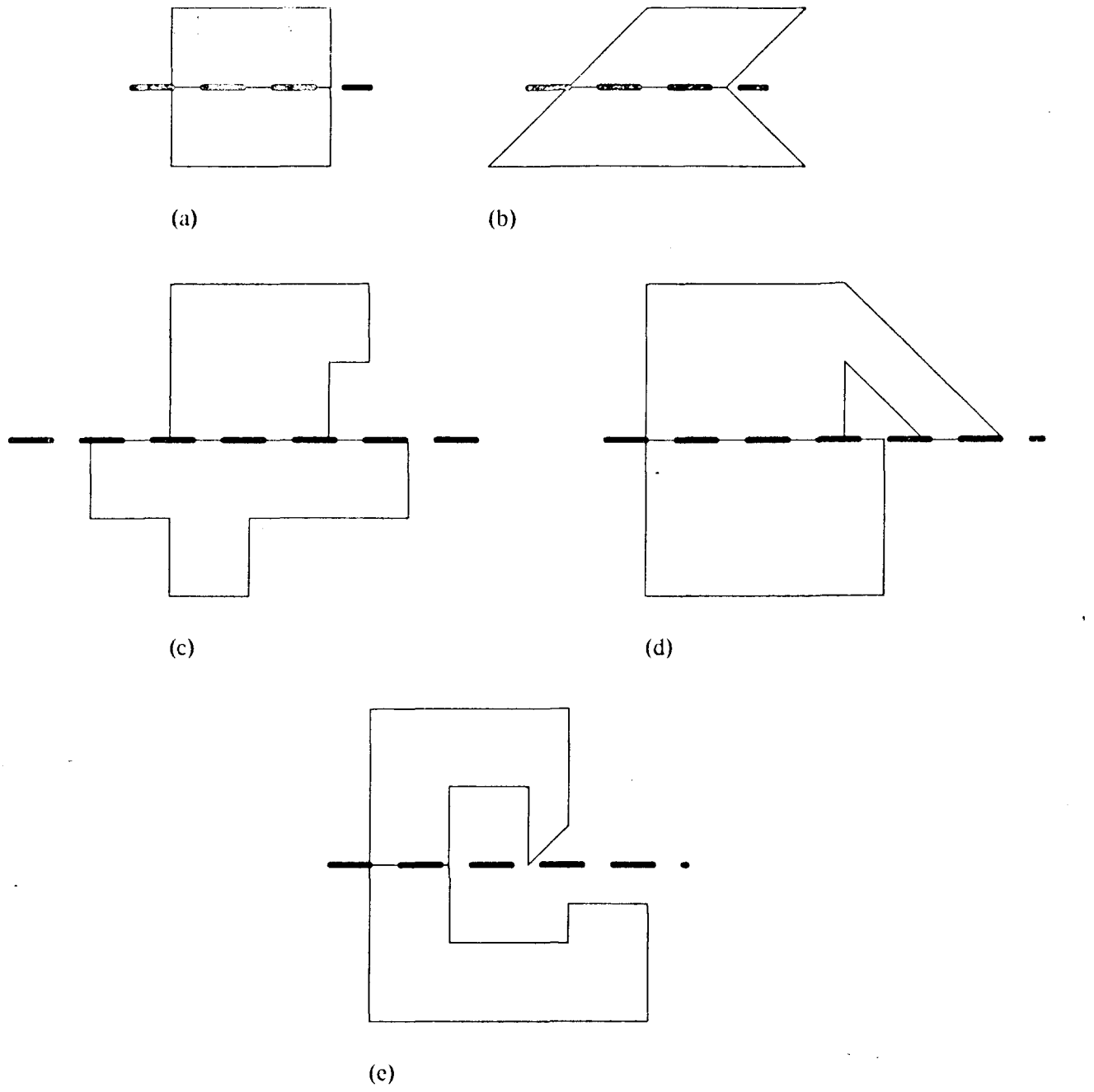


Figure 4-1: Merge possibilities: types (a)-(c) were handled; types (d) and (e) were not.

Toward the end, I put more complex device geometries lower on the priority list than

other tasks, and only devices of this type had their geometry correctly processed. The most important issue toward the end was how to handle a class of split devices, as detailed in the next section.

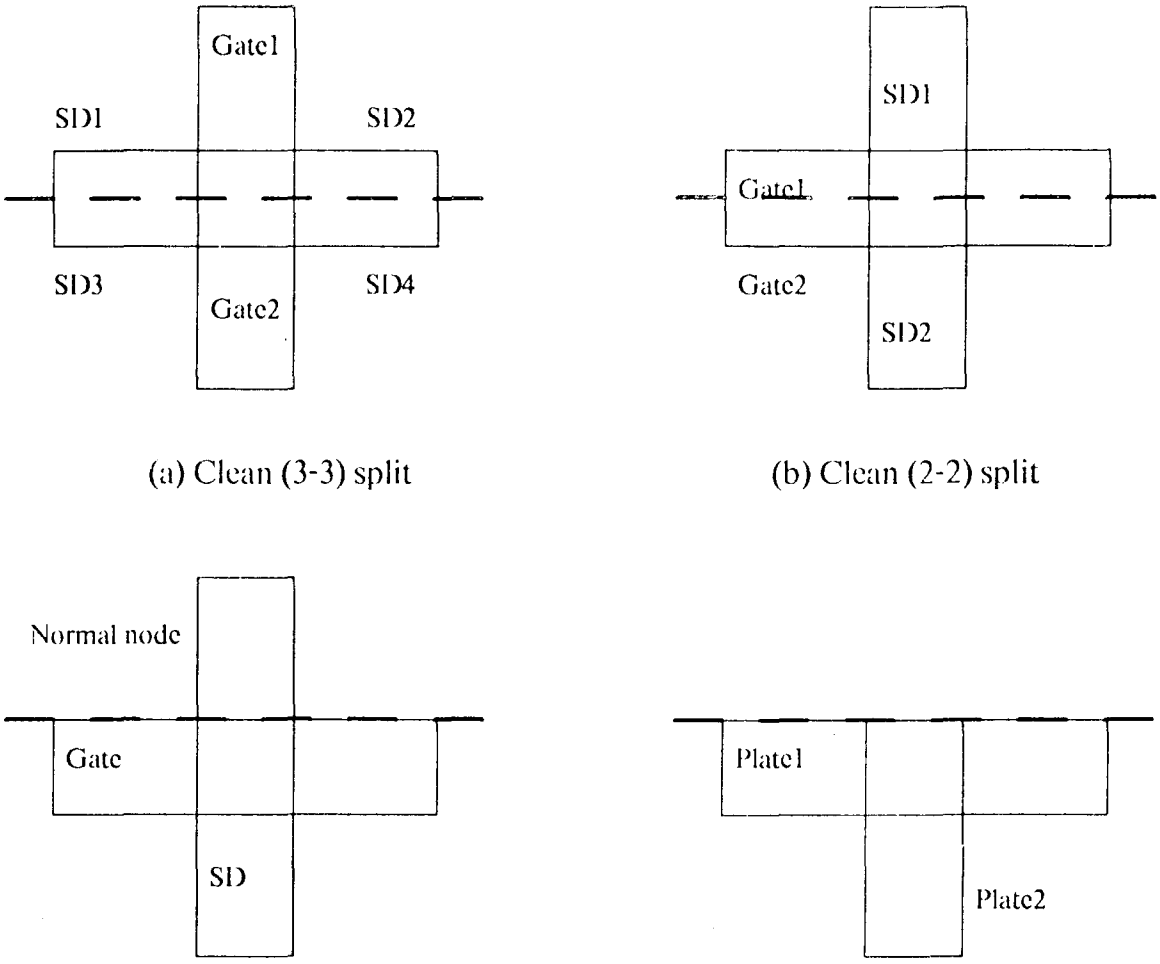
4.5 Split devices

When a cutline is arbitrarily imposed on layout, the structures split into pieces can get arbitrarily complex. Specifically, when a device is split, it can be split in several ways. I used the terminology in table 4-2 to describe devices in a layout across which cutlines had been imposed. Figure 4-2 shows what some of these cases look like.

Table 4-2: Different split device types

- unsplit devices: also called local devices. These devices have the channel not touching the cutline.
- cleanly split devices: also called a 3-3 split, because there are 3 terminals on both sides of the cutline. On both sides of the cutline, there is a recognized device.
- uncleanly split devices; and
- touching MOS capacitors.

For unsplit devices, only a single processor running extract recognizes the device. MERGE merely propagates the information, which is correct except for the global naming. For a cleanly split device, both processors running extract recognize that a device exists. Merging the device is possible because the list of nodes that cross the cutline includes the gate and both source-drain nodes in the two devices. Another form of this device is the 2-2 split. In this form, each processor sees an MOS capacitor, that is, a device with a gate and one source-drain. Fortunately for MACE, the extract code writes out a device record for this type of device. In a 2-2 split, only the gate node crosses the cutline; one source-drain is



(a) Clean (3-3) split

(b) Clean (2-2) split

(c) Unclean (1-2) split

(d) MOS Capacitor touching cutline

Figure 4-2: Split device types

above the cutline, the other source-drain is below. In this case, both device records are deleted and a new record created, with the proper names of the three nodes included. An uncleanly split device is a true *bete noire*. From one side, the device looks like an MOS capacitor. From the other side, there is no gate node to comprise a device. This happens when the cutline coincides with a horizontal edge of the gate node. Thus, when the previously restored structures are consulted, there is no matching device on the other side.

Recognition of this device is equivalent to differentiating between the following two cases.

- There is material on the other side that normal IV would have considered the second source-drain. The device is a normal, three terminal transistor.
- There is no material on the other side of the cutline. IV would have reported the device as an MOS capacitor. The device is a two-terminal capacitor.

To recognize an uncleanly split device would require making access to the data structure that represents the definition of the process technology, deciding which layer or layers would complete a device, and then searching the opposite edge list for such a layer. Although it may be difficult to believe, all the work in MACE was strictly geometric symbol manipulation, except for this recognition phase. Cleanly split devices, local devices, and MOS capacitors were handled properly in MACE, with device sizes not accurately calculated for split devices. Uncleanly split devices were not handled.

Chapter Five

Testing: Strategy, Cases, Results

This chapter analyzes the algorithm in MACE, especially parts of the MERGE algorithm. It also discusses the testing strategy of MACE, along with brief test results. An explanation of the performance is given.

5.1 Algorithm analysis

If there are N devices in the chip, then it is expected that there are at most $O(\sqrt{N})$ devices in any horizontal line. Similarly, if there are k cutlines drawn, there will be $O(k \cdot \sqrt{N})$ items that abut the cutlines. Also, k cutlines imply running MERGE k-1 times, which will require, if there are sufficient processors, $\lceil \log_2(k) \rceil$ stages of the processors running MERGE. Thus, we can complete the merge phase in polynomial space in log time.

In cases where there were $O(\sqrt{N})$ structures along the two sides of the cutline, searches for matches, as in the edge sewing phase, can be done naively in $O(N)$ time, and could be done cleverly in $O(\sqrt{N} \cdot \log_2(N))$ time by sorting them before searching. As a worst case, we would expect $O(\sqrt{N})$ devices to cross a given cutline, but with a bit of adjustment, especially in a regular layout, as might be found in a gate array chip, we might expect to do quite a bit better, avoiding split devices entirely.

The edge merging is proportional to the size of the side edges, or $O(\sqrt{N} \div k)$. If the merge algorithm does not have to rename the items inside the slice, then its phases are proportional either to the length of the cutline or to the height of the slice.

5.2 Testing strategy

During the development of MACE, I took the approach of testing and building feature by feature. Since VLSI chips often have complex geometry, it is difficult to say when all cases for testing a certain feature have been exhausted. It is also difficult, in the absence of a mass of statistical data, to judge which geometrical cases are more frequent than others. I often encountered the situation at which a certain MACE feature worked over a set of layout configurations, but there were identifiable configurations not in the set. I often did not know whether it was important to be able to deal correctly with these configurations.

One qualification for IV to work properly is that the layout be free of design rule errors. If there are design rule errors, IV's behavior is unpredictable. It might function correctly, and it might not. Since there are a few unusual cases that IV does not handle correctly, I decided not to expect MACE to behave better than IV. MACE's goal was emulation of IV, whether the behavior was correct or not.

5.3 Performance

I generated both two-way comparisons and three-way comparisons. Two-way comparisons involved serial running of IV, the existing tool, and sequential running of MACE. In a serial IV run, a command file calls on IV to extract the circuit and produce the edge file. The other output forms of the extraction are produced by default. The execution takes place on a single processor. In the sequential MACE condition, command file calls on MACE a number of times to perform the individual tasks: splitting, extracting, and merging. The execution takes place on a single processor. The three-way comparisons include these two conditions and the combination of MACE and EPIC as the third condition. A command file creates the EPIC master and starts the computation. Execution takes place across a group of processors all of which have the same hardware and memory size.

5.4 Results

In order to classify the test as a success or failure, the wirelists were checked against each other using WLC. The results in condensed form are shown in table 5-1. For the unexpurgated results, consult Appendix II.

Table 5-1: Brief performance results

Hardware characteristics:
Node1: VAX-11/780
Node2: VAX-11/780
Node3: VAX-11/780
Node4: VAX-11/785
Node5: VAX 8600

Test 1: Comparison of serial IV with sequential MACE on Node5, block 1.

IV:	CPU time: 01:03(min:sec)	elapsed time: 01:16
MACE:	CPU time: 01:32	elapsed time: 02:38

Test 2: Comparison of serial IV with sequential MACE on Node5, block 2.

IV:	CPU time: 17:54	elapsed time: 18:23
MACE:	CPU time: 24:20	elapsed time: 32:25

Test 3: Three-way comparison on Node1, Node2, and Node3, block 1.

IV:	CPU time: 04:04	elapsed time: 04:30
MACE:	CPU time: 05:39	elapsed time: 07:24
MACE+EPIC:	CPU time: 05:36	elapsed time: 05:15

Test 4: Three-way comparison on Node1, Node2, and Node3, block 2.

IV:	CPU time: 1:05:41	elapsed time: 1:09:31
MACE:	CPU time: 1:27:31	elapsed time: 1:52:08
MACE+EPIC:	CPU time: 1:25:32	elapsed time: 1:17:00

Note: CPU time in the MACE+EPIC condition is that consumed by all processors.

I compared CPU and elapsed time for the three conditions on test cases that were large enough to make the fixed costs insignificant. Sequential MACE required more CPU and elapsed time than serial IV. MACE with EPIC required roughly the same aggregate

CPU time as Sequential MACE. The elapsed time on the MACE with EPIC runs was larger than for serial IV.

5.5 Explanation

It has been shown that IV is a memory limited CAD tool. It builds such complex data structures, tending to fill all available memory, that performance analysis shows memory to be the bottleneck [4].

5.5.1 Serial IV vs. sequential MACE

The serial IV case required only one image activation; sequential MACE and MACE with EPIC require separate image activations. Although the test cases were large, the MACE runs required several more image activations that the IV run. Also, serial IV keeps all its data in main memory, while MACE has the additional overhead of having to write its data out to enable communication. Another weakness in MACE is that MACE contains an abundance of consistency checks. At various phases, MACE traces its linklists to ensure that connectivity is preserved. During the development phase this was crucial, because frequent examination tended to limit the number of routines that were suspected of corrupting the data structures. Infrequent checks would have resulted in a more difficult debugging task. The removal of these checks would speed up execution somewhat, but not significantly. Were MACE to be used for production, as a released tool to the VLSI designers, it would most likely have these checks removed.

5.5.2 Where MACE spends time

Without performing detailed analysis on exactly where MACE spends time, exact figures are unknown. However, some estimates can be given. Of the function performed by the split verb, some is mirrored by IV, the rest is pure overhead. The portion that IV performs is the reading of the layout file and flattening of the data. The overhead is the

writing out of the iotraps. In the extract phase, there is some overhead involved in reading in the iotraps and writing out intermediate extraction files, but for the most part, the CPU times are very close to those of IV. The merge phase is entirely overhead, with the exception of the extremely small portion in which the wirelist and error list are generated.

5.5.3 MACE + EPIC: any advantage?

The running of MACE with EPIC has not yet revealed an advantage over serial IV. Often the elapsed (*turnaround*) time was only slightly more than serial IV. However, it is clear from the data that working within EPIC tends to reduce the damage done by MACE's extra phases. In a condition of uniformly dense layout, and a cleanly functioning MERGE, it might be possible to bring the turnaround time below that of serial IV. Although the evidence seems to suggest that linear speedup is a goal far afield, it is reasonable to expect that with more processors devoted to the extraction and with a streamlined MERGE verb, the turnaround time could be brought much lower than that required with IV. From extrapolating from one of the test cases, I estimate that in a large chip extracted by 8 processors, the SPLIT time should be about 10% of the IV turnaround time, the EXTRACT time should be about 15%, and the time to perform a 3-stage binary merge should be about 40%, for an overall turnaround time of 65% that of IV.

5.6 An unanswered question about efficiency

For the sake of ease in reporting results, the runs performed with EPIC used the same type of processor, with the same amount of memory. Even though EPIC does not require slaves to be on the same VAXcluster, I only used processors on one VAXcluster due to availability. Were the goal to minimize turnaround time, one would not care which hardware was used. One would prefer using the fastest processor possible to perform as much as possible. But how does one measure speedup in the heterogeneous environment? Performance analysis is a tricky enough field without compounding the issues by introducing different processor types, memory sizes, and loading factors into the melee.

Different CPU types are VAX-11/780, VAX-11/785, VAX 8600, and MicroVax II, for example. Given this situation, the term *utilization* becomes ill-defined for an EPIC run with one slave a VAX 8800, another slave a VAX 8500, another slave a MicroVax II, and still another a VAX-11/780.

Chapter Six

Future Research Opportunities

6.1 Missing features

MACE did not achieve proficiency at handling all the desired layout features correctly. The uncleanly split device was not handled, although the hooks were put in to facilitate its insertion. Device size calculation for merged devices were consistently different from IV's figures. Digital Equipment Corporation uses a more complex model for device sizes than merely average length times average width. Because of this formula, these data were difficult to reproduce. Instead of making the area the product of length and width, area is computed by adding up the areas of the trapezoids in the channel. One of the two factors is maintained, and the other is *derived* when the device is complete. Without introducing dubious correction factors into the code, the size errors could not be eliminated for severed devices. Also, MACE's input method did not allow specified negative coordinate values.

6.2 Extensions to MACE

6.2.1 Alternate splitting modes

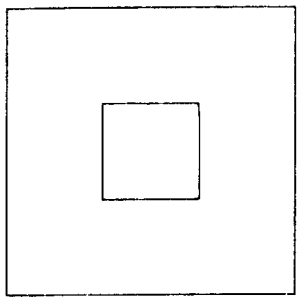
For testing, the variable SPLIT mode was essential to enable dropping a cutline on top of a feature in question. If one wanted to minimize the variance in the extraction times, one might like to have a method of splitting the layout that would evenly divide the number of polygons in each slice. Such a scheme would produce slices with variable heights. If extraction time is related to the number of polygons, this might be a good first step to achieving uniformity of the size of the extraction task. Note from one of the test cases that the times to perform the extraction were widely varying.

6.2.2 Automatic Testing Tools

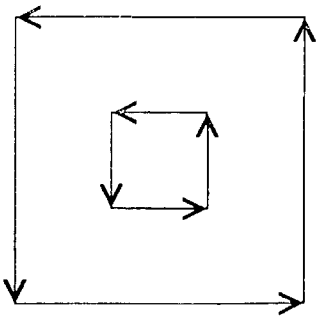
One area that could have benefitted from some work is that of automatic testing tools. Few things are more frustrating than to generate some test results and not to know if they are equivalent to the expected results. Wirelist Compare is an excellent tool to establish equivalence of two wirelists [7]. However, there are other output formats of IV that do not have such a sophisticated checker. The edge file format and XND file format in particular are two for which equivalence could have been established by an easily built postprocessor. A single tool to compare equivalence of edge and XND files would work as follows:

1. Read in the two edge files.
2. Sort edges into two data structures, one for the IV-generated structure, one for the MACE generated structure
3. Further divide the edges into left, right, top, and bottom edges.
4. Considering each area (left, right, top, or bottom) alone, attempt to build a mapping function, f (iv-node-name) = mace-node-name. Constructing the function and its inverse edge by edge would be straightforward.
5. Report any edges that were did not exactly match edges in the other structure.
6. Read in the XND files into two new structures.
7. Using the same mapping function and its inverse derived above, attempt to reconcile the nodes listed in the XND file, reporting errors as above.
8. Build a device-name mapping. (perhaps this can be done by a postprocessor to WLC)
9. Reconcile the device data with respect to gate name, source/drain names, location, device type, and device size.

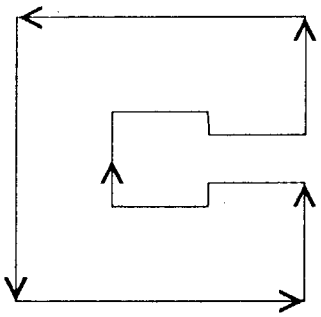
Device geometry, because of its inscrutable format, is difficult to check. Especially with more complex channel shapes, equivalence is not the same as coordinate-for-coordinate equality. The lack of a canonical representation for channel shapes and of



(a) Donut case. Center is empty



(b) Representation 1: two concentric squares



(c) Representation 2: limiting case of a 'C'

Figure 6-1: Two representations for a shape with a hole

procedures to eliminate interior lines makes the establishment of such equivalence an involved task. In figure 6-1, the actual layout is in part (a). One way to represent this is with two polygons, as in part (b). Another way to represent this is with one polygon whose edges meet, as can be extrapolated from the closing arms of the C in part (c). When IV encounters a device such as this, it writes the DGR in form (b). When thinking about merging portions of channels, I realized that my algorithm would represent it in form (c), and that deriving form (b) would be extremely difficult.

6.2.3 Glamorous yet functional user interface

MACE performs input by prompting the user for a chip name, a verb, qualifiers, and so on, one to a line. This method was the simplest, but investing the time to develop a command reader with the CDU (Command Definition Utility) would be a functional improvement as well [20]. The CDU provides for the description of the verbs available within a subsystem such as MACE. It is easy to specify that a verb such as SPLIT has two possible qualifiers, /FIXED and /VARIABLE, that the two qualifiers are mutually exclusive, and so on. In particular, it is possible to specify that /FIXED takes one argument, the number of slices to divide the layout into, and that /VARIABLE takes a list of y coordinates. These arguments can be passed as formal parameters to user-specified routines. CDU puts glamor on the front end, where it belongs, but it also provides a good way of command processing. What is apparent to the user would be a difference between these two dialogues:

```
$mace := $mace$dir:mace
$mace
MACE> split
Block to split: block1
Fixed or variable: v
Enter coordinate, or 0 when finished: 7600
Enter coordinate, or 0 when finished: 4850
Enter coordinate, or 0 when finished: 1040
Enter coordinate, or 0 when finished: 0
...
```

as opposed to

```
$mace := $mace$dir:mace
$mace split /fixed=4 block1
...
$mace split /var=(7600, 4850, 1040, 0, -2500) v2yscan
...
```

In the first, the task takes several lines to specify; in the second, any MACE task can be specified with one line. It should be noted that this approach would solve one liability with the approach implemented in MACE. MACE's SPLIT verb does not permit a named y coordinate to be less than 0. It also rejects out of order values. With the CDU interface, all values are available to the routine named in the command description, and could be sorted

there. Further, negative values, which have meaning in some chips, are acceptable with this method.

6.2.4 Automatic ECF generation

With the benefit of the CDU, MACE tasks become single command lines, instead of requiring a whole command file. Of course, a command file may still be needed if there are setup steps needed, or if there are steps to be taken after the task. The CDU mentioned above would enable the user to interact with MACE more efficiently. A new tool could be built as a new front end, that would prompt for some extraction parameters, and build the ECF and command files needed to run MACE under EPIC. This new tool would be similar to the module Marantz built to parse the rules file to generate the ECF [8].

6.2.5 Capacitance calculation

It was not a goal of this research to calculate capacitance with IV's accuracy. Indeed, given the problems representing complex shapes, this would have been very difficult. However, in another environment, the multiprocessing extractor need not be excluded from calculating capacitance. If the tool can correctly determine the shapes of the nodes, then such calculations should be straightforward.

6.3 Design decisions better redone

6.3.1 Static computability of merge might not be best

Recall section 3.3.3, in which dynamic computability of merging was judged not to be an advantage. The argument about static or dynamic computing of which slices to merge holds for a homogeneous computing environment. It is a different question entirely for the heterogeneous environment. Assume the slices are of roughly the same complexity, as measured subjectively by number of nodes, devices, polygons, and so on. Then, the

variance of execution times will be greater for a varying set of processor types than it will be for all identical processors. If the processors assigned each to extract one slice have power that ranges over two orders of magnitude, then the execution times are expected to range over two orders of magnitude also. Observe from Appendix II that in larger chips, the time required to merge two slices becomes small relative to the time required to extract a single slice. Given two slices, which ordinarily would not be merged in the first merge stage, that have passed the extraction phase, it might therefore be possible to complete the merge operation before either of the merges called for in the static strategy would be possible. Having layout of nonuniform complexity only blurs the issue more. Every factor that tends to increase the variance of the extraction times, such as processor memory, processor speed, and variable segment height, speaks for the accommodation of dynamic computing of the merge pattern.

6.3.2 Unified output format

I believe that using the existing IV formats was the correct decision. However, there could be one slight modification that would streamline the process. Rather than using the various IV output formats in many separate disk files, an alternative is to use a unified file format, that would store several files' data. The unified output format would be an output of the extract verb and an input of the merge verb. The merge verb would then merely have to read one file per input, and create one output file as its result. As before, the final merge operation could know to create from the single output file the additional reports, such as errors, capacitance charts, and wirelists. The advantages to be had by using this format include the following:

1. There is less overhead on file transfer between very loosely coupled processors.

2. Confusion from issues of file protection and file naming are minimized.

3. Directories of files involved in a merge operation are smaller, more comprehensible.

4. The ECF for a MACE circuit extraction run will be smaller, because there will

be fewer inputs and outputs of the various merge stages, thus is quicker to write and read for the EPIC modules that use ECF.

5. Validation of the unified format can still be done by separate tools in parallel if each validation tool opens the unified output file for READ access (with no intent to modify).

The identifiable disadvantages include:

1. Some information might be lost due to only one file organization. When different data types use different file organizations, for example, to facilitate reading by computer or by human, these decisions would have to be made in the context of a predetermined unified format.

2. Maintainers would not have the freedom to choose any file format when adding a new data type.

3. With separate files, it is immediately obvious from looking at the directory if the merged edge file was produced. With a unified output format, it is not obvious from looking at the directory listing if each format was dumped into the single file.

On the one hand, simplicity seems to be in favor of a single file for single data type, as in traditional IV. On a more global perspective, however, when there are many different formats for different data types, and the confusion multiplies for each different file type, creating a polyglot format might be attractive.

6.4 Modifications in the MACE and EPIC interaction

EPIC was built with the intention of scheduling independent tasks. The task processors should have no idea that there is even another processor in the universe. In this setup, slave processors cannot cooperate. An alternate paradigm is that of coworkers and bosses. The coworkers communicate as the task requires, and the boss channels the communication and controls the execution. Such a setup would require some minor rebuilding of the application to take advantage of the possibility to communicate.

However, the availability of such communication would greatly facilitate some goals, primary among which is smart cutline imposition.

6.4.1 Cut cleverly; Merge simply

Structure avoidance by arbitrarily placed horizontal cutlines is not dependable. It is an event with a certain probability of success. The imposition of many horizontal cutlines will sooner or later sever a device, and if there is no code to handle the severed device, there will be a mess. The suggestion is that by appropriate communication, cutlines could start horizontal, with the possibility of being nudged by a processor to avoid certain structures. A processor that recognizes a device touching the top boundary of its slice may decide to pull the cutline down below the device, in effect relegating the device to the duty of the processor of the adjacent slice. The device's data could be formed into a message, and sent, with the boss's aid, to the coworker in charge. The coworkers would not know who was assigned to what task, but the boss would. With a bit of work, so as to avoid the excision by both processors of two halves of a cleanly split device, MACE could be built to avoid device splitting entirely.

6.4.2 Alternate merge formats

MACE uses a binary merge format, in which two slices are merged in one task into a larger slice. Each strip has exactly two neighbors. An alternate merge method that still uses the strip decomposition method would be a bucket brigade algorithm. This algorithm would not be possible without the support for interprocess communication, but could conceivably function within the boss-coworker configuration.

Consider n processors working on n separate strips, and assume we have a reliable way to pass messages from slave to slave. On an even phase, processor 0 (P0) working on strip 0 (S0) talks to processor 1 (P1) working on strip 1 (S1). Similarly, even processors talk to their higher neighbors. The high processor, if even, remains idle. On an odd phase, P0 remains idle, and all odd numbered processors talk to their higher neighbors. Even and

odd phases alternate until message traffic stops. When message traffic stops, the memory of each processor is combined in any order to represent the circuit equivalence of the chip. The master can judge when message traffic stops by a function of the slaves' state. If a slave is computing, there might be more message traffic. If all slaves have finished computing on their own data and they have processed all of their incoming messages, then there are no more messages to be sent.

6.4.3 Messages for naming

The preceding explanation gives the general configuration of the processors. It could be used to perform any of the central MERGE functions. Messages could be used to emulate IV's naming scheme, if this is the desired method. In one phase, all labels could start flowing back and forth, until all named nodes were agreed upon by all the processors. In a second phase, the numbers for the unnamed nodes would spread out. In a third phase, the processor of the top slice would retrieve its highest node number and pass it to the neighbor. Each processor of an internal slice would read the number and adjust the nodes that are local and unnamed by this offset, then sending the new highest node number to its neighbor. To establish consistent naming among n processors would require $O(\log_2(N))$ time.

6.5 Recommendations for implementation

From the experience and difficulties encountered in the development of MACE, I have gleaned some impressions and thoughts on how to build a multi-processing circuit extractor. The observations are made both in reference to the software engineering aspects of the project and to the multiprocessing aspects of the project.

6.5.1 Software development suggestions

To ensure ease of building prototypes rapidly, consideration should be given to modular design of the components. The modules that perform input or output of a certain data type should completely insulate the realization of the format from the rest of the system. This feature, not a recent development of software engineering, is absent in the IV system. It is difficult to derive the syntax for an XND file from the code. It is even more difficult to be sure that changes to the format, which were necessary in the project, were adequately achieved without adverse side effects by the changes made to the code.

Each module should be built with its own debugging routines, for record analysis. The records IV manipulates tend to be quite large, with analysis of selected components difficult. By providing routines within each module that inspect data, scrutiny of the relevant parts is easier. This approach was taken in the modules of MACE.

Consistent module and variable naming, a unified approach to error handling, and unified approach to procedures' return values all will help the members of the development team ensure consistency of the code. Rather than have several modules in each member's style, establishing and enforcing conventions will provide greater ease to the group at the expense of ruffling each member's feathers only slightly.

6.5.2 Suggestions for multiprocessing extractors

I believe that a circuit extractor can be built that not only embodies these principles of high quality code, but also enables the system to work in a single processor or a multiprocessor environment. The computing environment assumed here is a moderately coupled multiprocessor. Several autonomous processors share memory and have a means of sending messages to each other. The notion of a task is central to the design of the system. There are several methods of breaking the task of circuit extraction of the entire chip into several smaller tasks.

Use hierarchy

One might wish the tool to take advantage of the hierarchy of the chip. In its present form, IV does not take advantage of the hierarchy; it flattens the layout information before processing it. We might wish to have the tool work at the grain of the leaf cells of the chip assembly, constructing descriptions for the cells. Then it would instantiate the descriptions, modifying them with any loose layout in the calling cell. A natural ordering of the tasks comes from the hierarchy. Possibly great savings could come from not having to perform the extraction for the same leaf cell many times. The added complication is the need to accommodate the loose layout that can overlay an instance of a cell.

Use pipelines

Just as a compiler has several phases of operation, each with a well-defined phase that transforms intermediate results, conceiving circuit extraction as a sequence of refinements on data might help. The various phases might be

- reading the layout
- flattening the layout
- connectivity calculations (contact recognition)
- device recognition (depends on the process)
- device attribute calculation (area, perimeter, length, width...)
- [capacitance/resistance calculation]
- global naming of items (such as edges, devices, nodes)
- output the data and
- output the errors in the layout.

Each task filters the data from the previous step or steps to accomplish its own task. The tasks are designed to be atomic. I don't know if this atomicity is possible, but this

might work. If the processes are not atomic, but if message passing is possible in the architecture, the tasks can be pipelined. The flattening can start before the reading is finished, but cannot complete until the reading is complete.

Use wisdom

Another approach is to make the merging as simple as possible. The simplest it could be is if appending individual results makes final results. By putting the name construction phase at the end, as has been suggested is preferable to the naming scheme within MACE, no time is spent computing and recomputing node names. In the current MACE, a local node in the bottom slice could be renamed once for each MERGE operation it is in. Also, if the cutlines are positioned cleverly, to avoid structure merging, temporary results are much closer to final results than they are in current MACE. By appropriate use of knowledge and communication, as described previously, the binary merging can be made essentially a local operation; no structure merging need occur.

6.6 Conclusions

Considering the complexity of the task, I am pleased with the results. Within the context of a large body of software, I was able to modify it to support a different mode of behavior. I was able to bring in the necessary data for the reconciliation phase. There are some drawbacks in MACE's performance: some data are not correctly calculated, such as device sizes; device geometry is not handled in some of the more complex device shapes; and not all split devices are handled. However, there is reason to believe that these features could be implemented. The project shows that there may be hope for a multiprocessing circuit extractor, and it is suggested that finer granularity would go a long way toward this goal.

The task of circuit extraction, on the surface a difficult one to mold for multiprocessing, does not present clear contraindications of this possibility. I have shown that, in spite of the nonlocal nature of the task of circuit extraction, temporary results made

on the basis of no context can be converted into results that are equivalent to what is normally obtained from extracting the whole chip. It is suggested that there are some reasons to rethink the process of circuit extraction to enable a cleaner decomposition of tasks and to provide for several multiprocessing prototypes. Tighter coupling of processors would aid in certain computation scenarios. It is further postulated that an approach similar to the one taken can bring up to a 50% speedup over the existing method.

References

[1] Adshead, H. G.
Towards VLSI Complexity: The DA Algorithm Scaling Problem: Can special DA Hardware Help?
In *Conference Proceedings*, pages 339-344. 19th ACM/IEEE Design Automation Conference, June 1982.

[2] Baker, Clark Marshall.
Artwork Analysis Tools for VLSI Circuits.
Technical Report MIT/LCS/TR-239, Massachusetts Institute Technology, 1980.

[3] Bier, George E. and Andrew R. Pleszkun.
An Algorithm for Design Rule Checking on a Multiprocessor.
In *Conference Proceedings*, pages 299-304. 22nd ACM/IEEE Design Automation Conference, June 1985.

[4] Gadoua, Mary.
Non-graphical CAD Tool Performance Testing on the MicroVAX II.
February 1986.
Internal publication of Digital Equipment Corporation.

[5] Gupta, Anoop.
ACE: A Circuit Extractor.
In *Conference Proceedings*, pages 721-725. 20th ACM/IEEE Design Automation Conference, June 1983.

[6] Kenah, Lawrence J., and Simon F. Bate.
VAX/VMS Internals and Data Structures.
Digital Press, 1984.

[7] Kodandapani, K. L. and Edward J. McGrath.
A Wirelist compare program for verifying VLSI layouts.
June 1986.
To appear in *IEEE Design and Test.*

[8] Marantz, Joshua D.
Exploiting Parallelism in VLSI CAD.
Master's thesis, Massachusetts Institute of Technology, 1986.
To appear.

[9] Mead, Carver and Lynn Conway.
Introduction to VLSI Systems.
Addison-Wesley Publishing Co., Reading, MA, 1980.

[10] Nagel, L. W. and Pederson, D. O.
SPICE (Simulation Program with Integrated Circuit Emphasis.
Technical Report ERL-M382, University of California, Berkeley, Electronics Research Laboratory, April 12 1973.

[11] Nielson, Richard D.
Algorithmically Accelerated CAD.
VLSI Systems Design VII(2):65-66, February 1986.
The author is with ECAD, Inc.

[12] Noyce, Bill.
Using VAX/VMS for Parallel Processing.
Digital Eqiupment Corporartion, Hudson LSI.
March 1986.
Seminar delivered to the Hudson Technical Seminar Series.

[13] Ousterhout, John K., Gordon T. Hamachi, Robert N. Mayo, Walter S. Scott, and George S. Taylor.
Magic: A VLSI Layout System.
In *Conference Proceedings*, pages 152-159. 21st ACM/IEEE Design Automation Conference, June 1984.

[14] Rezac, Roy R. and Leslie Turner Smith.
A simulation Engine in the Design Environment, Part 1: Normal Simulation Methodology and Results.
VLSI Design V(11):96 ff, November 1984.

[15] Tarolli, Gary M. and William J. Herman.
Hierarchical Circuit Extraction with Detailed Parasitic Capacitance.
In *Conference Proceedings*, pages 337-345. 20th ACM/IEEE Design Automation Conference, June 1983.

[16] Taylor, George S. and John K. Ousterhout.
Magic's Incremental Design-Rule Checker.
In *Conference Proceedings*, pages 160-165. 21st ACM/IEEE Design Automation Conference, June 1984.

[17] Terman, Christopher J.
A Multiprocessor Implmentation of a Logic-Level Timing Simulator.
Digital Eqiupment Corporartion, Hudson LSI.
March 1986.
Seminar delivered to the Hudson Technical Seminar Series.

[18] Turner Smith, Leslie, and Roy R. Rezac.
A simulation Engine in the Design Environment, Part 2: Fault Simulation Methodology and Results.
VLSI Design V(12):74 ff, December 1984.

[19] *Guide to VAXclusters.*
Digital Equipment Corporation, Maynard, MA, 1984.

[20] *VAX/VMS Command Definition Utility Reference Manual.*
Digital Equipment Corporation, Maynard, MA, 1984.

Appendix I

Glossary of Terms

channel the overlap area of polysilicon and diffusion for an NMOS device.

cutline an imaginary line imposed on a layout by which SPLITTING occurs. In MACE, all cutlines are horizontal.

device usually, a transistor. It usually has three terminals: a gate node and two source/drains. In the context of IV, an MOS capacitor is called a device with only two terminals, and can be generated by a three-terminal device whose two source/drains are the same electrical node.

ECF execution control file. This file instructs EPIC which tasks are part of the computation.

homogeneous describes a computing environment in which all the processors are substantially the same.

heterogeneous describes a computing environment in which all the processors are not substantially the same.

labels descriptive text associated with a node in a layout file that helps to identify for the designers' use the function of the node. Sample labels might be PHI, WRITE-ENABLE, DATA<3>, to indicate a clock signal, a control signal, or a slice of data, respectively.

Manhattan geometry layout that contains only horizontal and vertical lines. Modeled after the streets of Manhattan, which are largely box-like. *Non-Manhattan* geometry contains lines that are not horizontal or vertical, although in most cases this means oriented at some factor of 45 degrees.

MBB minimum bounding box, represents the smallest rectangle inside which all layout fits. It can be thought of as horizontal and vertical shrink wrap.

slice a rectangular portion of a chip. The width of the slice is the width of the MBB of the chip. The height of the slice is controlled by the user. Synonyms: strip, swath.

window similar to the MBB, except it may be of only a part of the chip. MACE uses only rectangular windows whose size in the x dimension is equal to that of the chip.

Appendix II

Test Results

Hardware characteristics:

Node1: VAX-11/780	24 Mb physical memory
Node2: VAX-11/780	24 Mb physical memory
Node3: VAX-11/780	24 Mb physical memory
Node4: VAX-11/785	32 Mb physical memory
Node5: VAX 8600	32 Mb physical memory
Node6: MicroVAX II	5 Mb physical memory

Test size (approximate)
Block1: 1600 nodes, 600 devices
Block2: 23000 nodes, 6000 devices

Test 1: Comparison of serial IV and sequential MACE on Node5, block1.

IV:	CPU time: 01:03	elapsed time: 01:16
MACE:		
split	CPU time: 00:21	elapsed time: 00:30
extract 1	CPU time: 00:27	elapsed time: 00:37
extract 2	CPU time: 00:25	elapsed time: 00:34
merge 1,2	CPU time: 00:19	elapsed time: 00:58
Total	CPU time: 01:32	elapsed time: 02:38

Test 2: Comparison of serial IV with sequential MACE on Node5, block 2.

IV:	CPU time: 17:54	elapsed time: 18:23
MACE:		
split	CPU time: 01:05	elapsed time: 01:51
extract 1	CPU time: 15:10	elapsed time: 17:58
extract 2	CPU time: 06:40	elapsed time: 07:30
merge 1,2	CPU time: 02:23	elapsed time: 05:06
Total	CPU time: 24:20	elapsed time: 32:25

Test 3: Three-way comparison.
Notes: IV ran on Node3; MACE ran on Node1; MACE+EPIC ran on Node1, Node2,

and Node3, with Node4 as master. Test case: Block1.

IV:	CPU time: 04:04	elapsed time: 04:30
MACE:		
split	CPU time: 01:22	elapsed time: 01:34
extract 1	CPU time: 01:35	elapsed time: 01:46
extract 2	CPU time: 01:33	elapsed time: 01:47
merge 1,2	CPU time: 01:09	elapsed time: 02:16
Total	CPU time: 05:39	elapsed time: 07:24
MACE+EPIC:	CPU time: 05:36	elapsed time: 05:15

Test 4: Three-way comparison.
Notes: IV ran on Node3; MACE ran on Node1; MACE+EPIC ran on Node1, Node2, and Node3 with Node4 as master. Test case: Block2.

IV:	CPU time: 1:05:41	elapsed time: 1:09:31
MACE:		
split	CPU time: 03:52	elapsed time: 04:52
extract 1	CPU time: 50:47	elapsed time: 1:07:53
extract 2	CPU time: 24:01	elapsed time: 27:31
merge 1,2	CPU time: 08:52	elapsed time: 12:43
Total	CPU time: 1:27:31	elapsed time: 1:52:08
MACE+EPIC:	CPU time: 1:25:32	elapsed time: 1:17:00

Appendix III

Test Cases

III.1 Edges

III.1.1 Side edges

Correct edge handling requires correctly identifying the coordinates and the name of the node to which the edge belongs. This latter issue is applicable to any edge that is in an output file. Particular cases include

- edge local to slice
- edge with one point on cutline, other inside
- edge with one point on cutline, other on extreme edge
- two edges with the same coordinates but on different (distinct) layers
- edge touching cutline with no match on other side
- edge touching cutline with match on other side
- edge running the whole length of the side of the chip

III.1.2 Cutline edges

Edges on the cutline are essential to establish connectivity. In MACE, an overlap of any non-zero length was sufficient to establish connectivity. A good cutline sewing algorithm should handle these cases:

- two edges, different layers, same coordinates

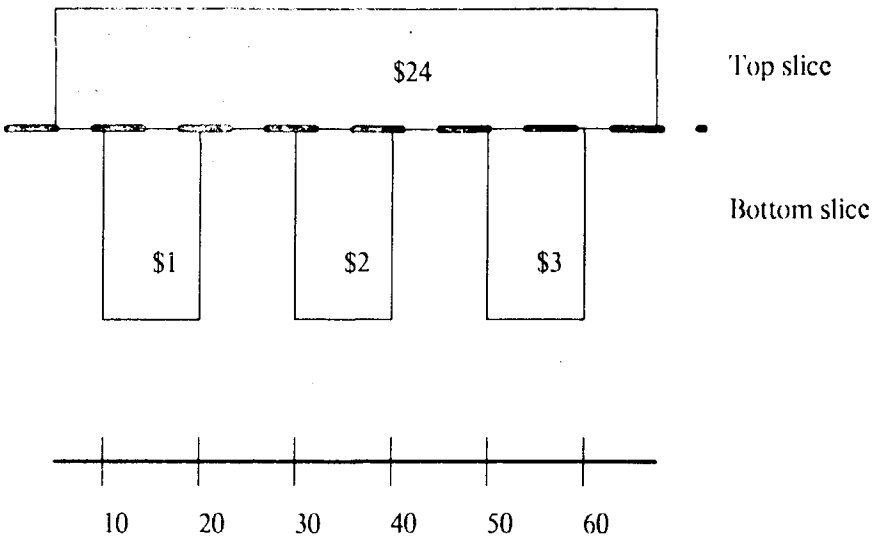


Figure III-1: Many to one edge/node match

- two edges, same layer, small negative overlap (no connection)
- two edges, same layer, zero overlap
- two edges, same layer, non-zero overlap
- two edges, one edge overhangs the other (begins before and ends after the other)
- two edges, the one that starts to the left ends to the left
- one edge on top that matches two or more seemingly distinct edges on the same layer below. See figure III-1
- checkerboard edges from top and bottom, all having zero overlap

III.2 Nodes

After the sewing up of the cutline edges, the connectivity of the nodes should be able to be deduced. Some pitfalls in node handling:

- Labels from top overrule numbers from bottom
- Labels from bottom overrule numbers from top
- Numbers from both sides are consistently handled
- Conflicting labels get reported
- Node flags of the resultant node are the inclusive or of node flags of the two constituents
- Conflicting flags (if any) are reported as errors

III.3 Devices

- Device flags of the resultant node are the inclusive or of the device flags of the two constituents
- Conflicting flags (if any) are reported as errors
- Devices touching the boundary of the whole chip are reported
- Unsplit devices are either handled or avoided and rehandled
- Splits (2-2 and 3-3) are handled

III.4 Device Geometry

Device geometry is exactly or equivalently constructed, for a testing tool to judge after the extraction. One especially tricky case appears in figure III-2. From the perspective of the top slice only, the channel has no hole. From the perspective of the bottom slice only, the channel has no hole. Both seem solid. However, when they are

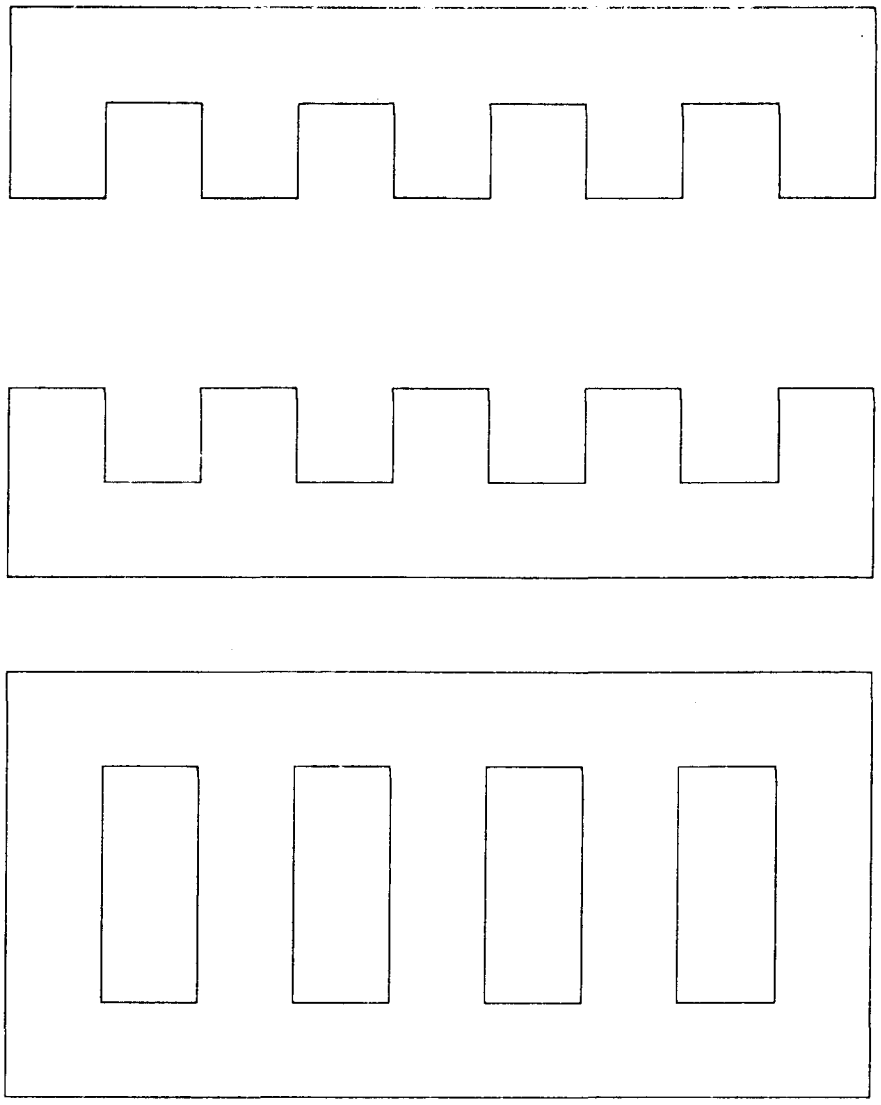


Figure III-2: The appearing hole channel

positioned adjacent, and the teeth come together, an arbitrary number of spaces are formed. By suitable positioning of the teeth (in a checkerboard pattern, for instance), even the number of holes, though constant, is a matter of semantics. If a hole connects diagonally to another hole, is that two holes or one? Recall figure 6-1 on page 51. There was ambiguity of how to represent a hole. If the teeth mesh exactly and leave some holes in the middle, is the polygon represented as one slab with some holes in the middle, or one slab with one complex hole in the middle?

There are other, more mundane cases. Just as a node chain can wind from top to bottom to top again, and all nodes must be recognized, so can a device's channel wind over the cutline. Cases (d) and (e) of figure 4-1 on page 39 should be handled correctly, in addition to the simple, four-point channel merges that MACE handled correctly.