

# Compiling for Computational Origami Using Probabilistic Optimization

by

Robert Scott French

Submitted to the Department of  
Electrical Engineering and Computer Science  
in Partial Fulfillment of the  
Requirements for the  
Degree of  
Bachelor of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1990

© 1990 Robert Scott French

The author hereby grants to MIT permission to reproduce and to distribute copies  
of this thesis document in whole or in part.

Signature of Author: \_\_\_\_\_

Department of Electrical Engineering and Computer Science  
May 21, 1990

Certified by: \_\_\_\_\_

Thomas F. Knight, Jr.  
Professor, Department of Electrical Engineering and Computer Science  
Thesis Supervisor

Accepted by: \_\_\_\_\_

Leonard A. Gould  
Chairman, Department Committee on Undergraduate Theses

# Compiling for Computational Origami Using Probabilistic Optimization

by

Robert Scott French

Submitted to the Department of Electrical Engineering and Computer Science  
on May 21, 1990 in partial fulfillment of the  
requirements for the Degree of  
Bachelor of Science in Computer Science and Engineering

## ABSTRACT

The term computational origami refers to a class of computational architectures that consist of systolic arrays of processors arranged in a tesselable, mosaic architecture. This interconnection strategy provides a computational origami architecture with certain properties, including the ability to “fold” the architecture to almost arbitrarily select space-time tradeoffs. This thesis presents a compiler that can be used to compile simple, non-recursive programs into a two-dimensional origami architecture called the alternate-slant network. Probabilistic optimizations based on simulated annealing are proposed which result in a 50% savings on array size.

Thesis Advisor: Thomas F. Knight, Jr.  
Professor, Department of Electrical Engineering and Computer Science

## **Acknowledgments**

Many people have helped me with my academic career, and I would like to take this opportunity to thank them. I would first like to thank my parents, whose support of my computer interests throughout my life has enabled me to get to where I am today. I would also like to thank Isaac Chuang for introducing me to computational origami and for participating in many hours of discussion about how to compile for an origami computer, the members of the Student Information Processing Board, who provided me with a social atmosphere in which I could flame about random ideas, and my thesis advisor, Tom Knight, who gave me free reign on my thesis topic. Finally, special thanks go to Carol Smith for proofreading drafts of this thesis.

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                        | <b>1</b>  |
| <b>2</b> | <b>Overview of Origami</b>                 | <b>3</b>  |
| 2.1      | Introduction . . . . .                     | 3         |
| 2.2      | A Sample Array . . . . .                   | 4         |
| 2.3      | Folding . . . . .                          | 6         |
| 2.3.1    | Depthwise Folding . . . . .                | 6         |
| 2.3.2    | Widthwise Folding . . . . .                | 8         |
| 2.3.3    | The Foldability Criteria . . . . .         | 11        |
| 2.3.4    | The Use of Delay Lines . . . . .           | 13        |
| 2.3.5    | Boundary Conditions . . . . .              | 15        |
| 2.4      | Higher-Dimensional Origami . . . . .       | 16        |
| 2.5      | Why Use Computational Origami? . . . . .   | 16        |
| <b>3</b> | <b>Compiling for Computational Origami</b> | <b>18</b> |
| 3.1      | Introduction . . . . .                     | 18        |
| 3.2      | The Architecture . . . . .                 | 19        |
| 3.3      | The Routing Algorithm . . . . .            | 19        |
| 3.3.1    | Preliminaries . . . . .                    | 20        |
| 3.3.2    | Module Placement . . . . .                 | 22        |
| 3.3.3    | Routing . . . . .                          | 24        |
| 3.4      | An Example . . . . .                       | 26        |
| <b>4</b> | <b>Optimization</b>                        | <b>32</b> |
| 4.1      | Simulated Annealing . . . . .              | 32        |
| 4.1.1    | Mutations . . . . .                        | 34        |
| 4.1.2    | The Cost Function . . . . .                | 35        |
| 4.2      | Empirical Results . . . . .                | 36        |
| <b>5</b> | <b>Future Work</b>                         | <b>46</b> |
| <b>6</b> | <b>Conclusion</b>                          | <b>48</b> |

|          |  |           |
|----------|--|-----------|
| <b>A</b> | <b>The Compiler</b>                      | <b>52</b> |
| A.1      | The Input Language . . . . .             | 52        |
| A.2      | Running the Compiler . . . . .           | 57        |
| A.3      | The Architecture Specification . . . . . | 61        |
| A.4      | The Library Specification . . . . .      | 63        |
| A.5      | Example . . . . .                        | 66        |
| A.6      | Error Messages . . . . .                 | 68        |
| <b>B</b> | <b>Compiler Source Code</b>              | <b>74</b> |

# List of Figures

|      |   |    |
|------|---|----|
| 2.1  | A sample $4 \times 4$ origami array. . . . .  | 5  |
| 2.2  | An origami array operating at full throughput and half throughput. . . . .  | 7  |
| 2.3  | An origami array after four and eight clock cycles running at half the maximum throughput rate. . . . .                                 | 7  |
| 2.4  | An origami array with half the number of processors, folded vertically. . . . .   | 8  |
| 2.5  | An origami array undergoing widthwise folding. . . . .  | 10 |
| 2.6  | A sample $4 \times 4$ alternate-slant architecture. . . . .   | 12 |
| 2.7  | The $4 \times 4$ alternate-slant architecture with the interconnection “irregularities” abstracted away. . . . .                        | 12 |
| 2.8  | A single-processor emulation of a $4 \times 4$ origami array using delay lines. . . . .   | 14 |
| 2.9  | The raster scan numbers for a $4 \times 4$ origami array. . . . .   | 14 |
| 2.10 | The interconnection of a $4 \times 4$ origami array including boundary conditions. . . . .  | 15 |
| 3.1  | The alternate-slant architecture. . . . .   | 19 |
| 3.2  | The module positions after placement. . . . .   | 28 |
| 3.3  | The final array after routing. . . . .  | 28 |
| 3.4  | The hand-optimized array after routing. . . . .   | 30 |
| 4.1  | The original four-bit ripple carry adder array, and the array after 3,750 optimization iterations. . . . .                              | 39 |
| 4.2  | Iteration vs. cost for the four-bit ripple carry adder, starting temperature 1, temperature multiplier .99, 100 iterations. . . . .     | 40 |
| 4.3  | Iteration vs. cost for the four-bit ripple carry adder, starting temperature 10, temperature multiplier .99, 500 iterations. . . . .    | 41 |
| 4.4  | Iteration vs. cost for the four-bit ripple carry adder, starting temperature 25, temperature multiplier .99, 750 iterations. . . . .    | 42 |
| 4.5  | Iteration vs. cost for the four-bit ripple carry adder, starting temperature 40, temperature multiplier .99, 1,000 iterations. . . . .  | 43 |
| 4.6  | Iteration vs. cost for the four-bit ripple carry adder, starting temperature 55, temperature multiplier .996, 1,250 iterations. . . . . | 44 |
| 4.7  | Iteration vs. cost for the four-bit ripple carry adder, starting temperature 70, temperature multiplier .999, 3,750 iterations. . . . . | 45 |

|     |  |    |
|-----|--|----|
| A.1 | The ANDOR library routine. . . . .             | 65 |
| A.2 | The four-bit ripple carry adder array. . . . . | 69 |

# List of Tables

|     |   |    |
|-----|---|----|
| 3.1 | Flavors used to resolve various routing conflicts. . . . .              | 26 |
| 4.1 | Simulated annealing trials for the four-bit ripple carry adder. . . . . | 37 |



# Chapter 1

## Introduction

The development of parallel-processing computer technology has proceeded at rapid pace over the past decade. Computers have progressed from systems with a single high-functionality processor to ones with hundreds of thousands or millions of low-functionality processors. These latter systems rely more on the parallelism of a problem than raw processor power to find a time-efficient solution. When millions of processors need to be connected, the interconnection strategy becomes a serious problem; in order to decrease the routing distance between processors, high-dimensional networks must be used, which in turn lead to problems in physical layout. A different approach is to arrange the processors in a regular-interconnect mesh. While this does not provide for easy communication between distant processors, it is easy to wire. It is also well-suited for some classes of applications, including those that require simple data flow, such as FFTs. One class of such architectures, developed by Alan Huang of AT&T Bell Laboratories, is called *computational origami*. While this class of ar-

chitectures can include any grain of processors from 2-input/2-output boolean gates to Cray-2s, a simple, fine-grained version is ideal for some currently planned optical computers [14, 3, 13, 5].

This thesis does not dwell on the properties of an origami architecture, although a brief overview is given in Chapter 2. More thorough treatments can be found in [15, 19, 3, 6, 4, 16]. Instead, it deals with the software problem of assigning tasks to the processors in such a machine. Since this is such a large problem, only one particular case is examined, that of a machine consisting of simple 2-input/2-output processors connected by a 2-dimensional alternate-slant interconnect. A program for compiling a simple language into task assignments is described.

Chapter 2 provides an overview of the concepts of computational origami. Chapter 3 describes the destination architecture for the compiler, and introduces the algorithms used by the compiler. Chapter 4 describes several probabilistic optimization methods for the compiler, and presents results on trial programs. Chapter 5 discusses future work that can be performed. Appendix A describes the compiler itself in detail, and Appendix B contains a complete code listing of the compiler.

# Chapter 2

## Overview of Origami

### 2.1 Introduction

The term “computational origami” was coined by Alan Huang of AT&T Bell Laboratories [14, 15, 16] to refer to a class of parallel computing architectures that have four main properties:

- Each processing element has identical functionality.
- Each processing element is stateless.
- The processing elements are connected by a regular, tessellable interconnect.
- The interconnect has causal data flow.

On the surface, an origami array looks identical to a systolic array, which also consists of a regularly connected array of processing elements (PEs). However, the added constraints that the architecture be tessellable and have causal data flow give

it certain properties that will be discussed below. There are no constraints on the size of the network, processor functionality, or level of connectivity, as long as the three constraints above are satisfied. There are many viable origami networks, but we will be dealing solely with simple rectangular arrays because they are the easiest to analyze. The width of each array,  $W$ , is the number of PEs arranged horizontally in the array, and the height,  $H$ , is the number of PEs arranged vertically.

More detailed information about origami arrays and their application can be found in [14, 15, 19, 3, 6, 4, 10, 13, 16, 5].

## 2.2 A Sample Array

A sample origami array is shown in Figure 2.1. This example consists of a  $4 \times 4$  array of PEs, each with two inputs and two outputs. Each PE is connected to the PE directly below, and to the PE directly below and to the right. Neither the computations that can be performed by the PEs (called *flavors*), nor the size of the inputs and outputs have been specified. Each PE could be a simple logic gate or a very complex logic function, and each input or output could be one bit wide, eight bits wide, or hundreds of bits wide. In fact, each PE could be a supercomputer with the connection channels containing gigabits worth of information. Within certain limitations, the properties discussed below are valid for any size PE and I/O channel.

In the example in Figure 2.1, data flows “down” the array. All inputs enter at the top, and all outputs exit at the bottom. We will ignore the boundary conditions

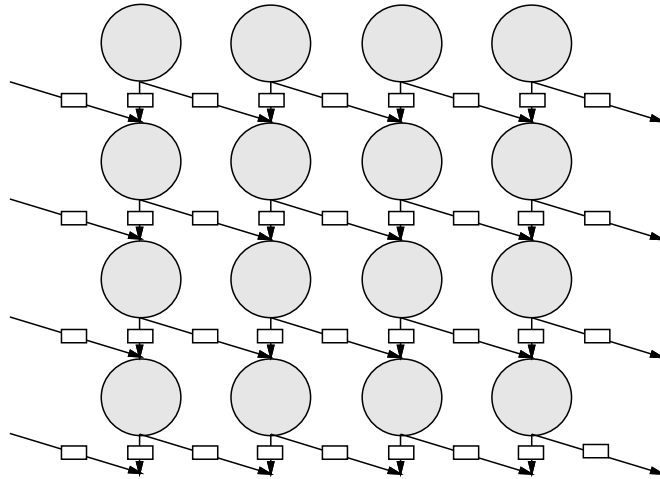


Figure 2.1: A sample  $4 \times 4$  origami array.

at the left and right sides for the moment. The processor array is *perfectly pipelined* and *synchronous*. That is, there is a delay between each level in the array in the direction of data flow, and each PE is allocated exactly one “clock cycle” to perform its computation. When a set of input data arrives at the top of the array, it takes  $H \cdot \Delta T$  seconds for the result to arrive at the outputs, where  $\Delta T$  is the time taken by a single pipelined stage. While the latency is dependent on the height of the array, the throughput is constant at  $\frac{1}{\Delta T}$ .

## 2.3 Folding

The observations in the previous section bring us to the concept of *folding*.<sup>12</sup> Obviously, if we have as many physical PEs as we need to execute a given array, we have a latency of  $H \cdot \Delta T$  and a throughput of  $\frac{1}{\Delta T}$ . However, hardware may be expensive and throughput not as critical. In this case, a reduction in hardware can be made without a loss of functionality.

### 2.3.1 Depthwise Folding

If we feed the inputs at half the full rate, as shown in Figure 2.2<sup>3</sup> and allow the data to propagate, several observations can be made. After four clock cycles, the bottom half of the array is unused. After eight clock cycles, the top half of the array is unused. These cases are shown in Figure 2.3. Thus, since only half the hardware is in use at any given time, and all the PEs are identical, it should be possible to implement the system using only half the number of processors.

One way to do this is illustrated in Figure 2.4. Half the number of processors are available, and the outputs of each processor are fed back to the inputs of the array through multiplexers. If the input multiplexers are set on “data input” for four clock cycles, and then switched to “feedback input” for four clock cycles, the array will

---

<sup>1</sup>Much of this section is taken from [19].

<sup>2</sup>The term *origami* refers to the Japanese art of paper folding. Along the same lines, *computational origami* refers to an architecture that can be “folded.” The word *origami* is composed of the roots *oru-*, meaning to fold, and *kami*, meaning paper. As such, the actual processor array will sometimes be referred to as a “kami map.”

<sup>3</sup>In this and subsequent figures PEs will be indicated by shaded squares and pipeline delays will not be shown. It is assumed that each PE has a built-in delay on each of its outputs.

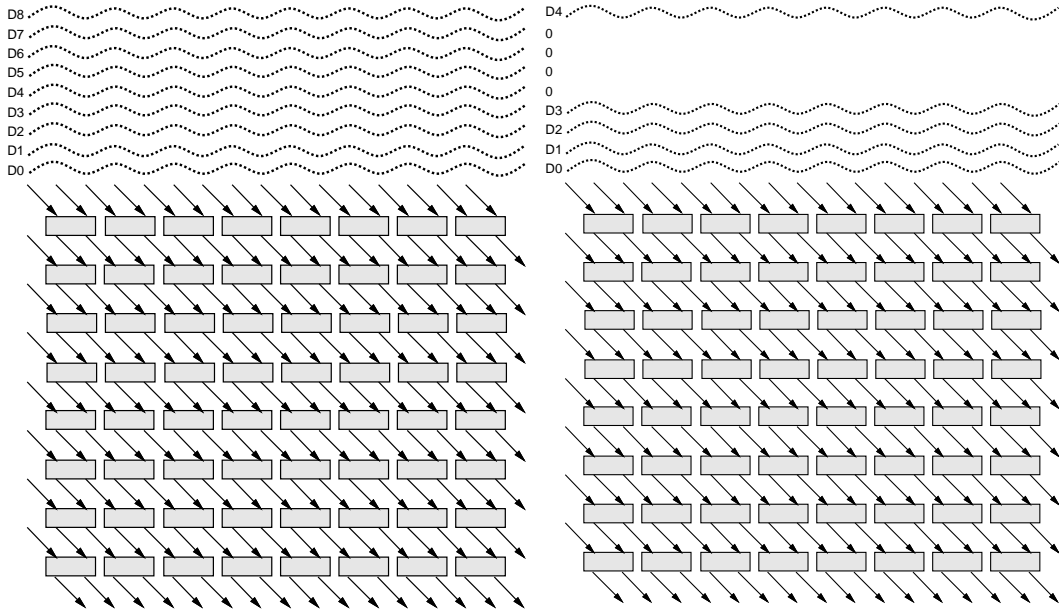


Figure 2.2: An origami array operating at full throughput and half throughput.

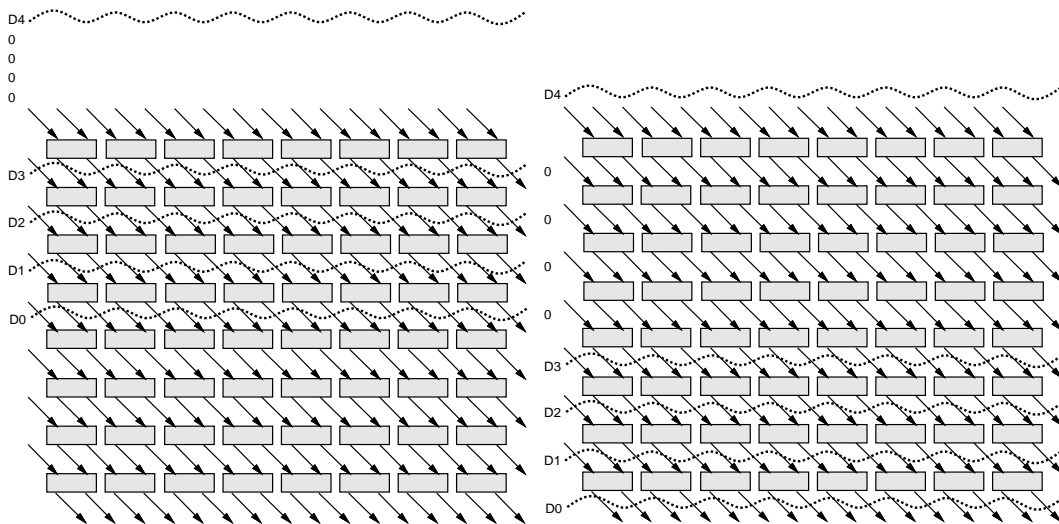


Figure 2.3: An origami array after four and eight clock cycles running at half the maximum throughput rate.

perform exactly as before, but with half the throughput rate and half the number of PEs. The PEs must switch tasks as appropriate to guarantee that the data is processed properly. The latency will remain constant.

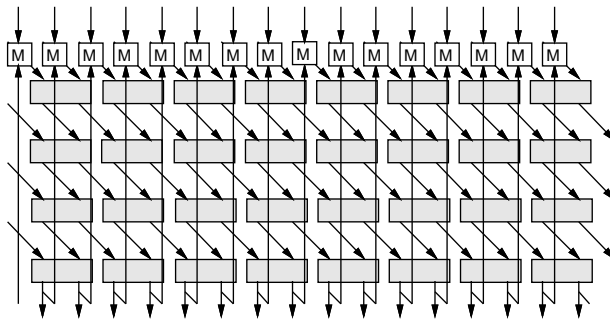


Figure 2.4: An origami array with half the number of processors, folded vertically.

This kind of vertical “folding,” called *depthwise folding*, can be performed to make the processor array arbitrarily small vertically. The properties of depthwise folding are:

- The throughput decreases from  $\frac{1}{\Delta T}$  to  $\frac{1}{F \cdot \Delta T}$ , where  $F$  is the folding factor (ratio of full array height to folded array height).
- The latency remains the same.
- No extra memory is required to store intermediate results.

### 2.3.2 Widthwise Folding

I have shown how to reduce the hardware required from  $H \times W$  to  $W$  PEs, but we can reduce it still further. Suppose this time that we feed the left half of the inputs in on the first eight clock cycles, and the right half of the inputs on the second eight



clock cycles, as shown in Figure 2.5. Except for a data dependency between the nodes on the line dividing the array in half, the two halves operate independently, and are not in use at the same time. We can use short term memory to store the results that cross the dividing line, and then the array can be executed with half the number of processors as was done earlier. Again the PEs must switch tasks as necessary. This kind of folding is called *widthwise folding*.

The properties of widthwise folding are:

- The throughput decreases from  $\frac{1}{\Delta T}$  to  $\frac{1}{F \cdot \Delta T}$ , where  $F$  is the folding factor (ratio of full array width to folded array width).
- The latency remains the same.
- The extra memory required to store intermediate results is proportional to  $H \cdot F \cdot P_c$ , where  $P_c$  is the number of outputs from each processor that cross the fold line.

Depthwise folding and widthwise folding can be combined arbitrarily to execute the desired array with as little as a single PE. This brings us to the first major property of an origami array:

*The amount of time required to execute an origami array is inversely proportional to the number of processors available, assuming the width and height of the desired array are perfect multiples of the width and height of the available processor array.*

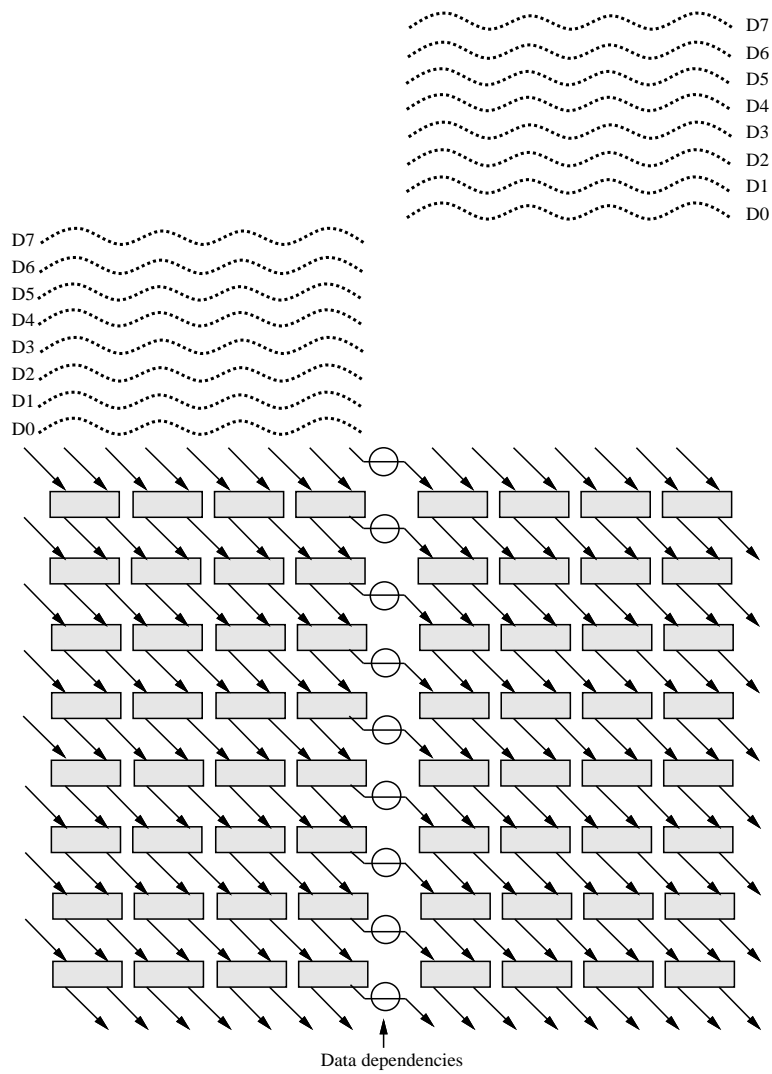


Figure 2.5: An origami array undergoing widthwise folding.

### 2.3.3 The Foldability Criteria

Not all regular arrays are foldable. We have already presented the four basic requirements for an array to be considered an origami array:

- Each processing element has identical functionality.
- Each processing element is stateless.
- The processing elements are connected by a regular, tessellable interconnect.
- The interconnect has causal data flow.

There are obviously some arrays that would be desirable to build that do not fit these criteria. Lu [19] showed that any semi-regular array can be turned into an origami array as long as the irregularities can be “abstracted” away. Take, for example, the alternate-slant architecture shown in Figure 2.6. This array is not foldable because the data dependencies vary on odd and even levels. However, such an array could be made foldable by taking each group of  $1 \times 2$  processors and turning them into a new, virtual processor, as shown in Figure 2.7. The new processor has three inputs and three outputs. The architecture is now foldable.

The smallest collection of processors that can be abstracted away to make a foldable architecture is called the *kernel*. What architectures can be made foldable? Lu [19] rephrases the “foldability criteria:”

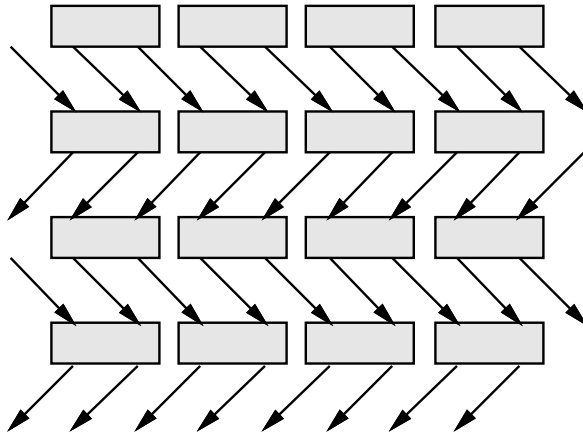


Figure 2.6: A sample  $4 \times 4$  alternate-slant architecture.

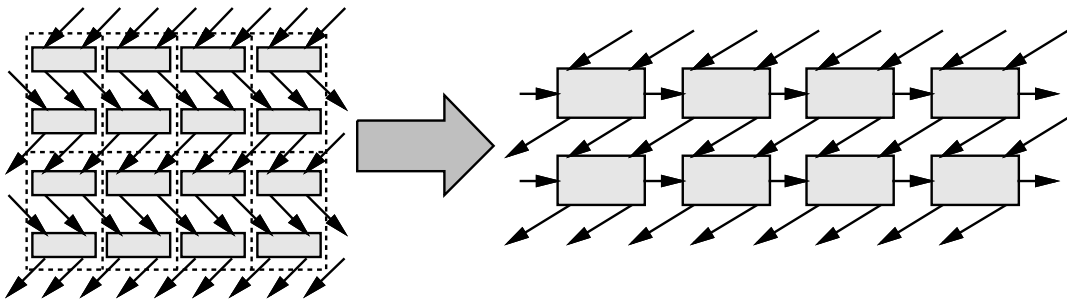


Figure 2.7: The  $4 \times 4$  alternate-slant architecture with the interconnection “irregularities” abstracted away.

*An architecture is foldable if and only if its local architecture has an abstraction which is a standard local mosaic architecture, provided that the boundaries are compatible.*

To this must also be added that the system as a whole have causal data flow.

### **2.3.4 The Use of Delay Lines**

So far the short-term memories that have been used to store the intermediate results of a folded array have been left undescribed. They could be implemented using standard RAM with the address determined by the particular phase of data flow, but a much more useful arrangement is possible. The short-term memory can actually be viewed as a set of delay lines, with one delay line for each of the  $P_o$  outputs of a processor. An example of this is shown in Figure 2.8. This figure shows a single-processor emulation of the  $4 \times 4$  array we introduced in Figure 2.1. For ease of discussion, the processor in this figure does not have an intrinsic delay.

The length of a given delay line can be easily determined. If the PEs in an origami array are numbered in a raster-scan fashion, as shown in Figure 2.9, the length of a delay line between the output of one PE and the input of another is simply the difference in the numbers of the two PEs. Our sample array requires two delay lines, one of length 4 and one of length 5. The length of the delay lines is approximately proportional to the width of the array, which is consistent with our earlier observation that only widthwise folding requires extra memory.

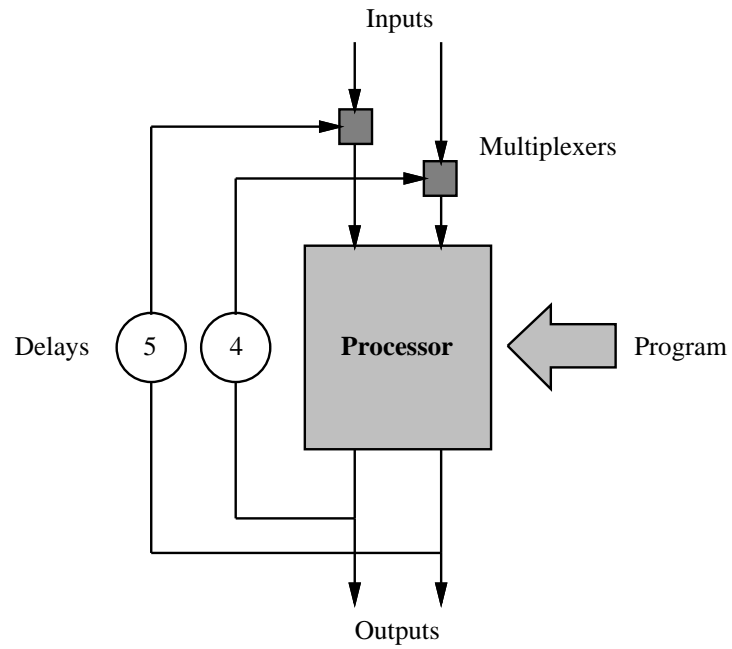


Figure 2.8: A single-processor emulation of a  $4 \times 4$  origami array using delay lines.

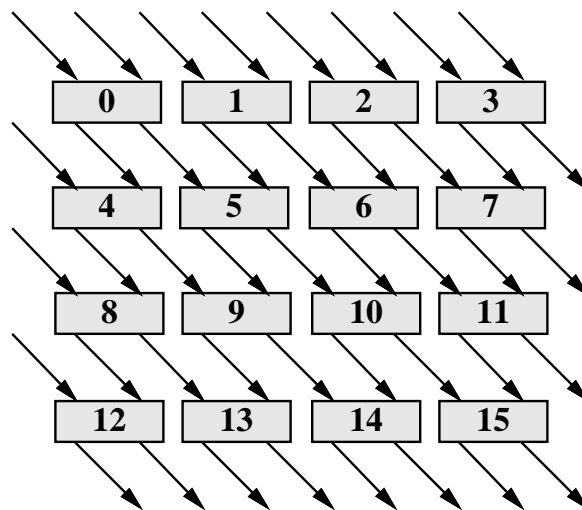


Figure 2.9: The raster scan numbers for a  $4 \times 4$  origami array.

### 2.3.5 Boundary Conditions

Once we see that delay line lengths can be calculated using raster-scan numbers, we can see how the array behaves at its boundaries. Using Figure 2.9 as a guide, we see that PE 0's outputs are fed to PEs 4 and 5. Looking at the array further, we see that PE 3 must talk to PE 7 and 8. Thus the behavior of the left and right edges can now be understood, and is drawn in Figure 2.10. When data flows off the right hand side of the array, it essentially "skips ahead" one clock cycle. This provides interesting possibilities for time-domain multiplexing, and is discussed briefly in Chapter 5. However, for the most part these boundary conditions are not in the scope of this thesis.

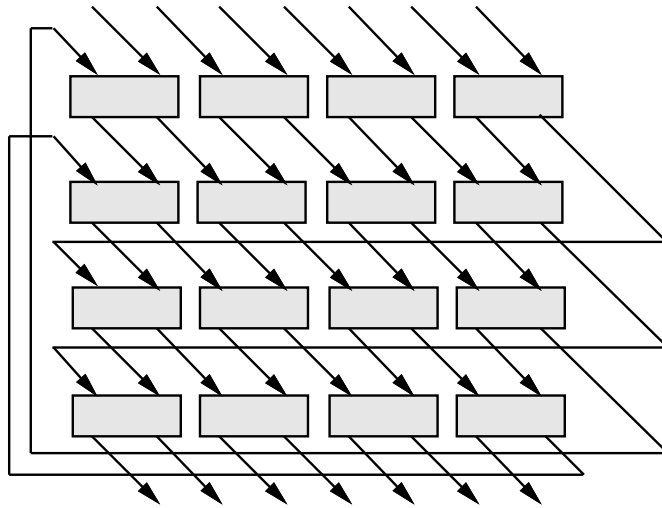


Figure 2.10: The interconnection of a 4x4 origami array including boundary conditions.

## 2.4 Higher-Dimensional Origami

All the examples that have been presented so far, and in fact all the examples that will be seen later, contain two-dimensional origami arrays. These are explored here simply because they are the easiest to analyze. However, all the points made in this chapter also apply to origami arrays with more than two dimensions. In these cases, there is one dimension that can fold depthwise, and the rest can fold widthwise.

## 2.5 Why Use Computational Origami?

It is evident from the above discussion that computational origami has a number of serious disadvantages. For example, if simple, fine-grained processors are used, either a lot of hardware or long delay lines are required to execute a complex program. It is impractical to implement an origami array in silicon because the long delay lines required are not area-efficient when implemented using standard techniques. In addition, even with the maximum number of useful processors available, the system will have a high latency. Finally, the systems that have been characterized here can only run programs that can be implemented without indeterminate recursion, since the data flow must be causal.

However, origami has a number of advantages. Origami systems are easy to lay out and wire. There is a nearly perfect processor/speed tradeoff, as opposed to most parallel processing architectures which get bogged down in communication costs as the number of processors increase. There is excellent throughput when enough processors



are available. Programs are independent of the number of processors available as long as sufficiently long delay lines can be constructed, and need not be recompiled as the number of processors changes. And complex problems can be performed with as little as a single processor if necessary.

# Chapter 3

## Compiling for Computational Origami

### 3.1 Introduction

One of the primary problems with an origami system is assigning functions to each of the PEs so that a task can be performed efficiently with a minimal amount of hardware. Standard compiler techniques can be used to generate data flow graphs from computer languages, but creating an origami array from the logic functions produced is a difficult problem. In many ways it is analogous to the generalized two-dimensional electrical circuit layout problem (*e.g.* the automatic routing of wires on a printed circuit board), but it is sufficiently different that circuit layout algorithms are not directly applicable. The problem of finding an optimal placement and routing is considered to be NP-complete.

## 3.2 The Architecture

The destination architecture for the compiler presented in this thesis will be the alternate-slant network mentioned briefly in Chapter 2. It is shown again in Figure 3.1. This architecture was chosen primarily for its simplicity. The simpler, unidirectional-slant architecture originally introduced is not used because it is more difficult to route properly, although Chuang has shown that it is possible to convert the unidirectional-slant architecture into the alternate-slant architecture assuming boundary conditions are ignored [5].

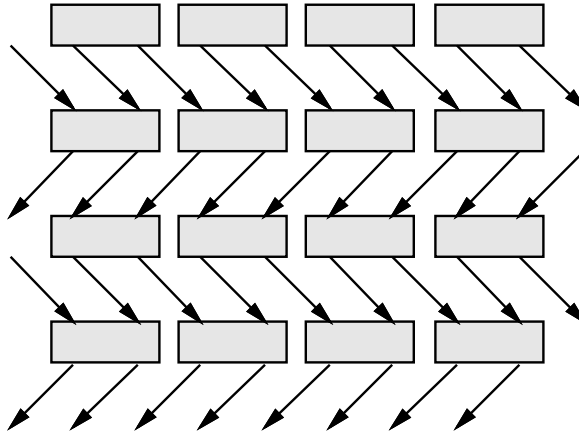


Figure 3.1: The alternate-slant architecture.

## 3.3 The Routing Algorithm

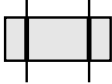
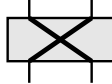
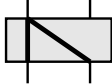
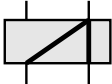
Chuang [3] developed a prototype routing system for a three-dimensional origami array using a flooding algorithm with backtracking. This was used to place and route

a Wallace tree adder. Here we propose an algorithm for placing logic functions and routing between them in two dimensions without the need for backtracking.

### 3.3.1 Preliminaries

For the purposes of this algorithm, an origami array is a two-dimensional staggered array of nodes as described above. In a fully automated logical compiler it might be desirable to treat each node independently for placement. However, in a large application the number of nodes may be very large (equal to the number of discrete logical functions that need to be performed) and compilation time quickly increases beyond the realm of practicality. Therefore it is frequently desirable to break down the problem into subproblems, and place and route the nodes required for each subproblem separately or perhaps even by hand. Once the placement and routing for a subproblem has been produced, we can consider the result to be an atomic unit for future connection to other nodes and subproblems. Such a subproblem is called a *module*, and is considered in this algorithm to be an  $x$  by  $y$  rectangular set of nodes with defined input and output locations. Typical modules might be adders, multipliers, and bus multiplexors. Individual ungrouped nodes are also considered to be modules.

This algorithm assumes that there are four flavors dedicated to routing. They are:

1. *passthrough* : copy the left input to the left output, and the right input to the right output.
2. *crossover* : copy the left input to the right output, and the right input to the left output.
3. *left broadcast* : copy the left input to both the left and right outputs, and discard the right input.
4. *right broadcast* : copy the right input to both the left and right outputs, and discard the left input.

A series of connected routing flavors is called a *wire*.

The algorithm requires the following data as input:

- $\mathcal{M}$ , the set of all the modules that need to be placed.
- $\mathcal{D}$ , the set of dependencies between the modules. A dependency is the set of modules that provide inputs to a particular module.
- $\mathcal{R}$ , the set of routings between the modules. Each routing  $r \in \mathcal{R}$  indicates an output pin of a module ( $r.src$ ) and the input pin of another module ( $r.dest$ ) to which it should be connected.

Program inputs and outputs are treated as special modules that are used during routing, but are not actually placed in the physical array.

### 3.3.2 Module Placement

The first step in the algorithm is to place the modules in an array so that routing between them is as short as reasonably possible. Modules are first ordered vertically into distinct *levels* (this is a *logical* placement—the physical placement won't be determined until later), and then are placed horizontally within each level.

The ordering of modules vertically is a simple process:

1. Mark all the modules as unused.
2. Add all the input modules to level 0, and mark them as used; set the level number to 1.
3. Add to the current level all modules (which are not output modules) that depend only on modules that have already been added to previous levels.
4. Increment the level number by 1.
5. If all modules that are not output modules are marked as used, add the outputs to the current level and stop.
6. Go to step 3.

Note that step 3 must eventually use all the modules because there can be no circular dependencies between modules (the modules form a strict hierarchy).

Once the modules have been ordered vertically, they must be arranged horizontally. This is done in two stages: ideal placement, and shifting to allow room for

routing. During the ideal placement stage, the modules are placed in such a way that the total idealized routing distance to each module is minimized. For a given module,  $m$ , the idealized routing distance is the sum of the squares of the horizontal displacements for each module that feeds data to an input of  $m$ . The horizontal position of these modules will always be known since module placement proceeds in order down the hierarchy. For each level, modules are picked one by one and placed in such a manner that their idealized routing distance is minimized and they do not overlap.

Once the modules are ordered vertically and placed horizontally in an ideal manner, some may need to be shifted to make room for wires that need to go between modules. This is done by tracing the ideal path of each wire while keeping counters (which we will call  $m.right$  and  $m.left$ ) indicating how much space adjacent modules need between them. The algorithm is:

1. For all  $m \in \mathcal{M}$ , let  $m.right = 0, m.left = 0$ .
2. For each  $r \in \mathcal{R}$ , follow the routing from its source to its destination in a straight diagonal line. Whenever  $r$  would intersect a module,  $m$ , increment either  $m.right$  or  $m.left$  depending on whether the wire is intersecting the right half or left half of the module, respectively. Keep track of which modules  $r$  needs to pass between, and whether it needs to pass on the right or the left.
3. Shift the modules (keeping their same relative horizontal position) such that the distance between adjacent modules  $m_1$  and  $m_2$  is at least  $m_1.right + m_2.left$ .

4. For each  $r \in \mathcal{R}$ , find the new position of each pair of modules it is going to pass between, and assign it a goal column for that level such that no two wires have the same goal column for that level (there must be enough columns because of steps 2 and 3). The goal column for the wire's starting level is the horizontal position of the appropriate output of the source module, and the goal column for the destination level is the horizontal position of the appropriate input of the destination module. The only time two wires can have the same goal column is at the wires' starting level. This will happen when the output of a module needs to be sent to two other modules.

Once this step is completed, all modules have been placed in such a manner that routing, using the appropriate goal columns, *must* be possible without backtracking.

### 3.3.3 Routing

Now that the modules have been placed horizontally and ordered vertically, and each wire that needs to be routed has a goal column for every level it must pass through, routing can proceed. Routing proceeds from the outputs, up through the levels, to the inputs (thus routing proceeds in the *opposite* direction of the flow of data). The routing algorithm maintains the following state: the current level and the set of wires,  $\mathcal{W}$ , that are currently being routed. It proceeds as follows:



1. Set  $\mathcal{W}$  equal to all  $r \in \mathcal{R}$  whose destination module is an output, and set each wire's current position to the horizontal position of the output. Set the current level to the level on which all outputs reside minus 1.
2. Determine the goal column for each wire for the current level.
3. Route each wire toward its goal column by the method outlined below. When all the wires destined for a given module are at their respective goal columns, and placing that module here would not block other wires from reaching their goal columns on this level, place this module in the array, and delete all  $w \in \mathcal{W}$  whose source module has now been placed.
4. Continue routing as in step 3. Whenever the top of a freshly placed module is reached, add the wires whose destinations are that module to  $\mathcal{W}$ . Continue until the tops of all modules on this level have been reached.
5. Decrement the current level number.
6. Repeat until level 0 (the level containing only array inputs) is reached.
7. Continue routing wires until all wires have reached their goal columns (the positions of the appropriate inputs).

As wires are being routed, a number of conflicts can arise. These include two wires interacting (such as needing to cross) or a wire needing to move left or right and not being able to (because it is already at the left or right side of a processor, and

thus can't go sideways on the current level). Such conflicts are resolved according to the appropriate entry in Table 3.1. For example, if the wire entering on the left side of the node needs to go right and the wire entering on the right side of the node needs to go left, a crossover should be placed at the current location. Likewise if only one wire is entering the node, is entering on the left, and needs to go left, a passthrough should be placed since the wire will be able to move left on the next level up (because of the alternate-slant architecture).

The only conflict not covered by this table is the case where two wires are entering a node and have the same goal column for the current level. In this case, the wires should be combined by using a left or right broadcast and removing one of the wires from the current wire list  $\mathcal{W}$ .

|                              |          | Wire on left needs to go |             |             |             |
|------------------------------|----------|--------------------------|-------------|-------------|-------------|
|                              |          | left                     | straight    | right       | none        |
| Wire on right<br>needs to go | left     | passthrough              | crossover   | crossover   | crossover   |
|                              | straight | passthrough              | passthrough | crossover   | passthrough |
|                              | right    | passthrough              | passthrough | passthrough | passthrough |
|                              | none     | passthrough              | passthrough | crossover   |             |

Table 3.1: Flavors used to resolve various routing conflicts.

### 3.4 An Example

Now that we've gone over the algorithm in detail, an example may help to get a more intuitive feel for the process. The language used in this example is described in detail in section A.1. The program that we will examine is:

```
INPUT a<2>@0, b<2>@4, c<1>@3;  
OUTPUT o<1>@0, p<1>@4, q<1>@3;
```

```
o = AND(a<0>, b<0>);  
p = AND(a<1>, b<1>);  
q = c;
```

This program takes two two-bit inputs, **a** and **b**, ANDs them together pairwise, and generates two one-bit outputs in **o** and **p**. There is also a single input, **c**, that is copied unchanged to the output **q**. The inputs have been specifically arranged in this example to cause the signals to cross during routing. The vertical hierarchy for this program is:

1. Inputs
2. Both ANDs
3. Outputs

After ideal placement and moving to avoid routing conflicts, the position of the modules is as shown in Figure 3.2. The final array, after routing with this algorithm, is shown in Figure 3.3. The processors are staggered by half on each level, which is equivalent to the alternate-slant interconnect we've been discussing. The coordinates on the bottom correspond to those specified in an **INPUT** or **OUTPUT** statement.

A number of observations can be made about this array. The position of the inputs and outputs is obviously very critical. The last row of processors is devoted solely to moving the outputs to the desired locations, and the first three rows are

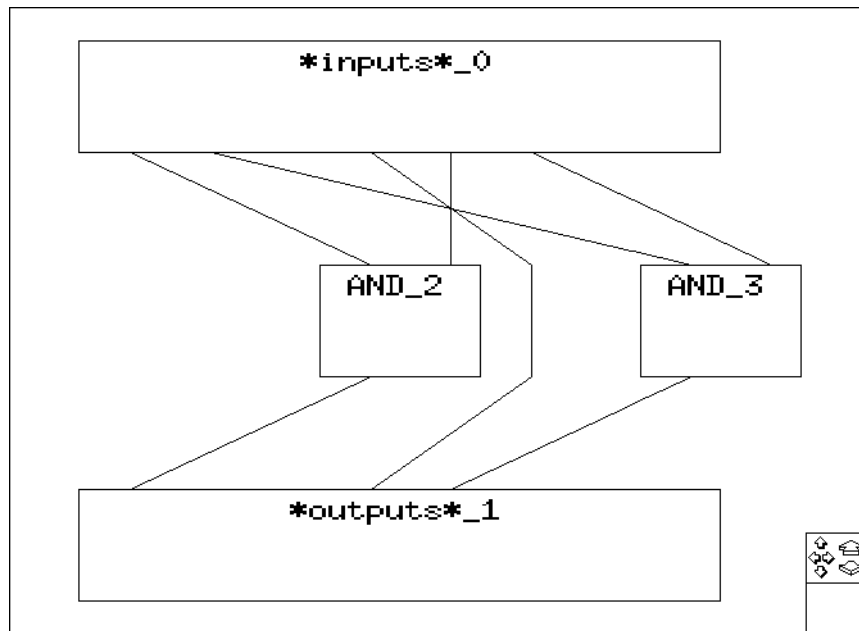


Figure 3.2: The module positions after placement.

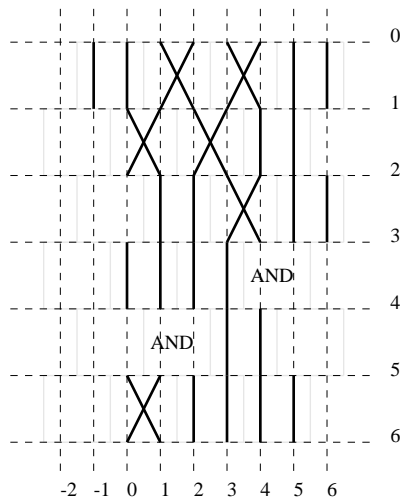


Figure 3.3: The final array after routing.

devoted to permuting the inputs to reach the AND gates properly. This is obviously a very bad use of processor hardware. Some hand-tweaking can produce a much better result. For example, seeing how this result came out, we can rearrange the inputs and outputs by hand:

```
INPUT a<2>@[1,3], b<2>@[2,4], c<1>@5;  
OUTPUT o<1>@1, p<1>@3, q<1>@5;  
  
o = AND(a<0>, b<0>);  
p = AND(a<1>, b<1>);  
q = c;
```

This produces the optimal array shown in Figure 3.4. From this array we also see that the height of the array must always be an even number of processors tall, so that the array will fold properly. It is definitely a problem that the user must rearrange the inputs and outputs in order to achieve a good array. As the arrays get larger, the inputs and outputs have less of a direct impact, though, and the exact positioning of the modules, over which the user has no control, becomes the overriding factor. The automatic optimization of module placement will be the subject of the next chapter. Optimization of inputs and outputs is not dealt with in this thesis, but is discussed briefly in Chapter 5.

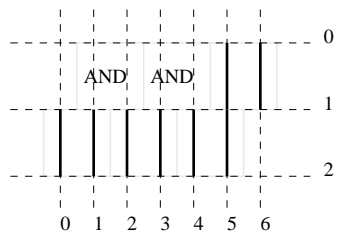


Figure 3.4: The hand-optimized array after routing.



# Chapter 4

## Optimization

The algorithm presented in Chapter 3 runs in polynomial time. Unfortunately, we pay for the speed of the algorithm with inefficiencies in the resulting origami array. In the applications that have been generated by this algorithm so far, routing accounts for approximately 45% of all assigned nodes, while another 45% of the nodes are left unused entirely. When we consider that in an ideal situation each node would have a physical piece of hardware associated with it, and that the latency of the system is proportional to the height of the array, we can see that this is a tremendous amount of wasted time and hardware. Luckily, there appear to be a number of general ways to improve this algorithm.

### 4.1 Simulated Annealing

Since the generation of an optimal placement and routing is probably NP-complete, any polynomial-time optimizations must be iterative in nature. One of the most



popular iterative optimization methods is *simulated annealing* [17, 22, 2, 9, 12, 1, 8, 20, 25]. This method is borrowed from materials science, where it is well known that to produce a solid in a low energy state (such as a perfect crystal), you first heat the system to a high temperature, and then slowly cool it. At any given temperature, the probability that a change in the structure with energy change  $\Delta E$  will occur is  $\min(e^{-\frac{\Delta E}{kT}}, 1)$ , where  $k$  is Boltzmann’s constant and  $T$  is the temperature of the system in degrees Kelvin. When the temperature is high, the system can change radically and many changes that do not lower the energy level are allowed. As the temperature decreases, fewer and fewer “bad” changes are permitted, until finally a fairly optimal state is achieved.

In the computational equivalent of this method, an initial system is chosen and then iteratively optimized. During each pass, one of a number of optimizations is chosen and applied to the current system. A cost function is used to indicate the desirability of a given system, and the change in cost ( $\Delta c$ ) from the current system to the new system is computed. The new system is accepted with probability:

$$P = \begin{cases} \Delta c \leq 0 & 1 \\ \Delta c > 0 & e^{-\Delta c/T} \end{cases}$$

where  $T$  is the “temperature” of the system.  $T$  is gradually decreased according to an “annealing schedule” as more optimizations are applied, thus causing the system to gradually accept fewer and fewer “bad” changes.

### **4.1.1 Mutations**

There are many “mutations” that could be chosen during each pass of the annealing. Three of the optimizations that could be iteratively applied using this or similar methods are described below.

#### **Local Module Swapping**

The modules have been placed to minimize the length of the horizontal component of the wires, but not the vertical component. Localized transposition of modules might decrease routing requirements in some cases. We will consider the case of transposing adjacent modules.

#### **Module Movement**

As described in the previous section, moving modules may improve the real routing distance. Modules can be moved one processor to the left or to the right, assuming the new position does not conflict with another module on the same level. This may also improve global routing in some cases.

#### **Input/Output Permutation of Modules**

Much of the routing in an origami array is devoted to permuting a set of wires to match the input requirements of a module. For example, if an adder has an output with the most significant bit on the left, and this result needs to be sent to a negation module that expects the most significant bit on the right, a great deal of time will be

spent rearranging the wires to satisfy this constraint. While it is possible to partially solve this problem by developing libraries of “matching” modules, it is impossible to do this for all combinations in practice. A simple solution to this problem is to provide more than one module capable of performing a particular task. The modules would be identical in function, but would have their input and output pins permuted in different ways so that the routing distance could be reduced by selection of the appropriate modules. Since it is impossible to determine which instance of a module will produce the largest reduction in global array size, modules must be chosen at random during the annealing process.

### **4.1.2 The Cost Function**

The cost function is an important part of the simulated annealing process, since it is responsible for recommending whether a particular configuration is better than another.

An obvious cost function is the size of the finished array,  $c = W \cdot H$ . Unfortunately, while this cost function is probably one of the best we could hope for, since it directly optimizes for minimum array size, it is one of the slowest to compute since the entire array must be routed.

A faster, but perhaps less accurate cost function is the sum of the total horizontal distance that wires need to be routed. To this could also be added the number of wires that cross when going between levels. Since permuting large collections of

wires is one of the most expensive operations in an origami array, this cost function might provide a good approximation to the final array size. In addition, it is easy to compute, and can be computed without routing the array at all.

We will deal primarily with the array-size cost function, since it is the most accurate, although the compiler supports all three cost functions.

## 4.2 Empirical Results

To illustrate simulated annealing, we will look at the optimization of a simple program, a four-bit ripple carry adder, as follows:

```
/* return two bit result of x+y+c */
ADDSTAGE(x<1>, y<1>, c<1>)
{
    DECL sum<1>, carry<1>, sum2<1>, carry2<1>, carryout<1>;

    sum, carry = ADD(x, y);
    sum2, carry2 = ADD(sum, c);
    carryout = OR(carry, carry2);

    RETURN sum2, carryout;
}

/* add 4bits+4bits -> 5 bits */
ADD4(x<4>, y<4>)
{
    DECL sum<5>, c<1>;

    sum<0>, c = ADD(x<0>, b<0>);
    sum<1>, c = ADDSTAGE(x<1>, y<1>, c);
    sum<2>, c = ADDSTAGE(x<2>, y<2>, c);
    sum<3>, c = ADDSTAGE(x<3>, y<3>, c);
    sum<4> = c;

    RETURN sum;
}
```

```

INPUT a<4>@4, b<4>@0;
OUTPUT sum<5>@2;

sum = ADD4(a, b);

```

This program was optimized with a variety of annealing schedules. The schedules are listed in Table 4.1. For each schedule, the graph of iteration number vs. cost is plotted in the following figures. Trial 0 was performed with no optimization, to provide a base case, and produced an array with 275 processors as shown in the left hand diagram of Figure 4.1. The other six trials were performed with increasingly large starting temperatures and more iterations. The final trial produced an array with 108 processors as shown in the right hand diagram of Figure 4.1. The annealing temperature was multiplied at each iteration by the multiplier in the table, causing the temperature to follow a  $\frac{1}{x}$  graph during the annealing process. This annealing schedule is better than a simple linear schedule since more time is spent at lower temperatures, when only relatively good mutations are accepted.

| Trial | Starting Temp | Multiplier | Iterations | Final Size | Annealing Graph Fig. |
|-------|---------------|------------|------------|------------|----------------------|
| 0     | N/A           | N/A        | 0          | 275        | N/A                  |
| 1     | 1             | .99        | 100        | 175        | 4.2                  |
| 2     | 10            | .99        | 500        | 175        | 4.3                  |
| 3     | 25            | .99        | 750        | 175        | 4.4                  |
| 4     | 40            | .99        | 1000       | 155        | 4.5                  |
| 5     | 55            | .996       | 1250       | 124        | 4.6                  |
| 6     | 70            | .999       | 3750       | 108        | 4.7                  |

Table 4.1: Simulated annealing trials for the four-bit ripple carry adder.

One main conclusion can be drawn from these graphs. As the initial temperature increases, and the annealing is allowed to take place over more iterations, the resultant system is more optimized. The first annealing schedule, the results of which are shown in Figure 4.2, does not allow any shifts to a “worse” system, and thus the system cannot escape from the local minima it is initially placed in. However, as the system becomes more “excited” during the beginning of the annealing process, as in Figure 4.7, it can escape from local minima and fall into a better local minima or, with some luck, a global minima. This illustrates the primary feature of simulated annealing, the ability to avoid local minima by exciting the system and then allowing it to “cool” gradually over a long period of time.

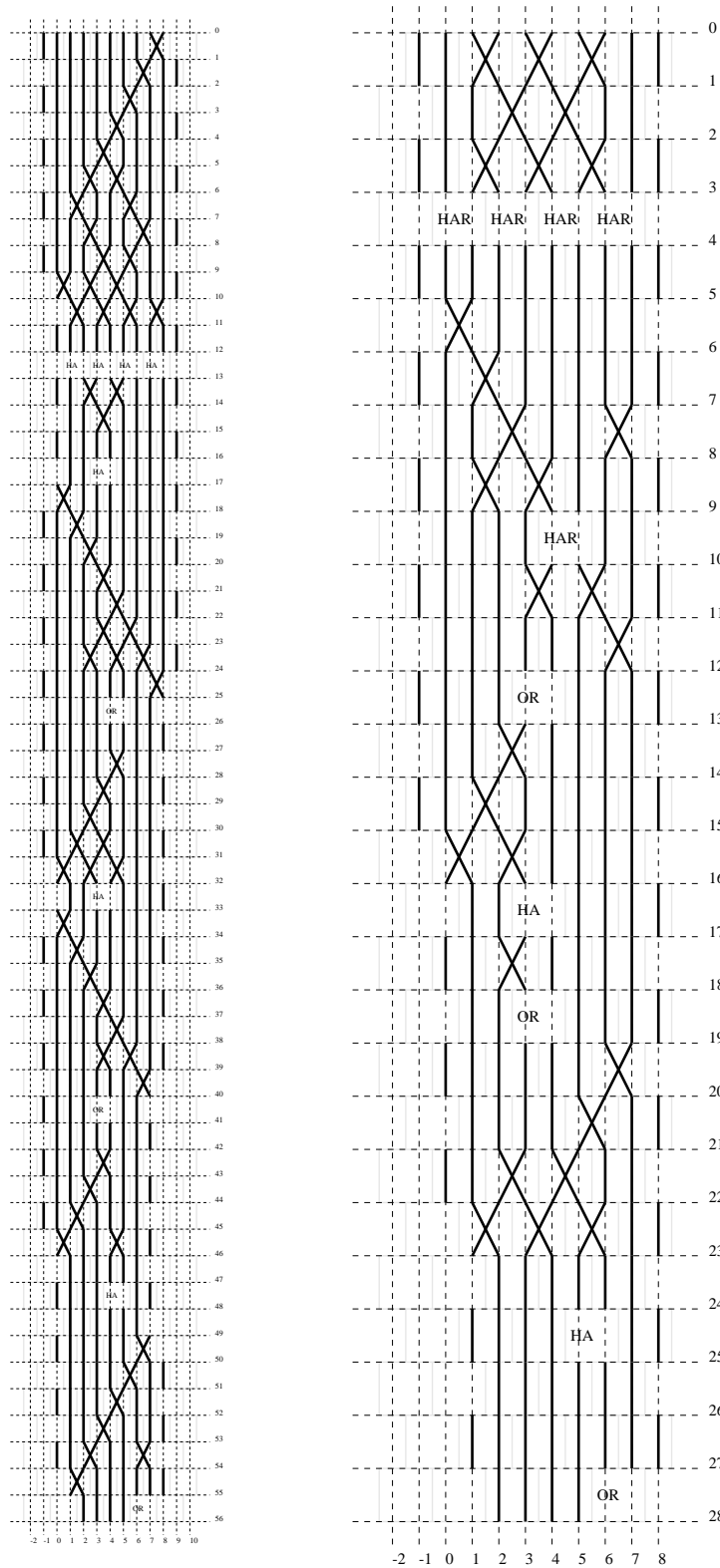
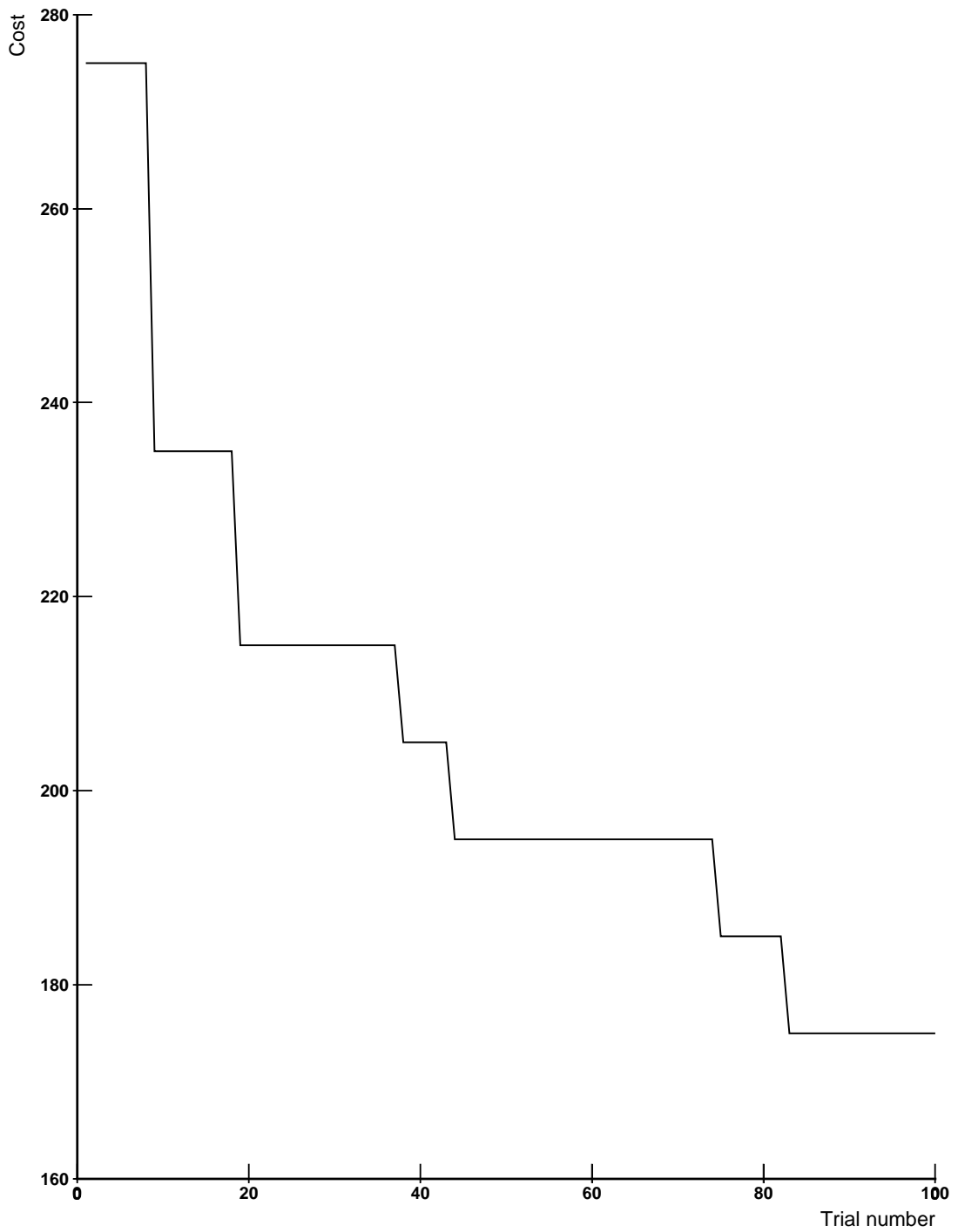


Figure 4.1: The original four-bit ripple carry adder array, and the array after 3,750 optimization iterations.



Simulated Annealing: Start temp 1.000000, Iters 100, Mult 0.990000

Figure 4.2: Iteration vs. cost for the four-bit ripple carry adder, starting temperature 1, temperature multiplier .99, 100 iterations.



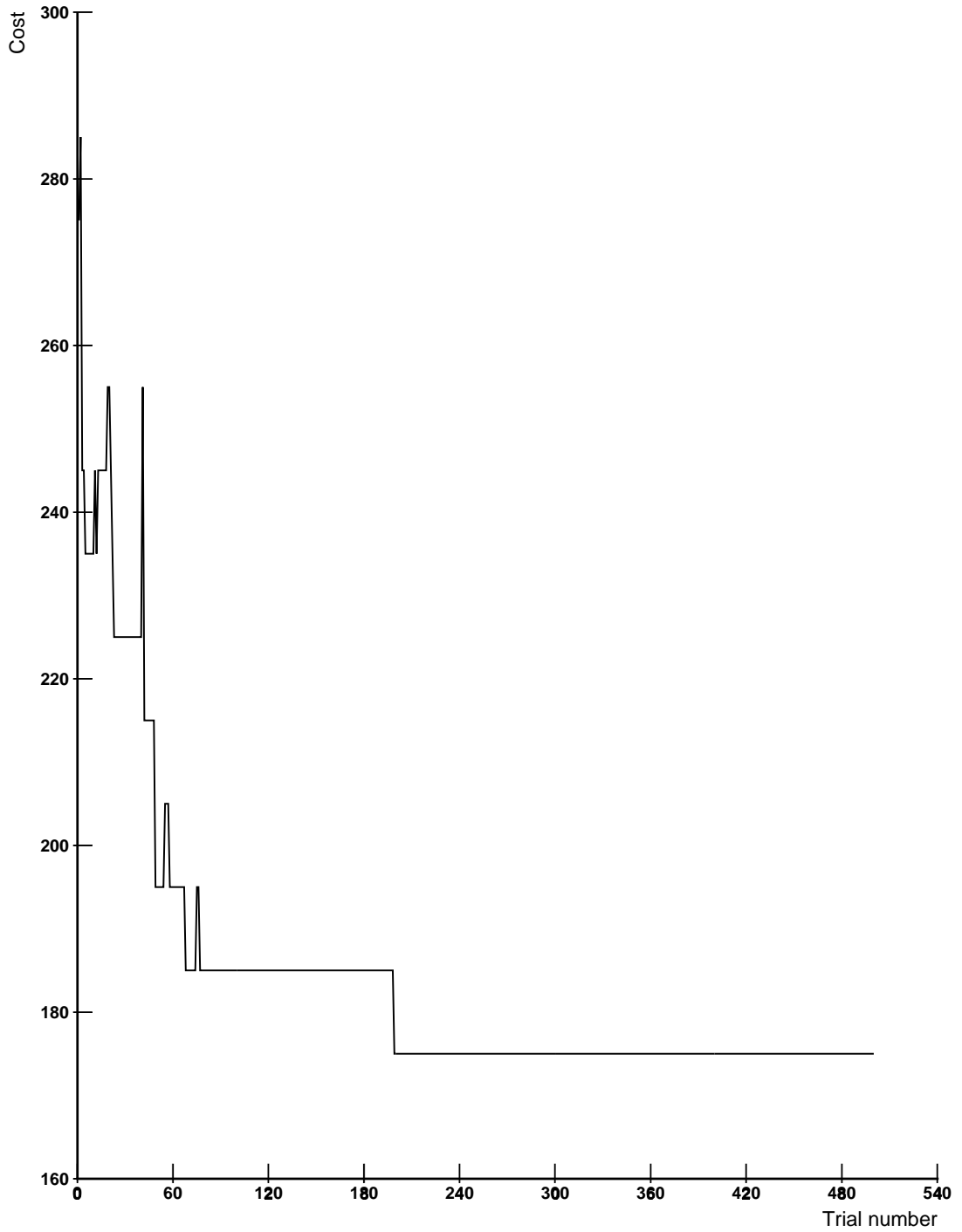


Figure 4.3: Iteration vs. cost for the four-bit ripple carry adder, starting temperature 10, temperature multiplier .99, 500 iterations.

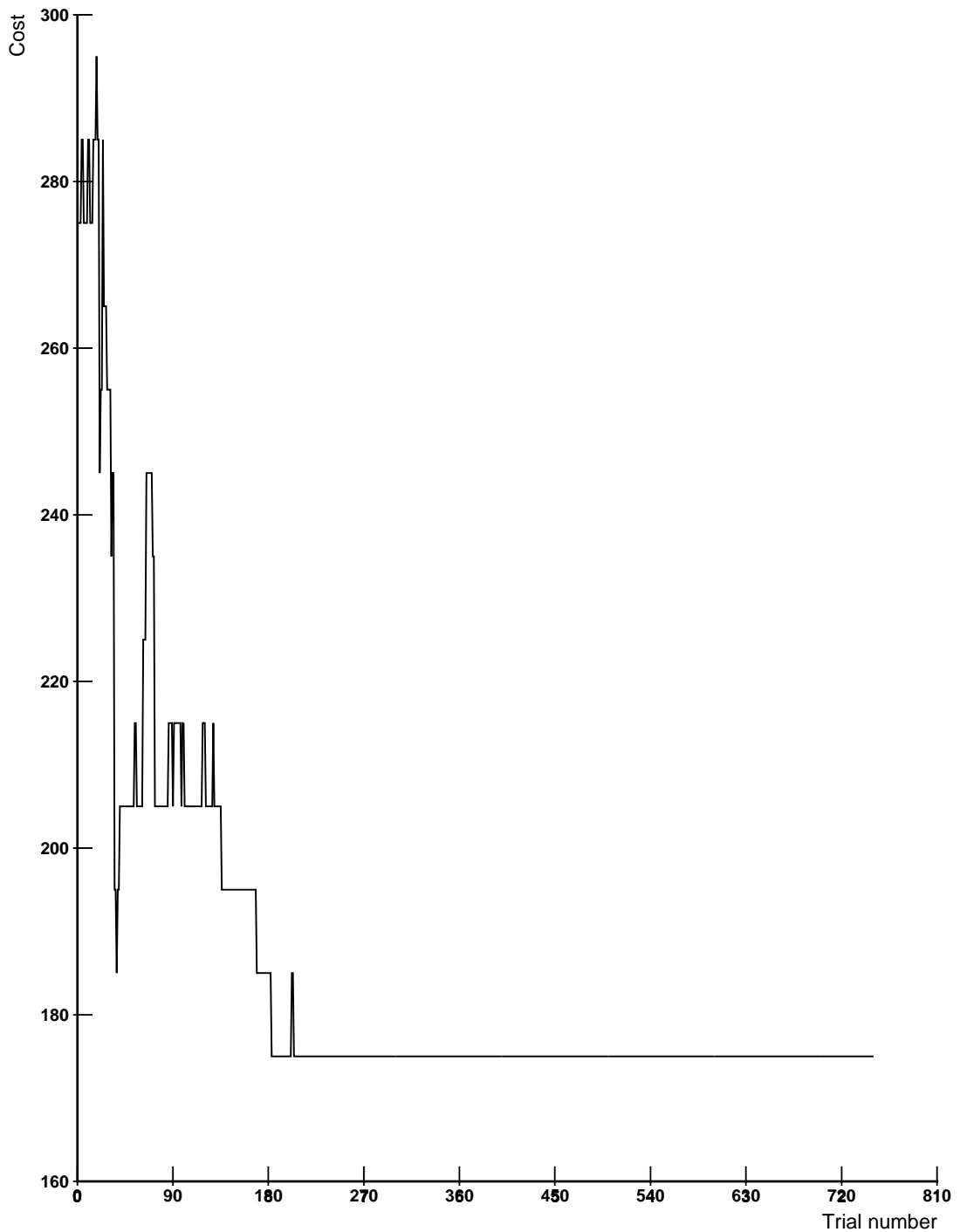
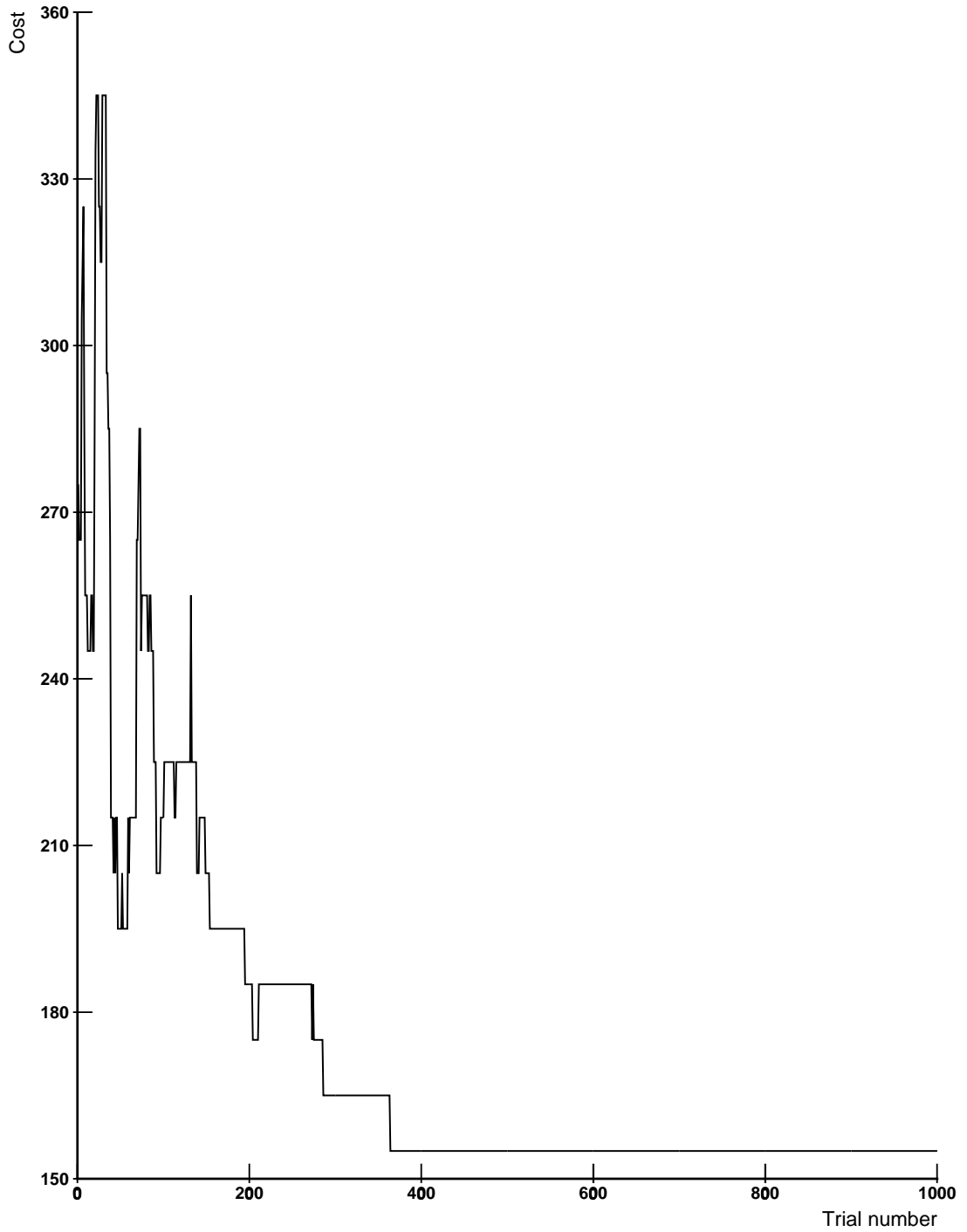
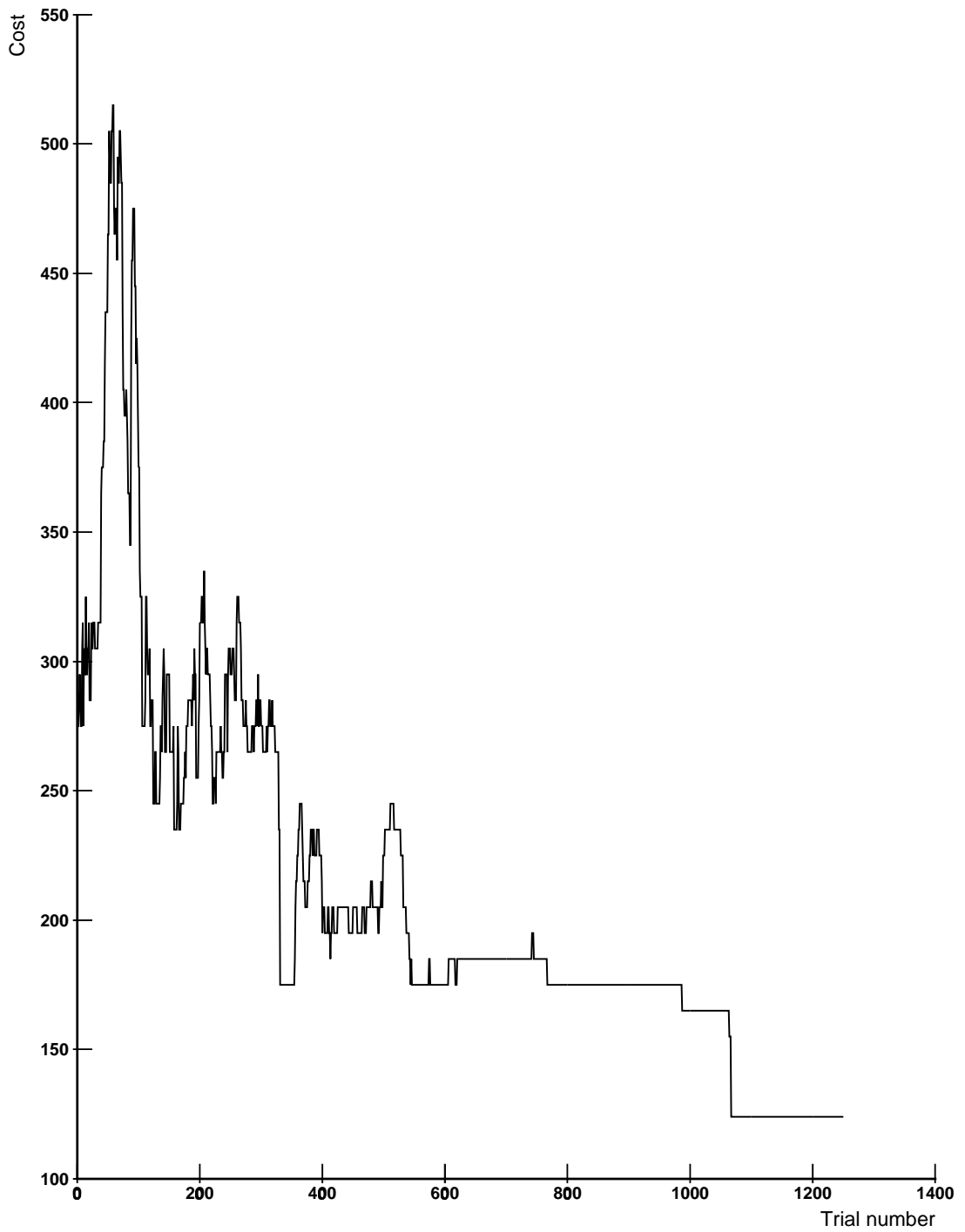


Figure 4.4: Iteration vs. cost for the four-bit ripple carry adder, starting temperature 25, temperature multiplier .99, 750 iterations.



Simulated Annealing: Start temp 40.000000, Iters 1000, Mult 0.990000

Figure 4.5: Iteration vs. cost for the four-bit ripple carry adder, starting temperature 40, temperature multiplier .99, 1,000 iterations.



Simulated Annealing: Start temp 55.000000, Iters 1250, Mult 0.996000

Figure 4.6: Iteration vs. cost for the four-bit ripple carry adder, starting temperature 55, temperature multiplier .996, 1,250 iterations.

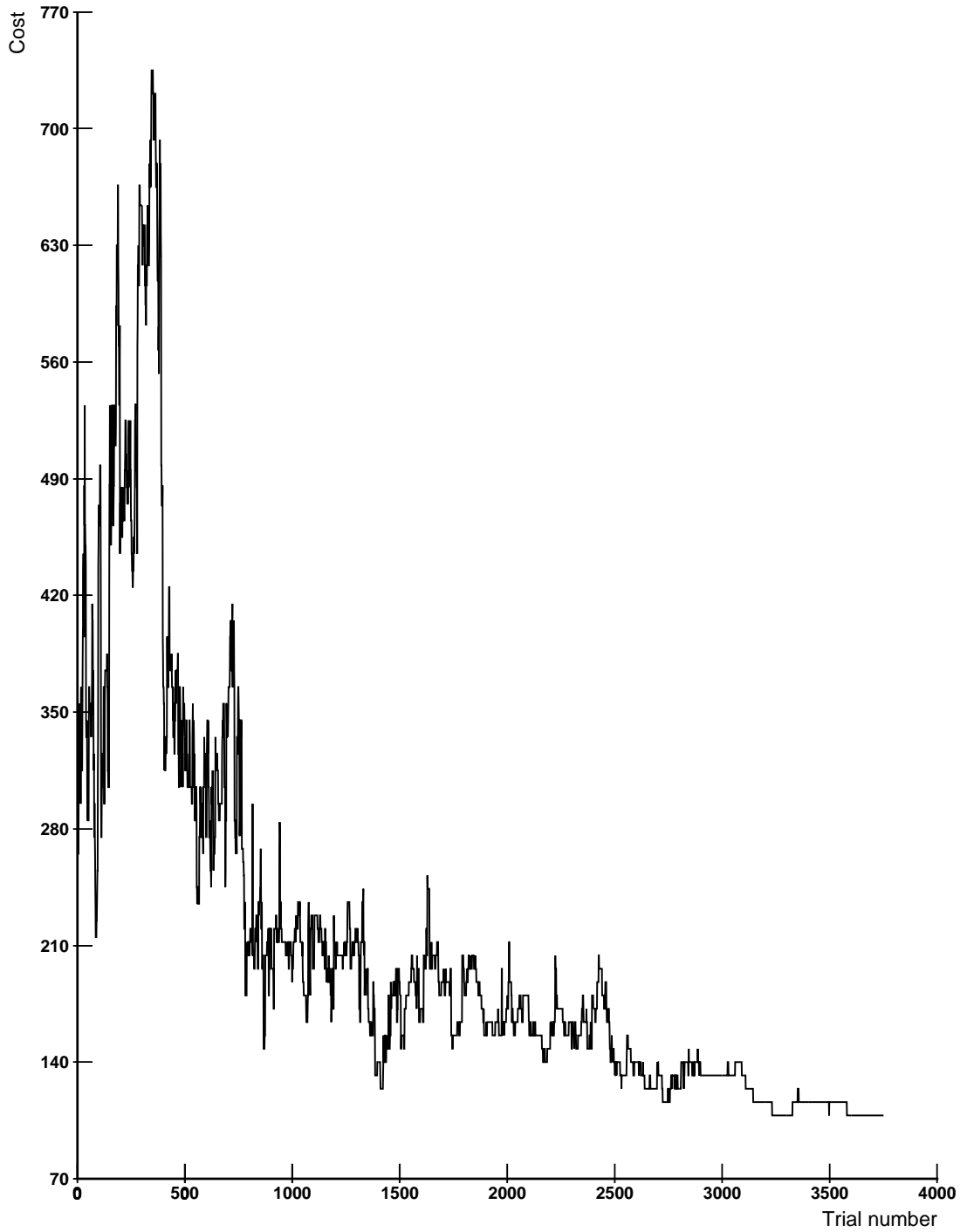


Figure 4.7: Iteration vs. cost for the four-bit ripple carry adder, starting temperature 70, temperature multiplier .999, 3,750 iterations.

# Chapter 5

## Future Work

Only a few areas of compiling for computational origami have been explored in this thesis. Many more remain. Among them are:

- A new type of input and output should be added to the language, the FLOATING INPUT and FLOATING OUTPUT. These channels would be free to move around during optimization, and would solve some of the hand-optimization problems encountered in section 3.4. Fixed position INPUTs and OUTPUTs will still be necessary for interfacing with external connections, and for making sure that outputs and inputs line up when two origami arrays are connected or the outputs of an array are fed back to the inputs to produce a finite state machine.
- The annealing schedule should be experimented with. The shape of the temperature vs. iteration curve can be very important in determining the quality of the resultant system.

- A number of aspects of the placement and routing algorithm could be improved. Specifically, the determination of goal columns needs some work since the modules are shifted to make room for wires in a very unintelligent fashion.
- Other iterative optimization methods should be investigated. For example, genetic algorithms [7, 8, 11, 24] have proven themselves worthwhile in many different applications.
- The boundary conditions of origami arrays should be analyzed and used to perform time-domain shifting of data. Among the applications that could be envisioned are parallel to serial and serial to parallel converters, time-domain multiplexers, and packet switchers. A new language should be developed to allow the programmer to express both data flow and time constraints in a concise, intuitive manner.

# Chapter 6

## Conclusion

Computational origami architectures are well-suited for some applications. However, programming a fine-grained, low-interconnect origami computer is a tedious task to do by hand. To solve this problem a compiler was introduced to automate the process of placing and routing logic elements in a two-dimensional alternate-slant origami array. Since the problem of optimally routing an origami array is probably NP-complete, and the compiler should run in polynomial time, the resultant origami array is usually suboptimal. A partial solution to this problem was presented in the form of iterative optimizations. Simulated annealing was used to selectively apply mutations to the array. This resulted in a decrease in final array size by more than 50% in some applications.

The compiler, combined with iterative optimization, provides a viable environment for developing programs for an origami array. The resultant arrays are still not optimal, but are sufficient for experimental purposes. Better optimization methods



are discussed in Chapter 5. With these improvements to the optimization method, the compiled array should approach the quality that can be achieved by hand routing.

# Bibliography

- [1] BARMAK, B. Routing in data networks using simulated annealing. Master's thesis, MIT Department of Mechanical Engineering, Jan. 1987.
- [2] ČERNÝ, V. A thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications* 45, 1 (Jan. 1985), 41–51.
- [3] CHUANG, I. L. Computational origami. Aug. 1988.
- [4] CHUANG, I. L. An introduction to the application of computational origami. Feb. 1989.
- [5] CHUANG, I. L. A computational origami architecture time slot interchanger. Tech. rep., AT&T Bell Laboratories, Feb. 1990.
- [6] CHUANG, I. L., AND FRENCH, R. S. Karma I: An origami architecture computer. Dec. 1988.
- [7] COHOON, J. P., AND PARIS, W. D. Genetic placement. In *Proceedings IEEE International Conference on Computer-Aided Design* (1986), pp. 422–425.
- [8] DAVIS, L., Ed. *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Publishers, 1987.
- [9] FLEISHER, H., GIRALDI, J., MARTIN, D. B., PHOENIX, R. L., AND TAVEL, M. A. Simulated annealing as a tool for logic optimization in a CAD environment. In *Proceedings IEEE International Conference on Computer-Aided Design* (1985), pp. 203–205.
- [10] FRENCH, R. S. A simple placement and routing algorithm for a two-dimensional computational origami architecture. In *Papers of the MIT-ACM Undergraduate Computer Science Conference* (Apr. 1989).
- [11] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

- [12] GROVER, L. K. A new simulated annealing algorithm for standard cell placement. In *Proceedings IEEE International Conference on Computer-Aided Design* (1986), pp. 378–380.
- [13] HOUH, H. H., N. A. WHITAKER, J., AND CHUANG, I. L. Implementation of an arbitrary finite state machine using a handful of logic gates. Tech. rep., AT&T Bell Laboratories, Nov. 1989.
- [14] HUANG, A. Architectural considerations involved in the design of an optical digital computer. *Proceedings of the IEEE* 72, 7 (July 1984), 780–786.
- [15] HUANG, A. Computational origami. Patent application, July 1987.
- [16] HUANG, A. Computational origami - the folding of circuits and systems. In *Proceedings of the 1989 Optical Computing Conference* (Feb. 1989). To appear.
- [17] KIRKPATRICK, S., GELATT, JR., C. D., AND VECCHI, M. P. Optimization by simulated annealing. *Science* 220, 4598 (May 1983), 671–680.
- [18] LINTON, M. A., CALDER, P. R., AND VLISSIDES, J. M. Interviews: A C++ graphical interface toolkit. Tech. rep., Stanford University, July 1988. CSL-TR-88-358.
- [19] LU, H. Computational origami: A geometric approach to regular multiprocessing. Master's thesis, MIT Department of Electrical Engineering and Computer Science, May 1988.
- [20] SECHEN, C. *VLSI Placement and Global Routing using Simulated Annealing*. Kluwer Academic Publishers, 1988.
- [21] STROUSTRUP, B. *The C++ Programming Language*. Addison-Wesley, Reading, MA, 1986.
- [22] VECCHI, M. P., AND KIRKPATRICK, S. Global wiring by simulated annealing. *IEEE Transactions on Computer-Aided Design CAD-2*, 4 (Oct. 1983), 215–222.
- [23] VLISSIDES, J. M., AND LINTON, M. A. Applying object-oriented design to structures graphics. In *Proceedings of the 1988 USENIX C++ Conference* (Oct. 1988), pp. 81–94.
- [24] WALBRIDGE, C. T. Genetic algorithms: What computers can learn from Darwin. *Technology Review* (Jan. 1989), 47–53.
- [25] WONG, D. F., LEONG, H. W., AND LIU, C. L. *Simulated Annealing for VLSI design*. Kluwer Academic Publishers, 1988.

# Appendix A

## The Compiler

A prototype version of the algorithms described in this thesis was implemented. The implementation was done in C++ [21] in an attempt to make the compiler modular and somewhat object oriented. The compiler consists of 7,200 lines of code, and uses the InterViews [18, 23] library for graphical output.

### A.1 The Input Language

The language understood by the compiler is a high level, procedural language. It supports a single data type, an array of bits with a predetermined size. Variable names consist of an arbitrary sequence of the characters A-Z, 0-9, and `_`. The first character must be a non-digit. Variables are used in the form *var*<*expr*>. When declaring a variable, *expr* is an integer defining the number of bits in the array. These bits are numbered starting with zero. Thus a declaration for `a<5>` would create the variable `a` with bits numbered 0, 1, 2, 3, and 4. When referencing a variable, *expr* can be a

more complex expression indicating the particular bits that are to be referenced. A single integer (representing a particular bit), a range of integers (denoted *num:num*), or a sequence of these separated by commas are valid bit specifiers. For example, the reference `b<2,5:7,10:8>` would refer (in order) to the bits 2, 5, 6, 7, 10, 9, and 8 of the variable `b`. In addition, a variable with no *<expr>* present can be used as shorthand for all bits in the array, starting with bit 0.

Variables are *dynamically* scoped, and all functions are call-by-value (it is impossible for a function to mutate one of its parameters).

A program consists of a set of (optional) function declarations followed by the main body of the program. A function declaration is specified as:

```
function(var<num>, var<num>, ...)  
{  
    ∴  
}
```

where *function* is the name of the function and *var*<*num*> ... is a set of formal arguments.

A statement can be a declaration or an assignment. Each statement must end with a semicolon (“;”). The following statements are supported:

```
DECL var<num> ...
```

DECL declares one or more variables so that they can be used later in the program. If more than one variable is declared, they are separated by commas. A DECL statement can occur anywhere in the program, but must precede the first usage

of the declared variable(s). Variables declared with DECL have no initial value, and an error will be generated if a variable is referenced before a value is assigned to it.

Example: DECL sum<8>, opcode<2>;

INPUT *var*<*num*>@*bitpos* ...

INPUT declares one or more variables that will be used as inputs to the origami array. These variables can be used in exactly the same manner as variables declared with DECL, but they are guaranteed to have an initial value. *bitpos* is either a single integer denoting the input position to be assigned to bit zero of this variable (with subsequent input positions being assigned to subsequent bits), or a list of comma separated integers enclosed in square brackets (“[” and “]”) denoting explicit input positions for each bit. If the second form is used, the number of integers between the brackets must equal the number of bits in the bit array. INPUT statements may only appear in the main body of the program; they may *not* appear in function declarations.

Example: INPUT value1<16>@10, value2<6>@[40,42,44,46,48,50];

OUTPUT *var*<*num*>@*bitpos* ...

OUTPUT declares one or more variables that will be used as outputs from the origami array. These variables may only be assigned to, and may only be assigned to once. *bitpos* is declared in exactly the same manner as in INPUT above. OUTPUT

statements may only appear in the main body of the program; they may *not* appear in function declarations.

Example: `OUTPUT result<16>@10, overflow<2>@[5,9];`

*varref* ... = *expr*

Assignment statements compute arbitrary expressions and assign the results to one or more variables. A *varref* is a set of comma separated variable references as described above. For example, `a<2>` and `b<1:3>`, `c<5>`, `d` are valid variable reference lists for the left hand side of an assignment statement. The *expr* can be a variable reference (*not* a variable reference list) or a function call. The number of bits in the result of the right hand side expression must agree with the number of bits in the left hand side variable reference list.

Function calls are specified as *name*(*arg*, *arg* ...) where each *arg* is a variable reference or function call. The arguments do not have to match up variable for variable with the variables declared in the function header. However, the number of bits must agree. The bits are assigned, in order, starting from the first bit of the first *arg* and proceeding to the last bit of the last *arg*. Thus, if function `B` takes the parameters `a<2>`, `b<2>`, and is called with `B(f<1>, g<0:1>, h<5>)`, bit 1 of `f` will be assigned to `a<0>`, bit 0 of `g` will be assigned to `a<1>`, bit 1 of `g` will be assigned to `b<0>`, and bit 5 of `h` will be assigned to `b<1>`.

Calls may also be made to routines defined in *libraries*. A library consists of a set of precomputed or hand-compiled origami arrays with fixed input and output

positions. They are useful because in many cases prepackaged routines can be smaller and more efficient than routines generated by the compiler. In addition, since there are no intrinsic functions, library routines are the only way to perform any computation.

Example: `sum<1>, carry = ADD(a<0>, b<0>, carry);`

`RETURN varref...`

`RETURN` is used to return a set of bits from a function to the calling routine. It takes a list of variable references (as described under the assignment statement above) and returns the bits in the order specified.

Example: `RETURN sum<0:7>, carry;`

Following is a short program that, given the basic logic operations `ADD` (half adder) and `OR` from a library, will add two 4-bit numbers together.

```
FULLADDER(a<1>, b<1>, c<1>)
{
    DECL sum<1>, carry<1>, sum2<1>, carry2<1>, carryout<1>;

    sum, carry = ADD(a, b);
    sum2, carry2 = ADD(sum, c);
    carryout = OR(carry, carry2);

    RETURN sum2, carryout;
}

INPUT a<8>@6, b<8>@14;
DECL sum<9>, carry<1>;
OUTPUT result<9>@10;

sum<0>, carry = ADD(a<0>, b<0>);
```



```
sum<1>, carry = FULLADDER(a<1>, b<1>, carry);
sum<2>, carry = FULLADDER(a<2>, b<2>, carry);
sum<3>, carry = FULLADDER(a<3>, b<3>, carry);
sum<4>, carry = FULLADDER(a<4>, b<4>, carry);
sum<5>, carry = FULLADDER(a<5>, b<5>, carry);
sum<6>, carry = FULLADDER(a<6>, b<6>, carry);
sum<7>, sum<8> = FULLADDER(a<7>, b<7>, carry);

result = sum;
```

## A.2 Running the Compiler

The compiler has an interactive command-line interface. Commands consist of a single command name followed by multiple, space-separated arguments. The following commands are supported:

- |                                   |   |
|-----------------------------------|---|
| <b>help</b>                       | Display all available commands.   |
| <b>arch</b> <i>[cppargs] file</i> | Load in an architecture specification, as described in section A.3. The file is fed through <code>/lib/cpp</code> first, and the <code>-D</code> , <code>-U</code> , and <code>-I</code> <code>cpp</code> options can be specified on the command line. All future operations with the compiler will use this architecture as a base. |
| <b>dir</b> <i>dir</i>             | Add <i>dir</i> to the list of directories searched when looking for library files. If no <i>dir</i> is specified, display the list of currently searched directories.   |

- echo** [*y*]*n* Turn command echo on or off. When command echo is on, all compiler commands are echoed to the standard output. This is useful when executing script files. If no argument is supplied, echo is turned on.
- file** *file* Read in script file *file*. The commands in the file are executed as if they were typed by the user. They are not echoed unless command echo has been turned on. Script files may be nested.
- lib** [*name*] ... Add the specified libraries to the list of searched libraries. The directories specified with the **dir** command are searched for a file named *klibname*. The format of a *klib* file is described in section A.4. If no library names are specified, the names of all currently loaded libraries are displayed.
- parse** [*cppargs*] *file* Parse a program. This program will be used for future routing and optimization. The file is fed through */lib/cpp* first, and the **-D**, **-U**, and **-I** *cpp* options can be specified on the command line.
- postscript** *file* Dump the current array to *file* in PostScript format.

- quit** Exit the compiler.
- route2d** Place and route the array with the algorithm described in Chapter 3. This also performs various optimizations if they are enabled.
- set *var value*** Set the internal variable *var* to *value*. The available variables are described below. If no value is specified, the variable is set to 1. If no variable or value are specified, all currently set variables are displayed.
- verbose [*y|n*]** Turn on verbose mode, which displays informational messages for some operations. If no argument is specified, verbose mode is turned on. Verbose mode is on by default.

The following variables can be set with the **set** command.

| Variable            | Default | Description   |
|---------------------|---------|---|
| show_arch           | 0       | Show the internal representation of the architecture after an <b>arch</b> command.                                  |
| flow_show_hierarchy | 0       | Show the module hierarchy after parsing/simulation but before vertical ordering.                                    |
| sim_show_hierarchy  | 0       | Show the module hierarchy after parsing/simulation and vertical ordering.   |
| optimization        | 0       | The type of optimization to be performed; “sa” for simulated annealing. No other optimizations are implemented yet. |

| Variable            | Default | Description   |
|---------------------|---------|---|
| sa_start_temp       | 10.0    | The starting temperature for simulated annealing.   |
| sa_num_iters        | 100     | The number of iterations for simulated annealing.   |
| sa_temp_scale       | .95     | The multiplier for the change in temperature from one iteration to the next.  |
| sa_cost             | size    | The cost function to use during simulated annealing; “size” for total array size, “crosses” for number of wire crosses and horizontal distance, “hordist” for horizontal distance.                                  |
| sa_plotfile         | 0       | The file to send iteration vs. cost data to. The file is plotted with the “splot” package.  |
| pr2dblock_wire_dist | 1       | The minimum distance between the goal columns of wires.   |
| pr2d_show_ideal     | 0       | Show the module hierarchy after ideal placement.  |
| pr2d_display        | 0       | Graphically display the module placement.   |
| pr2d_show_nodes     | 0       | Show the module hierarchy after full placement.   |
| pr2d_show_routings  | 0       | Show the routings for the module hierarchy.   |
| show_array          | 0       | Show the array in text characters after routing.  |
| ps_scale            | 25      | The scale of PostScript output.   |
| ps_latex            | 0       | Format the PostScript output for inclusion in a $\LaTeX$ document. The figure is placed at the lower-left hand corner of the page, no page number is included, and no show-page is included.                        |
| ps_texture          | 0       | Format the PostScript output in a “texture map.” This eliminates the grid lines, processor boundaries, etc., and only displays the processor flavors. This is useful for very large arrays printed at small scales. |

| Variable  | Default | Description  |
|-----------|---------|--|
| ps_nogrid | 0       | Don't include the grid lines on the PostScript output. |

### A.3 The Architecture Specification

The compiler supports arbitrary architecture specifications. Even though only simple two-dimensional routing is currently implemented, the support is present for much more complex architectures. The architecture file defines the number of dimensions in an architecture, the processor flavors available, and the interconnection of the processors. An architecture specification must be read in before any other compiler operations are performed. The format of an architecture specification is shown below.

All white space is ignored in the file.

```

file := DIMENS integer node_defs kernel_def

node_defs := node_defs node_def

node_def := NODEDEF string INPUTS integer OUTPUTS integer
           node_flavors

node_flavors := node_flavors node_flavor

node_flavor := FLAVOR id id

kernel_def := KERNEL string INPUTS integer OUTPUTS integer
            kernel_nodes

kernel_nodes := kernel_nodes kernel_node

kernel_node := TYPE string kern_conn_in kern_conn_out

kern_conn_in := INPUTS io_list

```

```

kern_conn_out := OUTPUTS io_list

io_list := input_list input_item

io_item := KERN integer
          | NODE integer ':' integer

```

The DIMENS clause defines the number of dimensions in the architecture. The node\_defs defines the types of nodes that are available. Each type of node can have a different number of inputs and outputs and different flavors available. The flavors have both a short name (the first id) which is used for internal use, and a long function name, which can be used for external representation. The kernel\_def defines the kernel of the architecture, and how the nodes are connected within it. As an example, the alternate-slant architecture can be defined as follows:

```

DIMENS 2

NODEDEF "2D-alt-node"
inputs 2
outputs 2
FLAVOR PT passthru /* 0=0 1=1 */
FLAVOR XOVER crossover /* 0=1 1=0 */
FLAVOR RBRD rightbroadcast /* 0=1 1=1 */
FLAVOR LBRD leftbroadcast /* 0=0 1=0 */
FLAVOR AND andbothnodes /* 0=AND(0,1) 1=AND(0,1) */
FLAVOR OR orbothnodes /* 0=OR(0,1) 1=OR(0,1) */
FLAVOR NOT notbothnodes /* 0=NOT(0) 1=NOT(1) */
FLAVOR HA halfadder /* 0=XOR(0,1) 1=AND(0,1) */
FLAVOR NOOP noop /* 0=X 1=X */

KERNEL "2D-alt-slant"
inputs 3
outputs 3

```

```

type "2D-alt-node"
inputs kern 1 kern 2
outputs node 1:1 kern 2

type "2D-alt-node"
inputs kern 0 node 0:0
outputs kern 0 kern 1

```

The only portion of this description that should be confusing is the kernel definition. The “2D-alt-slant” kernel consists of two “2D-alt-node” nodes. The first node takes its inputs from outputs 1 and 2 of other kernels. Its outputs go to input 1 of the second node, and input 2 of another kernel. The second node takes its inputs from input 0 of the kernel and output 0 of the first node. Its outputs go to inputs 0 and 1 of the next kernel.

## A.4 The Library Specification

Library routines are the only way that functions can actually be performed by the compiled code. There are no intrinsic functions in the language. Library routines must be written for every basic operation that needs to be performed. A library file consists of a series of routines. Each routine consists of a name, a set of input and output specifications, a size, and a rectangular array of flavors that define the implementation of the routine for a given architecture.

A sample library routine for a two-input AND element might be:

```
AND
INPUTS 0 1
OUTPUTS 0
SIZE 1 1
AND
```

In this example, the “AND” routine has two inputs that arrive at array coordinates 1 and 2. It has one output that is present on array coordinate 1, and is 1x1 nodes large. It consists of a single AND flavor. A slightly more sophisticated example is the “ANDOR” routine, which takes four one-bit inputs, and computes the expression  $\text{result}=\text{OR}(\text{AND}(\text{in}\langle 0\rangle, \text{in}\langle 1\rangle), \text{AND}(\text{in}\langle 2\rangle, \text{in}\langle 3\rangle))$ . The library definition for this routine is:

```
ANDOR
INPUTS 1 2 3 4
OUTPUTS 2
SIZE 2 2
AND AND
NOOP OR
```

The array representation for this routine is shown in Figure A.1. In this figure the array coordinates are more obvious. The inputs are positioned at coordinates 1, 2, 3, and 4, and the output is available at coordinate 2.

The coordinate locations in the INPUTS and OUTPUTS part of the specification define the locations for the function arguments and results in order. Thus if you rearrange the coordinates the function will look the same to the programmer, but will be placed and routed differently. This can be put to good use. For example,



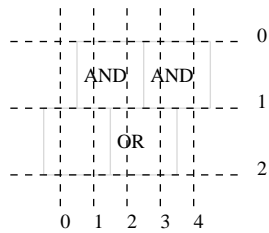


Figure A.1: The ANDOR library routine.

since AND is commutative, it is valid to swap the input signals. This could be more efficient for routing depending on how the source modules are arranged. To take advantage of this, more than one library routine with the same name can be specified in a file. The first one in the file is used by default, but the others could be used during optimization. The different modules must be the same size in order for this to work. For example, these two definitions for AND could be specified:

```

AND
INPUTS 1 2
OUTPUTS 1
SIZE 1 1
AND

AND
INPUTS 2 1
OUTPUTS 1
SIZE 1 1
AND

```

## A.5 Example

Following is a sample run of the compiler to compile a simple program, a four-bit ripple-carry adder. The program is:

```
/* return two bit result of x+y+c */

ADDSTAGE(x<1>, y<1>, c<1>)
{
    DECL sum<1>, carry<1>, sum2<1>, carry2<1>, carryout<1>;

    sum, carry = ADD(x, y);
    sum2, carry2 = ADD(sum, c);
    carryout = OR(carry, carry2);

    RETURN sum2, carryout;
}

/* add 4bits+4bits -> 5 bits */

ADD4(x<4>, y<4>)
{
    DECL sum<5>, c<1>;

    sum<0>, c = ADD(x<0>, b<0>);
    sum<1>, c = ADDSTAGE(x<1>, y<1>, c);
    sum<2>, c = ADDSTAGE(x<2>, y<2>, c);
    sum<3>, c = ADDSTAGE(x<3>, y<3>, c);
    sum<4> = c;

    RETURN sum;
}

INPUT a<4>@[0,2,4,6], b<4>@[1,3,5,7];
OUTPUT sum<5>@2;

sum = ADD4(a, b);
```

Here is a transcript of the compiler run. The resultant array is shown in Figure A.2.

This array is printed with a very small scale so that it can fit on one page, and is thus hard to read.

```
CMD> arch arch/slant
CMD> dir lib
CMD> lib test
Reading definitions from lib/klibtest
OR...AND...NOT...ADD...ANDOR...
CMD> parse tests/new/ripple-4
Simulating...
CMD> set optimization sa
CMD> set sa_start_temp 1
CMD> set sa_num_iters 10
CMD> set sa_temp_scale .95
CMD> route2d
Placement Pass 1: 0..1..2..3..4..5..6..7..8..
Placement Pass 2: *0..1..2..3..4..5..6..7..8..*
Routing: 6..5..4..3..2..1..
Temp 1, new cost 132...Picked node OR_11 for moving ideal pos
Old pos 5 new pos 4
Placement Pass 2: *0..1..2..3..4..5..6..7..8..*
Routing: 6..5..4..3..2..1..
Temp 0.95, new cost 164...Rejecting!
Picked node OR_11 for moving ideal pos
Old pos 5 new pos 6
Placement Pass 2: *0..1..2..3..4..5..6..7..8..*
Routing: 6..5..4..3..2..1..
Temp 0.9025, new cost 124...Accepting!
Picked node ADD_7 to change module - selected 1
Placement Pass 2: *0..1..2..3..4..5..6..7..8..*
Routing: 6..5..4..3..2..1..
Temp .857375, new cost 132...Rejecting!
Picked node ADD_9 to change module - selected 1
Placement Pass 2: *0..1..2..3..4..5..6..7..8..*
Routing: 6..5..4..3..2..1..
Temp .814506, new cost 132...Rejecting!
Picked node ADD_6 to change module - selected 1
Placement Pass 2: *0..1..2..3..4..5..6..7..8..*
```

```

Routing: 6..5..4..3..2..1..
Temp .773781, new cost 132...Rejecting!
Picked node OR_11 for moving ideal pos
Old pos 6 new pos 5
Placement Pass 2: *0..1..2..3..4..5..6..7..8..*
Routing: 6..5..4..3..2..1..
Temp .735092, new cost 132...Rejecting!
Picked node ADD_6 to change module - selected 1
Placement Pass 2: *0..1..2..3..4..5..6..7..8..*
Routing: 6..5..4..3..2..1..
Temp .698337, new cost 132...Rejecting!
Picked node ADD_7 to change module - selected 1
Placement Pass 2: *0..1..2..3..4..5..6..7..8..*
Routing: 6..5..4..3..2..1..
Temp 0.66342, new cost 132...Rejecting!
Picked node ADD_10 for moving ideal pos
Old pos 4 new pos 3
Placement Pass 2: *0..1..2..3..4..5..6..7..8..*
Routing: 6..5..4..3..2..1..
Temp .630249, new cost 124...Accepting!
Routing: 6..5..4..3..2..1..
SA Optimization: 10 tried, initial cost 132, final cost 124
    Final array size: 4 by 31 = 124
CMD> set ps_latex
CMD> set ps_nogrid
CMD> set ps_scale 17
CMD> postscript ripple-4.ps
Printout will take 1 by 1 = 1 pages total
Page 0,0
CMD> quit

```

## A.6 Error Messages

All errors cause the compiler to immediately abort; no error recovery is attempted.

The file name and line number on which the error was detected are given along with a backtrace of function calls within the program. The following error messages may

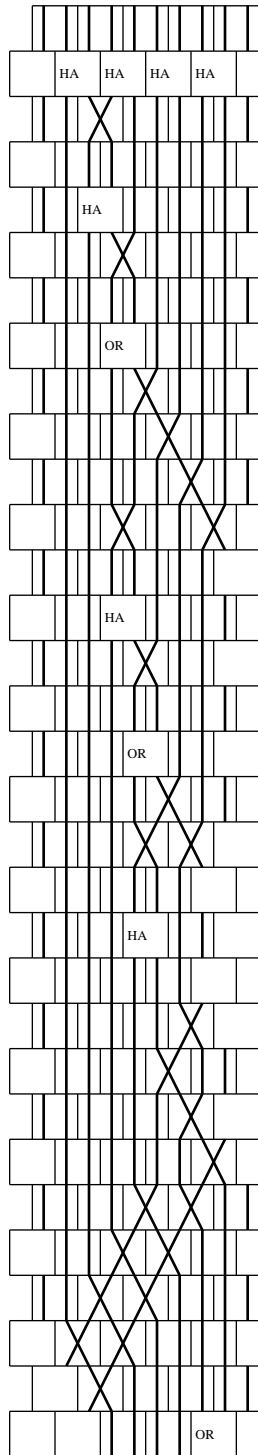


Figure A.2: The four-bit ripple carry adder array.

be generated during the compilation of a program. Other warnings may be given at different times, but will not be enumerated here.

**Reference to undeclared variable** *name*

Variable *name* was referenced but never declared.

**Multiply defined variable** *name*

Variable *name* was declared twice in the same scope.

**Out of bounds reference:** *var*<*num*>

An attempt was made to reference a bit which was not within range for the given variable. A common mistake is to declare a variable as **a<5>** and then attempt to reference the bit position **a<5>**.

**Reference to uninitialized bit** *var*<*num*>

An attempt was made to read a value from a bit which had never been initialized.

**Illegal number of bits in assignment: expecting** *num*, **got** *num*

The number of bits on the left hand side and right hand side of an assignment statement do not agree.

**INPUT may only be specified in the main body**

An INPUT declaration may only appear in the main body of the program, not in a function declaration.

**OUTPUT may only be specified in the main body**

An OUTPUT declaration may only appear in the main body of the program, not in a function declaration.

### **Number of bits declared does not agree with positions specified**

The number of bits specified in an INPUT or OUTPUT declaration does not agree with the number of explicit positions provided.

### **Can't read value from an OUTPUT**

An attempt was made to use a variable declared as an OUTPUT on the right hand side of an assignment.

### **Call to undeclared function *name***

Function *name* was not declared in the main program and could not be found in any of the specified libraries.

### **Multiply defined function *name***

Function *name* has been declared more than once in the main program.

### **Function *name* called with *num* bits, expecting *num***

The function was called with a different number of bits than its arguments needed. This could also be caused by improper use of a library routine.

### **Too many nested functions**

Too many nested function calls were made. A maximum of 100 nested function calls are allowed. Check for a recursive function call.

### **More function arguments than maximum allowed**

More bits were specified as arguments than the maximum allowed. Currently, a function may only take 500 bits worth of arguments.

### **No main body specified - nothing to compile**

No code was specified after any function declarations, and thus there is no starting point for the compilation.

### **Main function specified return values**

The main function cannot return values, since it is never called.

In addition, a number of fatal errors can be generated if there are unfixed bugs in the compiler. These are preceded by the string "INTERNAL FATAL error."





# Appendix B

## Compiler Source Code

### File Makefile

```
#
# Makefile for Origami compiler
#
# By Robert S. French
#
# Copyright (c) 1988-1990, Robert S. French
#

CCXX = g++
CC = gcc

INCLUDES = -I.
CXXFLAGS = -g ${INCLUDES}
# -DDEBUGPR2D -DDEBUGPR2DROUTE
CFLAGS = -g ${INCLUDES}
LIBS = -L/mit/gnu/vaxlib -L/mit/interviews/vaxlib -lg++
-lInterViewsX11 -lgraphic -lX11 -lm
#LIBS = -L/mit/gnu/decmipslib -lg++ -lX11 -lm

.SUFFIXES: .c .cc .h .o

.c.o:
    ${CCXX} -c ${CFLAGS} $*.c

.cc.o:
    ${CCXX} -c ${CXXFLAGS} $*.cc
```

all: oc

```
OBJS = \  
  Arch.o \  
  Array.o \  
  Code.o \  
  DAG.o \  
  Expr.o \  
  MemBlock.o \  
  Node.o \  
  SPlotFile.o \  
  SymbolTable.o \  
  VarBits.o \  
  View.o \  
  archprods.o \  
  archy.tab.o \  
  flow.o \  
  front.o \  
  function.o \  
  langprods.o \  
  langy.tab.o \  
  lex.o \  
  lib.o \  
  pr2dblock.o \  
  pr2dblockdisplay.o \  
  pr2dblockplace.o \  
  pr2dblockroute.o \  
  util.o \  
  varargs.o
```

```
SRCS = \  
  Arch.cc \  
  Code.cc \  
  DAG.cc \  
  Expr.cc \  
  MemBlock.cc \  
  Node.cc \  
  SPlotFile.cc \  
  SymbolTable.cc \  
  VarBits.cc \  
  archprods.cc \  
  archy.tab.c \  
  flow.cc \  
  front.cc \  
  function.cc \  
  langprods.cc \  
  langy.tab.c \  
  lex.cc \  
  lib.cc \  
  pr2dblock.cc \  
  pr2dblockplace.cc \  
  pr2dblockroute.cc
```

```

    util.cc \
    varargs.cc

GENFILES = \
    ConnectionMB.h \
    FlavorMB.h \
    FromNodeMB.h \
    KernNodeMB.h \
    NodeDefMB.h \
    intMB.h

TEMPSRC = \
    langy.tab.c \
    ${GENFILES}

oc: ${GENFILES} ${OBSJS}
    ${CCXX} -o oc ${CXXFLAGS} ${OBSJS} ${LIBS}

langy.tab.o: langy.tab.c

langy.tab.c: language.y
    rm -f y.tab.c y.tab.h langy.tab.c langy.tab.h
    yacc -d language.y
    sed -e 's:yy:yylang:g' -e 's:YY:YYLANG:g' < y.tab.c > langy.tab.c
    sed -e 's:yy:yylang:g' -e 's:YY:YYLANG:g' < y.tab.h > langy.tab.h
    rm -f y.tab.c y.tab.h

archy.tab.o: archy.tab.c

archy.tab.c: architecture.y
    rm -f y.tab.c y.tab.h archy.tab.c archy.tab.h
    yacc -d architecture.y
    sed -e 's:yy:yyarch:g' -e 's:YY:YYARCH:g' < y.tab.c > archy.tab.c
    sed -e 's:yy:yyarch:g' -e 's:YY:YYARCH:g' < y.tab.h > archy.tab.h
    rm -f y.tab.c y.tab.h

${GENFILES}: MB.hP
# makeclass <type> <include> <comment> < input > output
makeclass Connection Arch.h "" < MB.hP > ConnectionMB.h
makeclass Flavor Arch.h "" < MB.hP > FlavorMB.h
makeclass FromNode SymbolTable.h "" < MB.hP > FromNodeMB.h
makeclass KernNode Arch.h "" < MB.hP > KernNodeMB.h
makeclass NodeDef Arch.h "" < MB.hP > NodeDefMB.h
makeclass int "" // < SimpMB.hP > intMB.h

#

clean:
    rm -f oc ${OBSJS} ${TEMPSRC} core C++

install: all

```

```
install oc ../bin/compile
```

```
foo: foo.cc
```

```
    g++ -o foo -I. -I/mit/interviews/interviews-2.6/iv/  
installed/include foo.cc /mit/interviews/vaxlib/libInterViewsX11.a  
-lX11 -lm
```

## File makeclass

```
#!/bin/csh -f  
/bin/sed -e "s:<T>:${1}:g" -e "s:<I>:${2}:g" -e "s:<C>:${3}:g"
```

## File MB.hP

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _<T>MB_h_
#define _<T>MB_h_

#include "MemBlock.h"
<C>#include "<I>"

class <T>MB : public MemBlock {
public:
    <T>MB() : (sizeof(<T>)) {};
    <T>** get_data() { return (<T>**data; }
    <T>MB* copy() { return (<T>MB*)((MemBlock*)this->copy()); }
    <T>* last() { return (<T>*)(data+bsize*(num-lowbound-1)); }
    <T>* operator [] (int n) { return (<T>*)(data+bsize*(n-lowbound)); }
};

#endif /* _<T>MB_h_ */
```

## File SimpMB.hP

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _<T>SimpMB_h_
#define _<T>SimpMB_h_

#include "MemBlock.h"
<C>#include "<I>"

#include <stream.h>

class <T>MB : public MemBlock {
public:
    <T>MB() : (sizeof(<T>)) {};
    <T>** get_data() { return (<T>**data; }
    add(<T> n) { ((MemBlock*)this)→add(&n); }
    <T>MB* copy() { return (<T>MB*)((MemBlock*)this)→copy(); }
    <T> last() { return *((<T>*)(data+bsize*(num-1))); }
    <T>& operator [] (int n) { return *((<T>*)(data+bsize*n)); }
    friend ostream& operator << (ostream& s, <T>MB& mb) {
        for (int i=0; i<mb.num; i++) {
            s << *((<T>*)(mb.data+mb.bsize*i);
            if (i ≠ mb.num-1)
                s << ", ";
        }
        return s;
    }
};

#endif /* _<T>SimpMB_h_ */
```



## File Arch.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _arch_h_
#define _arch_h_

#include "util.h"

class ConnectionMB;
class FlavorMB;
class KernNodeMB;
class NodeDefMB;
class ostream;

extern int FLAVOR_nextnum;

class Flavor {
public:
    Flavor(char* nm, char*f) {
        name = xsave_string(nm);
        func = xsave_string(f);
        num = FLAVOR_nextnum++;
    }
    int num;
    char* name;
    char* func;
};

struct NodeDef {
    char* name;
    int inputs;
    int outputs;
    FlavorMB* flavors;
};

enum ConnType { CONN_KERNEL, CONN_NODE };

struct Connection {
    ConnType type;
    int num; /* node number */
    int bit; /* node/kern bit */
};
```

```

struct KernNode {
    char* type;
    ConnectionMB* inputs;
    ConnectionMB* outputs;
};

class Arch
{
public:
    char* name;
    int dimens;
    NodeDefMB* nodedefs;
    int kerninputs;
    int kernoutputs;
    KernNodeMB* kernnodes;
public:
    Arch();
    ~Arch();
    Flavor* lookup_flavor(char*);
    friend ostream& operator << (ostream&, Arch&);
};

#endif /* _arch_h_ */

```

## File Arch.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "Arch.h"
#include "ConnectionMB.h"
#include "FlavorMB.h"
#include "KernNodeMB.h"
#include "NodeDefMB.h"

int FLAVOR_nextnum = 0;

Arch::Arch()
{
    nodedefs = new NodeDefMB();
    kernnodes = new KernNodeMB();
}

Arch::~Arch()
{
    /* XXX This needs to free sublists! */
    delete nodedefs;
    delete kernnodes;
}

ostream& operator << (ostream& s, Arch& arch)
{
    s.form("Architecture %s (%d dimensions)\n", arch.name, arch.dimens);
    s << "Node definitions:\n";
    NodeDef* def;
    memblock_iter(arch.nodedefs, def) {
        s.form("  Node %s (%d in, %d out)\n", def->name, def->inputs,
            def->outputs);
        Flavor* flavor;
        memblock_iter(def->flavors, flavor)
            s.form("    Flavor %s\n", flavor->name);
    }
    s.form("Kernel (%d in, %d out)\n", arch.kerninputs, arch.kernoutputs);
    KernNode* knode;
    memblock_iter(arch.kernnodes, knode) {
        s.form("  Type %s\n", knode->type);
        Connection* conn;
        memblock_iter(knode->inputs, conn)
```

```

    if (conn→type == CONN_KERNEL)
        s.form("    Input from KERN%d\n", conn→bit);
    else
        s.form("    Input from NODE%d:%d\n", conn→num, conn→bit);
    memblock_iter(knode→outputs, conn)
    if (conn→type == CONN_KERNEL)
        s.form("    Output to KERN%d\n", conn→bit);
    else
        s.form("    Output to NODE%d:%d\n", conn→num, conn→bit);
}
return s;
}

Flavor* Arch::lookup_flavor(char* name)
{
    NodeDef* nd;
    memblock_iter(nodedefs, nd) {
        Flavor* flavor;
        memblock_iter(nd→flavors, flavor)
            if (!strcmp(flavor→name, name))
                return flavor;
    }
    return NULL;
}

```

## File architecture.y

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

/* These three MUST be first! */
%token      A_INTEGER A_STRING A_ID

%token      A_DIMENS A_NODEDEF A_INPUTS A_OUTPUTS A_FLAVOR A_KERNEL
%token      A_TYPE A_COORD A_KERN A_NODE

%start file

%union {
    char *s;
    int n;
}

%{
#include "archprods.h"
#include "lex.h"
#include "util.h"
%}

%%

file          : A_DIMENS A_INTEGER { aprod_dimens($2); }
              node_defs kernel_def
              ;

node_defs     : /* NULL */
              | node_defs node_def
              ;

node_def      : A_NODEDEF A_STRING A_INPUTS A_INTEGER A_OUTPUTS A_INTEGER
              { aprod_add_nodedef($2, $4, $6); } node_flavors
              ;

node_flavors  : /* NULL */
              | node_flavors node_flavor
              ;

node_flavor   : A_FLAVOR A_ID A_ID { aprod_add_flavor($2, $3); }
              ;
```

```

kernel_def      : A_KERNEL A_STRING A_INPUTS A_INTEGER A_OUTPUTS A_INTEGER
                { aprod_def_kernel($2, $4, $6); } kernel_nodes
                ;

kernel_nodes    : /* NULL */
                | kernel_nodes kernel_node
                ;

kernel_node     : A_TYPE A_STRING { aprod_add_node($2); }
                kern_conn_in kern_conn_out
                ;

kern_conn_in    : A_INPUTS input_list
                ;

input_list      : /* NULL */
                | input_list input_item
                ;

input_item      : A_KERN A_INTEGER { aprod_add_input_kern($2); }
                | A_NODE A_INTEGER ':' A_INTEGER
                { aprod_add_input_node($2,$4); }
                ;

kern_conn_out   : A_OUTPUTS output_list
                ;

output_list     : /* NULL */
                | output_list output_item
                ;

output_item     : A_KERN A_INTEGER { aprod_add_output_kern($2); }
                | A_NODE A_INTEGER ':' A_INTEGER
                { aprod_add_output_node($2,$4); }
                ;

%%

```

## File archprods.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _archprods_h_
#define _archprods_h_

class Arch;

extern Arch* new_arch;

void aprod_init();
void aprod_dimens(int);
void aprod_add_nodedef(char*, int, int);
void aprod_add_flavor(char*, char*);
void aprod_def_kernel(char*, int, int);
void aprod_add_node(char*);
void aprod_add_input_kern(int);
void aprod_add_input_node(int, int);
void aprod_add_output_kern(int);
void aprod_add_output_node(int, int);

int yyarchparse();

#endif /* _archprods_h_ */
```

## File archprods.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "Arch.h"
#include "ConnectionMB.h"
#include "FlavorMB.h"
#include "KernNodeMB.h"
#include "NodeDefMB.h"
#include "archprods.h"
#include "config.h"
#include "util.h"

Arch* new_arch = 0;

/*
 * Initialize for a new architecture specification.
 */

void aprod_init()
{
    DebugArch(">> aprod_init\n");

    new_arch = new Arch();
}

/*
 * Declare the number of dimensions in this architecture.
 */

void aprod_dimens(int ndim)
{
    DebugArch(">> aprod_dimens " << ndim << "\n");

    if (ndim < 1)
        error("Number of dimensions must be greater than 1");
    new_arch->dimens = ndim;
}

/*
 * Add a node definition.
 */
```



```

void aprod_add_nodedef(char* name, int inputs, int outputs)
{
    /* XXX Check for existing name */
    DebugArch(">> aprod_add_nodedef " << name << " in " << inputs <<
        " out " << outputs << "\n");

    if (inputs < 1)
        error("Node must have at least one input");
    if (outputs < 1)
        error("Node must have at least one output");

    NodeDef nodedef;
    nodedef.name = name;
    nodedef.inputs = inputs;
    nodedef.outputs = outputs;
    nodedef.flavors = new FlavorMB();

    new_arch->nodedefs->add(&nodedef);
}

/*
 * Add a flavor to the current node.
 */

void aprod_add_flavor(char* name, char* funcname)
{
    /* XXX Check for existing name */
    DebugArch(">> aprod_add_flavor " << name << "\n");

    Flavor flavor(name, funcname);

    new_arch->nodedefs->last()->flavors->add(&flavor);
}

/*
 * Define the kernel.
 */

void aprod_def_kernel(char* name, int inputs, int outputs)
{
    DebugArch(">> aprod_def_kernel " << name << " in " << inputs <<
        " out " << outputs << "\n");

    new_arch->name = name;
    new_arch->kerninputs = inputs;
    new_arch->kernoutputs = outputs;
}

/*
 * Add a node to the kernel definition.
 */

```

```

void aprod_add_node(char* type)
{
    DebugArch(">> aprod_add_node " << type << "\n");

    KernNode node;

    node.type = type;
    node.inputs = new ConnectionMB();
    node.outputs = new ConnectionMB();

    new_arch->kernnodes->add(&node);
}

/*
 * Add an input from a kernel input.
 */

void aprod_add_input_kern(int bit)
{
    DebugArch(">> aprod_add_input_kern " << bit << "\n");

    Connection conn;

    conn.type = CONN_KERNEL;
    conn.num = 0;
    conn.bit = bit;

    new_arch->kernnodes->last()->inputs->add(&conn);
}

/*
 * Add an input from a node output.
 */

void aprod_add_input_node(int node, int bit)
{
    DebugArch(">> aprod_add_input_node node " << node << " bit " << bit <<
              "\n");

    Connection conn;

    conn.type = CONN_NODE;
    conn.num = node;
    conn.bit = bit;

    new_arch->kernnodes->last()->inputs->add(&conn);
}

/*
 * Add an output from a kernel output.
 */

```

```

*/
void aprod_add_output_kern(int bit)
{
    DebugArch(">> aprod_add_output_kern " << bit << "\n");

    Connection conn;

    conn.type = CONN_KERNEL;
    conn.num = 0;
    conn.bit = bit;

    new_arch->kernnodes->last()->outputs->add(&conn);
}

/*
 * Add an output from a node output.
 */
void aprod_add_output_node(int node, int bit)
{
    DebugArch(">> aprod_add_input_node node " << node << " bit " << bit <<
              "\n");

    Connection conn;

    conn.type = CONN_NODE;
    conn.num = node;
    conn.bit = bit;

    new_arch->kernnodes->last()->outputs->add(&conn);
}

```

## File Array.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _array_h_
#define _array_h_

class istream;
class ostream;
class Flavor;
class MemBlock;

class Array {
private:
    int ndims;
    MemBlock* data;
public:
    Array(int);
    ~Array();
    dims() { return ndims; }
    void poke(int, int, Flavor*);
    Flavor* peek(int, int);
    void dump_ps(char* file, int startxpage=1, int endxpage=10000,
                int startypage=1, int endypage=10000);
    void getdims(int* xmin, int* xmax, int* ymin, int* ymax);
    int read_ascii(istream*, int, int);

    friend ostream& operator << (ostream&, Array&);
};

extern Flavor* null_flavor;

#endif /* _array_h_ */
```

## File Array.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <ctype.h>
#include <stdio.h>
#include <stream.h>
#include "Arch.h"
#include "Array.h"
#include "MemBlock.h"
#include "front.h"
#include "inlines.h"
#include "util.h"

Array::Array(int num)
{
    if (num < 2)
        fatal("Array::Array - Attempt to create an Array with < 2 dimensions");
    ndims = num;
    data = new MemBlock(sizeof(MemBlock*));
}

Array::~~Array()
{
    MemBlock** mb;

    memblock_iter(data, mb)
        delete (*mb);
    delete data;
}

/*
 * The following routines are currently designed only for
 * 2-dimensional array. This should be expanded in the future.
 */

Flavor* null_flavor = new Flavor("null", "noop");
Flavor* boundary_flavor = new Flavor("xxxx", "noop");

void Array::poke(int x, int y, Flavor* flavor)
{
    if (y ≥ data→next()) {
        for (int newy=data→next(); newy≤y; newy++) {
            MemBlock* mb = new MemBlock(sizeof(Flavor*));

```

```

        data->add(&mb);
    }
}
if (y < data->low()) {
    for (int newy=data->low()-1; newy≥y; newy--) {
        MemBlock* mb = new MemBlock(sizeof(Flavor*));
        data->add_low(&mb);
    }
}
MemBlock* row = memblock_get_data(data, MemBlock*, y);

if (x ≥ row->next()) {
    for (int newx=row->next(); newx≤x; newx++)
        row->add(&null_flavor);
}
if (x < row->low()) {
    for (int newx=row->low()-1; newx≥x; newx--)
        row->add_low(&null_flavor);
}
*memblock_get_datap(row, Flavor*, x) = flavor;
}

Flavor* Array::peek(int x, int y)
{
    if (y < data->low() || y ≥ data->next())
        return boundary_flavor;
    MemBlock* row = memblock_get_data(data, MemBlock*, y);
    if (x < row->low() || x ≥ row->next())
        return boundary_flavor;
    return memblock_get_data(row, Flavor*, x);
}

void Array::getdims(int* xmin, int* xmax, int* ymin, int* ymax)
{
    *xmin = *xmax = 0;
    MemBlock** rowp;
    memblock_iter(data, rowp) {
        if ((*rowp)->low() < *xmin)
            *xmin = (*rowp)->low();
        if ((*rowp)->next()-1 > *xmax)
            *xmax = (*rowp)->next()-1;
    }
    *ymin = data->low();
    *ymax = data->next()-1;
}

ostream& operator << (ostream& s, Array& a)
{
    int minx, miny, maxx, maxy;
    a.getdims(&minx, &maxx, &miny, &maxy);

```

```

s << "X: " << minx << ":" << maxx << "  Y: " <<
  miny << ":" << maxy << "\n";
for (int y=miny; y<=maxy; y++) {
  if (y & 1)
    s << " ";
  for (int x=minx; x<=maxx; x++) {
    Flavor* f = a.peek(x, y);
    if (f == null_flavor)
      s << " ";
    else if (f == boundary_flavor)
      s << " ";
    else if (f == current_arch->lookup_flavor("XOVER"))
      s << "XX";
    else if (f == current_arch->lookup_flavor("LBRD"))
      s << "|\\";
    else if (f == current_arch->lookup_flavor("RBRD"))
      s << "/|";
    else if (f == current_arch->lookup_flavor("PT"))
      s << "||";
    else
      s << "--";
    s << " ";
  }
  s << "\n";
}
return s;
}

```

```

int Array::read_ascii(istream* fp, int xsize, int ysize)
{
  for (int y=0; y<ysize; y++) {
    char bfr[2048];
    if (!fp->good())
      return 0;
    if (!fp->getline(bfr, sizeof(bfr)))
      return 0;
    if (bfr[strlen(bfr)-1] == '\n')
      bfr[strlen(bfr)-1] = '\0';
    while (isspace(*bfr))
      strcpy(bfr, bfr+1);
    while (isspace(bfr[strlen(bfr)-1]))
      bfr[strlen(bfr)-1] = '\0';
    char* ptr = bfr;
    char* ptr2;
    for (int x=0; x<xsize; x++) {
      ptr2 = index(ptr, ' ');
      if (!ptr2 && x != xsize-1)
        return 0;
      if (ptr2)
        *ptr2++ = '\0';
      Flavor* f = current_arch->lookup_flavor(ptr);
    }
  }
}

```

```

        if (!f) {
            delete fp;
            error("Can't locate flavor %s", ptr);
        }
        poke(x, y, f);
        ptr = ptr2;
    }
    if (ptr2)
        return 0;
}
}

void Array::dump_ps(char* file, int startxpage, int endxpage, int startypage,
    int endypage)
{
    void do_page(int, int, int, int, int, int);

    FILE* output = fopen(file, "w");
    if (!output)
        error("Can't open file '%s' for writing", file);

    /* SCALE is the width and height of one node */
#define SCALE 25.0

    /* Following measurements are in inches */
#define PAGESWIDTH      8.5
#define PAGEHEIGHT      11
#define XMARGIN         .5
#define YMARGIN         .75

#define STRSIZE 1024

#define MAXPERLINE ((int)((PAGESWIDTH-XMARGIN*2)*72/scale))
#define MAXPERCOLUMN (((int)((PAGEHEIGHT-YMARGIN*2)*72/scale))&~1)

#define PSHEADER "/mit/rfrench/thesis/src/compiler/PSHEADER"
#define STRLEN 1024

    char mapname[STRLEN];
    mapname[0] = '\0';

    float scale = get_param_float("ps_scale", SCALE);
    int forlatex = get_param_int("ps_latex", 0);
    int texturemap = get_param_int("ps_texture", 0);
    int doagrid = !get_param_int("ps_nogrid", 0);

    FILE* fppsheader = fopen(PSHEADER, "r");
    if (!fppsheader) {
        fclose(output);
        error("Can't open postscript header '%s'\n", PSHEADER);
    }
}

```



```

char bfr[STRLEN];
while (fgets(bfr, STRLEN, fppsheader))
    fprintf(output, "%s", bfr);
fclose(fppsheader);

int xmin, xmax, ymin, ymax;
getdimens(&xmin, &xmax, &ymin, &ymax);

int xpages = (xmax-xmin+MAXPERLINE) / MAXPERLINE;
int ypages = (ymax-ymin+MAXPERCOLUMN) / MAXPERCOLUMN;
if (endxpage > xpages)
    endxpage = xpages;
if (endypage > ypages)
    endypage = ypages;
cout << "Printout will take " << endxpage-startxpage+1 << " by " <<
    endypage-startypage+1 << " = " <<
    (endxpage-startxpage+1)*(endypage-startypage+1) <<
    " pages total\n";

if (forlatex && (endxpage-startxpage > 0 || endypage-startypage > 0))
    error("Too large for LaTeX figure...aborting");

fprintf(output, "%%\n%% Width: %d Height: %d\n%\n",
    xmax-xmin+1, ymax-ymin+1);
if (!forlatex)
    fprintf(output, "initgraphics\n");
fprintf(output, "%f %f translate\n", forlatex ? 0 : XMARGIN*72, forlatex ?
    (ymax-ymin+1)*scale : 792-YMARGIN*72);
fprintf(output, "%f %f scale\n", scale, -scale);

for (int yp=startypage-1; yp<endypage; yp++)
    for (int xp=startxpage-1; xp<endxpage; xp++) {
        fprintf(output, "%%\n%% Page %d\n%\n",
            yp*(endxpage-startxpage+1)+xp-startxpage+2);
        cout << "Page " << xp << ", " << yp << "\n";
        int x1 = xp*MAXPERLINE+xmin;
        int y1 = yp*MAXPERCOLUMN+ymin;
        int x2 = min((xp+1)*MAXPERLINE, xmax-xmin+1)+xmin;
        int y2 = min((yp+1)*MAXPERCOLUMN, ymax-ymin+1)+ymin;

        fprintf(output, "/ArrayHeight %d def\n", y2-y1-1);
        fprintf(output, "/ArrayWidth %d def\n", x2-x1);
        if (!texturemap) {
            fprintf(output, ".5 %f div setlinewidth\n", scale);
            if (doagrid) {
                fprintf(output, "/GridX %d def\n", x1*2);
                fprintf(output, "/GridY %d def\n", y2);
                fprintf(output, "gsave .9 setgray layoutpage grestore\n");
                fprintf(output, "drawgrid\n");
            }
        }
        else

```

```

        fprintf(output, "layoutpage\n");
    }
    fprintf(output, "1 %f div setlinewidth\n", scale);
    if (!forlatex) {
        fprintf(output, "gsave initgraphics /Times-Roman findfont 10 scalefont
setfont\n");
        fprintf(output, "560 765 moveto (%d,%d) show\n", xp+1, yp+1);
        if (mapname[0])
            fprintf(output, "40 765 moveto (%s) show\n", mapname);
        fprintf(output, "grestore\n");
    }

    for (int y=y1; y<y2; y++) {
        fprintf(output, "% Line %d\n", y);
        for (int x=x1; x<x2; x++) {
            fprintf(output, "%s %d %d drawkernel\n",
                peek(x, ymax-y)→func, x-x1, y-y1);
        }
    }

    if (!forlatex)
        fprintf(output, "%\ngsave showpage grestore\n");
}

fprintf(output, "%\n% The End!\n%\n");
fclose(output);
}

```

## File PSHEADER

```
%!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Postscript format for Computational Oragami sheets.
%
% Copyright (c) 1988-1990, Robert S. French
% All rights reserved.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Draw the layout for the page.

/layoutpage {
  newpath
  0 1 ArrayHeight 1 add {
    /Ypos exch def
    ArrayHeight 1 add Ypos eq {
      ArrayHeight 2 mod 0 eq {.5} {0} ifelse
    } {
      Ypos 0 eq {.5} {0} ifelse
    } ifelse
    Ypos moveto
    ArrayWidth Ypos 0 eq Ypos ArrayHeight 1 add eq or
    {0} {.5} ifelse add 0 rlineto
  } for
  stroke
  0 1 ArrayHeight {
    /Ypos exch def
    newpath
    0 1 ArrayWidth {
      Ypos 2 mod 0 eq {.5} {0} ifelse add
      Ypos moveto
      0 1 rlineto
    } for
    stroke
  } for
} def

/drawgrid {
  gsave
  0 setlinewidth
  [.11] 0 setdash
  /Times-Roman findfont .3 scalefont setfont
  0 1 ArrayHeight 1 add {
    newpath
    /Ypos exch def
    -.5 Ypos moveto
    ArrayWidth 1.5 add 0 rlineto
  }
```

```

        stroke
        ArrayWidth 1.2 add Ypos moveto GridY ArrayHeight Ypos sub sub 1 sub
        (      ) cvs
        gsave 1 -1 scale show grestore
    } for
    0 1 ArrayWidth 2 mul {
        newpath
        /Xpos exch def
        Xpos 2 div .25 add -.5 moveto 0 ArrayHeight 2 add rlineto
        stroke
        Xpos 2 div .25 add ArrayHeight 1.8 add moveto GridX Xpos add
        (      ) cvs
        gsave 1 -1 scale show grestore
    } for
    grestore
} def

%
% Translate to the proper location and execute the appropriate
% drawing function - on entry stack contains: func x y
%

/drawkernel {
    gsave
        /Ypos exch def
        /Xpos exch def
        Xpos Ypos translate
        Ypos 2 mod 1 eq not { .5 0 translate } if
        cvx exec
    grestore
} def

%
% Functions for drawing the insides of a kernel
%

% Don't do anything

/noop {
} def

% Draw a crossover

/crossover {
    .25 0 moveto .75 1 lineto stroke
    .75 0 moveto .25 1 lineto stroke
} def

% Draw a passthru

/passthru {

```

```

        .25 0 moveto .25 1 lineto stroke
        .75 0 moveto .75 1 lineto stroke
    } def

% Draw a left broadcast

/leftbroadcast {
    .75 0 moveto .75 1 lineto stroke
    .75 0 moveto .25 1 lineto stroke
} def

% Draw a right broadcast

/rightbroadcast {
    .25 0 moveto .25 1 lineto stroke
    .25 0 moveto .75 1 lineto stroke
} def

% Draw and AND, HalfAdder, or OR (for now)

/dotext {
    gsave /Times-Roman findfont .3 scalefont setfont
    .1 .6 moveto 1 -1 scale show grestore
} def

/andbothnodes {
    (AND) dotext
} def

/halfadder {
    (HA) dotext
} def

/halfadderrev {
    (HAR) dotext
} def

/orbothnodes {
    (OR) dotext
} def

/notbothnodes {
    (NOT) dotext
} def

/andpassleft {
    (ANDPL) dotext
} def

/andpassright {
    (ANDPR) dotext
} def

```

```
} def

/orpassleft {
  (ORPL) dotext
} def

/orpassright {
  (ORPR) dotext
} def

/andnotleft {
  (ANDNL) dotext
} def

/notpassleft {
  (NOTPL) dotext
} def

/notpassright {
  (NOTPR) dotext
} def
```

%%%

## File Code.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _code_h_
#define _code_h_

#include "MemBlock.h"

enum code_type { CODE_ASSIGN, CODE_DECL, CODE_INPUT, CODE_OUTPUT,
                CODE_RETURN, CODE_TYPEOUT };

class ostream;

class Code {
public:
    code_type type;
    char* filename;
    int lineno;
    MemBlock* args;
public:
    Code(code_type);
    ~Code();
    addarg(int& arg) { args→add(&arg); }
    friend ostream& operator << (ostream&, Code&);
};

#endif /* _code_h_ */
```

## File Code.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "Code.h"
#include "Expr.h"
#include "VarBits.h"

Code::Code(code_type newtype)
{
    type = newtype;
    filename = 0;
    lineno = 0;
    args = new MemBlock(sizeof(int));
}

Code::~Code()
{
    if (filename)
        delete filename;
    delete args;
}

ostream& operator << (ostream& s, Code& code)
{
    int i;
    VarBits **varbitsargs;

    switch (code.type) {
    case CODE_ASSIGN:
        for (i=1; i<code.args->size(); i++) {
            if (i != 1)
                s << " ";
            s << *memblock_get_data(code.args, VarBits*, i);
        }
        return s << " <- " << *memblock_get_data(code.args, Expr*, 0);
    case CODE_DECL:
        s.form("DECL %s<%d>", memblock_get_data(code.args, char*, 0),
            memblock_get_data(code.args, int, 1));
        return s;
    case CODE_INPUT:
        s.form("INPUT %s<%d>@[", memblock_get_data(code.args, char*, 0),
            memblock_get_data(code.args, int, 1));
    }
```



```

    for (i=2; i<code.args→size(); i++) {
        if (i ≠ 2)
            s << ", ";
        s << memblock_get_data(code.args, int, i);
    }
    return s << "]";
case CODE_OUTPUT:
    s.form("OUTPUT %s<%d>@", memblock_get_data(code.args, char*, 0),
        memblock_get_data(code.args, int, 1));
    for (i=2; i<code.args→size(); i++) {
        if (i ≠ 2)
            s << ", ";
        s << memblock_get_data(code.args, int, i);
    }
    return s << "]";
case CODE_RETURN:
    s << "RETURN";
    memblock_iter(code.args, varbitsargs)
        s << " " << **varbitsargs;
    return s;
case CODE_TYPEOUT:
    return s << "TYPEOUT " << memblock_get_data(code.args, char*, 0);
}
}

```

## File config.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _config_h_
#define _config_h_

#ifdef DEBUGLANG
#define DebugLang(x) cerr << x
#else
#define DebugLang(x)
#endif /* DEBUGLANG */

#ifdef DEBUGARCH
#define DebugArch(x) cerr << x
#else
#define DebugArch(x)
#endif /* DEBUGARCH */

#ifdef DEBUGFLOW
#define DebugFlow(x) cerr << x
#else
#define DebugFlow(x)
#endif /* DEBUGFLOW */

#ifdef DEBUGOPTIMIZE
#define DebugOptimize(x) cerr << x
#else
#define DebugOptimize(x)
#endif /* DEBUGOPTIMIZE */

#ifdef DEBUGTECH
#define DebugTech(x) cerr << x
#else
#define DebugTech(x)
#endif /* DEBUGTECH */

#ifdef DEBUGPR2D
#define DebugPR2D(x) cerr << x
#else
#define DebugPR2D(x)
#endif /* DEBUGPR2D */

#ifdef DEBUGPR2DROUTE
```

```

#define DebugPR2DRoute(x) cerr << x
#else
#define DebugPR2DRoute(x)
#endif /* DEBUGPR2DROUTE */

/* Front end constants */
#define MAXNESTFILES      20      /* Max nested files */
#define MAXARGS           50      /* Max args to a command */
#define CPP "/lib/cpp"          /* cpp command */
#define MAXCPPARGS        50      /* Max args to cpp */
#define MAXLIBSEARCHDIRS  20      /* Max library search directories */
#define MAXLIBS           50      /* Max libraries */

#define STRSIZE           256      /* Max string size for some things */

/* Flow simulation */
#define MAXFUNCARGS       500      /* Max bits of args to a function */
#define MAXFUNCNEST       100      /* Max nesting of function calls */
#define IOOFFSET          100000    /* Offset for output registers */
#define MAXARRAYWIDTH     10000    /* Max width of the output array */
#define MAXLIBOUTPUTS     100      /* Max outputs of a library routine */
#define NODEWIDTH         2        /* Width a an array node in bits */

/* Optimization */
#define OPTIMIZATION_ITERATIONS 100
#define OPTIMIZATION_START_TEMP 100.0
#define OPTIMIZATION_TEMP_SCALE .9

#endif /* _config_h_ */

```

## File ConnectionMB.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _ConnectionMB_h_
#define _ConnectionMB_h_

#include "MemBlock.h"
#include "Arch.h"

class ConnectionMB : public MemBlock {
public:
    ConnectionMB() : (sizeof(Connection)) {};
    Connection** get_data() { return (Connection**)data; }
    ConnectionMB* copy() { return (ConnectionMB*)((MemBlock*)this->copy()); }
    Connection* last() { return (Connection*)(data+bsize*(num-lowbound-1)); }
    Connection* operator [] (int n) { return (Connection*)(data+bsize*(n-lowbound)); }
};

#endif /* _ConnectionMB_h_ */
```

## File DAG.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _dag_h_
#define _dag_h_

class MemBlock;
class Node;
struct Signal;
class ostream;

class DAG {
public:
    int maxlevel;
    MemBlock* nodelist;

    DAG();
    ~DAG();
    DAG* copy();
    int add_node(Node*);
    add_signal(int, Signal*);
    void make_hierarchy();
    int get_maxlevel() { return maxlevel; }
    int level_start(int);
    int level_end(int);
    void translate_idx();
    void cleanup_unused_nodes();
    void refresh_backsignals();
    void verify();
    friend ostream& operator << (ostream&, DAG&);
};

#define NODEREF(mb, i) (*((Node**)(*mb)[i])

#endif /* _dag_h_ */
```

## File DAG.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "DAG.h"
#include "MemBlock.h"
#include "Node.h"
#include "front.h"
#include "inlines.h"
#include "intMB.h"
#include "util.h"

static int sort_by_vertmin_comp(Node**, Node**);

/*
 * Create a new DAG.
 */

DAG::DAG()
{
    nodelist = new MemBlock(sizeof(Node*));
}

/*
 * Delete an existing DAG and deallocate everything.
 */

DAG::~~DAG()
{
    Node** node;

    memblock_iter(nodelist, node)
        delete *node;
    delete nodelist;
}

/*
 * Make an exact duplicate of a DAG.
 */

DAG* DAG::copy()
{
    DAG* ret = new DAG();
```

```

    /* Duplicate all the nodes */
    for (int i=0; i<nodelist→size(); i++) {
        Node* node = new Node(*NODEREF(nodelist, i));
        ret→nodelist→add(&node);
    }
    ret→refresh_backsignals();
    Node** nodep;
    memblock_iter(ret→nodelist, nodep) {
        Signal* signal;
        memblock_iter((*nodep)→get_signals(), signal) {
            signal→goals = NULL;
            signal→goalhpos = NULL;
            signal→routegoal = NULL;
        }
    }
    ret→maxlevel = maxlevel;
    return ret;
}

/*
 * Add a Node to a DAG.
 * Once a Node is added, it belongs to the DAG. The DAG will delete it
 * when the DAG is destroyed.
 */

int DAG::add_node(Node* node)
{
    nodelist→add(&node);
    int sizem1 = nodelist→size()-1;
    NODEREF(nodelist, sizem1)→idx = sizem1;
    return sizem1;
}

/*
 * Add a Signal to a Node.
 */

DAG::add_signal(int nodeidx, Signal* signal)
{
    signal→src = nodeidx;
    NODEREF(nodelist, nodeidx)→add_signal(signal);
}

/*
 * Regenerate the backsignals
 */

void DAG::refresh_backsignals()
{
    Node** nodep;
    memblock_iter(nodelist, nodep)

```

```

    (*nodep)→get_backsignals()→reinit();

memblock_iter(nodelist, nodep) {
    Signal* signal;
    memblock_iter((*nodep)→get_signals(), signal)
        NODEREF(nodelist, signal→dest)→add_backsignal(signal);
}
}

/*
 * Sort a DAG into a dependency hierarchy.  vertmin, vertmax, and
 * curvert are assigned, and the list is sorted according to vertmin.
 */

void DAG::make_hierarchy()
{
    /*
     * First zero out each Node's flag so we'll know which one's we've
     * picked so far...vertmin is also used as a flag since we have to
     * do an entire level before marking the nodes as used.
     */
    Node** node;
    memblock_iter(nodelist, node) {
        (*node)→flag = 0;
        (*node)→vertmin = -1;
    }

    /*
     * Now iterate through the nodelist many times and assign minimum
     * vertical levels to each node.
     */
    int curlevel = 0;
    int notdone = 1;
    while (notdone) {
        notdone = 0;
        memblock_iter(nodelist, node) {
            if ((*node)→flag)
                continue;
            notdone = 1;
            Signal** backsignal;
            int nodesatisfied = 1;
            memblock_iter((*node)→get_backsignals(), backsignal) {
                if (!NODEREF(nodelist, (*backsignal)→src)→flag) {
                    nodesatisfied = 0;
                    break;
                }
            }
            if (nodesatisfied) {
                (*node)→vertmin = curlevel;
                (*node)→curvert = curlevel;
            }
        }
        curlevel++;
    }
}

```



```

    }
    memblock_iter(nodelist, node)
        if ((*node)→vertmin ≥ 0)
            (*node)→flag = 1;
    curlevel++;
}
maxlevel = curlevel-2;
/*
 * Now that we know the minimum level for all of the nodes, assign
 * the maximum level also...this is done by taking the minimum of
 * the levels of all nodes further down the hierarchy from a given node.
 */
memblock_iter(nodelist, node) {
    Signal* forwsignal;
    int minlevel = maxlevel+1;
    memblock_iter((*node)→get_signals(), forwsignal)
        minlevel = min(minlevel,
            NODEREF(nodelist, forwsignal→dest)→vertmin);
    (*node)→vertmax = minlevel-1;
}
nodelist→sort((PFRI)sort_by_vertmin_comp);
translate_idx();
}

/*
 * Find the start of a level.
 */

int DAG::level_start(int level)
{
    for (int i=0; i<nodelist→size(); i++)
        if (NODEREF(nodelist, i)→curvert == level)
            return i;
    fatal("DAG::level_start - Request for start of nonexistant level %d",
        level);
}

/*
 * Find the end of a level.
 */

int DAG::level_end(int level)
{
    for (int i=0; i<nodelist→size(); i++)
        if (NODEREF(nodelist, i)→curvert == level &&
            (i == nodelist→size()-1 || NODEREF(nodelist, i+1)→curvert == level+1))
            return i;
    fatal("DAG::level_end - Request for end of nonexistant level %d",
        level);
}

```

```

/*
 * Display a DAG.
 */

ostream& operator << (ostream& s, DAG& dag)
{
    Node** tmpnode;
    memblock_iter(dag.nodelist, tmpnode) {
        cout << **tmpnode << "\n";
        Signal* signal;
        memblock_iter((*tmpnode)→get_signals(), signal)
            cout << "\t" << signal→srcpos << " -> " <<
                *NODEREF(dag.nodelist, signal→dest) << " @ " <<
                signal→destpos << "\n";
    }
    return s;
}

/*
 * Retranslate the signals in the nodelist to reflect a new node
 * ordering (say, after a sort operation).
 */

void DAG::translate_idx()
{
    int largest = 0;
    for (int i=0; i<nodelist→size(); i++)
        if (NODEREF(nodelist, i)→idx > largest)
            largest = NODEREF(nodelist, i)→idx;

    int* trans_list = new int[largest+1];

    for (i=0; i<nodelist→size(); i++)
        trans_list[NODEREF(nodelist, i)→idx] = i;
    Node** node;
    memblock_iter(nodelist, node) {
        (*node)→idx = trans_list[(**node)→idx];
        Signal* signal;
        memblock_iter((*node)→get_signals(), signal) {
            signal→src = trans_list[signal→src];
            signal→dest = trans_list[signal→dest];
        }
        /* Back signals are pointers to forw signals, and will be
         * translated at the same time. */
    }

    delete trans_list;
}

/*
 * Sorting comparison routines.
 */

```

```

*/

static int sort_by_vertmin_comp(Node** n1, Node** n2)
{
    return (*n1)→vertmin - (*n2)→vertmin;
}

/*
 * Find nodes that have no signals coming out of them, and delete
 * them; then delete any signals that were going into that node, etc...
 */

void DAG::cleanup_unused_nodes()
{
    for (;;) {
        int changes = 0;
        for (int nodnum=0; nodnum<nodelist→size(); ) {
            Node* node = memblock_get_data(nodelist, Node*, nodnum);
            if (node→type ≠ NODE_BLOCK || node→get_signals()→size()) {
                nodnum++;
                continue;
            }
            cout << "Removing inactive node " << node→get_name() << "\n";
            changes = 1;
            Signal** bsignal;
            memblock_iter(node→get_backsignals(), bsignal) {
                Node* srcnode = NODEREF(nodelist, (*bsignal)→src);
                Signal* srcsig;
                for (int i=0; i<srcnode→get_signals()→size(); ) {
                    srcsig = memblock_get_datap(srcnode→get_signals(),
                                                Signal, i);
                    if (srcsig→dest == node→idx)
                        srcnode→get_signals()→del(i);
                    else
                        i++;
                }
            }
            nodelist→del(nodnum);
            translate_idx();
        }
        if (!changes)
            break;
    }
    this→refresh_backsignals();
}

/*
 * Verify the integrity of a DAG; used for debugging.
 */

void DAG::verify()

```

```

{
  Node** nodep;

  memblock_iter(nodelist, nodep) {
    Signal** bsignal;
    memblock_iter((*nodep)→get_backsignals(), bsignal) {
      Node* srcnode = NODEREF(nodelist, (*bsignal)→src);
      Signal* srcsignal;
      int ok = 0;
      memblock_iter(srcnode→get_signals(), srcsignal) {
        if (srcsignal == *bsignal)
          ok = 1;
      }
      if (!ok) {
        cout << "\n\nCan't find signal back from node " << **nodep <<
          "\n\n";
      }
    }
  }
}

```

## File Expr.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _expr_h_
#define _expr_h_

#include "MemBlock.h"

class VarBits;
class ostream;

class Expr {
public:
    int leaf;
    VarBits* varbits;
    char* op;
    MemBlock* children;
public:
    Expr(char*);      /* function call */
    Expr(VarBits*);  /* leaf */
    ~Expr();
    addchild(Expr* exp) { children→add(&exp); }
    friend ostream& operator << (ostream&, Expr&);
};

#endif /* _expr_h_ */
```

## File Expr.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */
```

```
#include <stream.h>
#include "Expr.h"
#include "MemBlock.h"
#include "VarBits.h"
```

```
/*
 * Make an Expr leaf from a VarBits
 */
```

```
Expr::Expr(VarBits* bits)
{
    leaf = 1;
    varbits = bits;
    op = 0;
    children = 0;
}
```

```
/*
 * Make an Expr function call
 */
```

```
Expr::Expr(char* func)
{
    leaf = 0;
    varbits = 0;
    op = func;
    children = new MemBlock(sizeof(Expr*));
}
```

```
/*
 * Delete an Expr object.
 */
```

```
Expr::~Expr()
{
    if (!leaf)
        delete children;
}
```

```
/*
```

```

* Send a representation of an Expr object to an ostream.
*/
ostream& operator << (ostream& s, Expr& expr)
{
    Expr **child;

    if (expr.leaf)
        return s << *(expr.varbits);
    s << "(" << expr.op;
    memblock_iter(expr.children, child)
        s << " " << **child;
    return s << ")";
}

```

## File FlavorMB.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _FlavorMB_h_
#define _FlavorMB_h_

#include "MemBlock.h"
#include "Arch.h"

class FlavorMB : public MemBlock {
public:
    FlavorMB() : (sizeof(Flavor)) {};
    Flavor** get_data() { return (Flavor**)data; }
    FlavorMB* copy() { return (FlavorMB*)((MemBlock*)this)→copy(); }
    Flavor* last() { return (Flavor*)(data+bsize*(num-lowbound-1)); }
    Flavor* operator [] (int n) { return (Flavor*)(data+bsize*(n-lowbound)); }
};

#endif /* _FlavorMB_h_ */
```



## File flow.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _flow_h_
#define _flow_h_

class ostream;
class DAG;
class FromNodeMB;
class Function;
class SymbolTable;

extern SymbolTable* symtab;

extern flow_print_stack(ostream&);
extern DAG* flow_simulate();

#define FLOW_NODE_INPUTS 0
#define FLOW_NODE_OUTPUTS 1

void flow_init();
DAG* flow_simulate();
FromNodeMB* flow_simfunc(Function*, FromNodeMB&);

#endif /* _flow_h_ */
```

## File flow.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "Code.h"
#include "DAG.h"
#include "Expr.h"
#include "FromNodeMB.h"
#include "MemBlock.h"
#include "Node.h"
#include "SymbolTable.h"
#include "VarBits.h"
#include "config.h"
#include "flow.h"
#include "front.h"
#include "function.h"
#include "lex.h"
#include "lib.h"
#include "util.h"

static Function *funcstack[MAXFUNCNEST];
static int curfuncnest;

SymbolTable* symtab = 0;
DAG* dag = 0;

FromNodeMB* flow_simexpr(Expr*);
FromNodeMB* flow_code_return(Code*);
FromNodeMB* flow_simfunc(Function*, FromNodeMB&);
int flow_code_decl(Code*), flow_code_input(Code*), flow_code_output(Code*);
int flow_code_assign(Code*), flow_code_typeout(Code*);

/*
 * Initialize the flow routines.
 */

void flow_init()
{
    curfuncnest = 0;
    delete symtab;
    symtab = new SymbolTable();

    delete dag;
```

```

    dag = new DAG();
}

/*
 * This is the beginning of the main flow analysis. Starting with the
 * function "__main__", we execute the program and create a directed
 * acyclic graph.
 */

DAG* flow_simulate()
{
    FromNodeMB* ret;

    Function* mainfunc = function_lookup("__main__");
    if (!mainfunc)
        fatal("Can't find function '__main__'");

    dag->add_node(new Node(NODE_INPUT, "*inputs*"));
    dag->add_node(new Node(NODE_OUTPUT, "*outputs*"));

    cout << "Simulating...\n";
    ret = flow_simfunc(mainfunc, FromNodeMB());

    if (ret->size()) {
        delete ret;
        error("Main function specified return values");
    }

    dag->refresh_backsignals();
    dag->cleanup_unused_nodes();
    dag->verify();

    if (get_param_int("flow_show_hierarchy", 0)) {
        cout << "Hierarchy after simulation:\n";
        cout << *dag << "\n";
    }

    return dag;
}

/*
 * Simulate a function - given a pointer to a Function and a block of
 * FromNodes to substitute for the formals, simulate the function and
 * return the FromNodeMB that the function returns.
 */

FromNodeMB* flow_simfunc(Function* func, FromNodeMB& args)
{
    Code* code;
    SymbolEntry* newsym;
    FormalArg* formal;

```

```

/*
 * If we're simulating a function that's defined in a library
 * somewhere, we don't really have that much to do...just enter
 * the appropriate function call in the DAG, create a bunch of
 * output registers, and return them.
 */

if (func→inlib) {
    if (func→libentry→inputs→size() ≠ args.size())
        error("Function %s called with %d bits, expecting %d", func→name,
            args.size(), func→libentry→inputs→size());
    int newnodeidx = dag→add_node(new Node(NODE_BLOCK, func→libentry));
    for (int i=0; i<func→libentry→inputs→size(); i++) {
        FromNode* fromnode = args[i];
        Signal signal;
        signal.dest = newnodeidx;
        signal.srcpos = fromnode→pos;
        signal.destpos = i;
        signal.goals = NULL;
        signal.goalhpos = NULL;
        signal.routegoal = NULL;
        dag→add_signal(fromnode→nodeidx, &signal);
    }
    FromNodeMB* funcret = new FromNodeMB();
    for (i=0; i<func→libentry→outputs→size(); i++) {
        FromNode fromnode;
        fromnode.nodeidx = newnodeidx;
        fromnode.pos = i;
        funcret→add(&fromnode);
    }
    return funcret;
}

/*
 * Add the function to the function stack and push a new level in
 * the symbol table.
 */
funcstack[curfuncnest++] = func;
symtab→push();

#ifdef DEBUGFLOW
    DebugFlow("Func "≪func→name≪" called with fromnodes");
    FromNode* fromnode;
    memblockiter(&args, fromnode)
        DebugFlow(" "≪*fromnode);
    DebugFlow("\nfor formal args: ");
    memblockiter(func→formals, formal)
        DebugFlow(formal→name≪"<"≪formal→size≪"> ");
    DebugFlow("\n");
#endif

```

```

/*
 * Go through the formals of this function, add each to the symbol
 * table, and give them a starting FromNode from the arg list.
 */
int formalsize = 0;
memblock_iter(func→formals, formal) {
    newsym = symtab→add(formal→name, formal→size, STAB_FORMAL);
    for (int j=0; j<formal→size; j++)
        newsym→fromnodes→add(args[j+formalsize]);
    DebugFlow(*newsym<<"\n");
    formalsize += formal→size;
}

if (formalsize ≠ args.size())
    error("Function %s called with %d bits, expecting %d", func→name,
        args.size(), formalsize);

/*
 * Now iterate through the code for this function and simulate
 * each statement.
 */
memblock_iter(func→code, code) {
    lex_set_file_line(code→filename, code→lineno);
    DebugFlow(*code<<"\n");
    switch (code→type) {
    case CODE_ASSIGN:
        flow_code_assign(code);
        break;
    case CODE_DECL:
        flow_code_decl(code);
        break;
    case CODE_INPUT:
        flow_code_input(code);
        break;
    case CODE_OUTPUT:
        flow_code_output(code);
        break;
    case CODE_RETURN:
        curfuncnest--;
        return flow_code_return(code);
    case CODE_TYPEOUT:
        flow_code_typeout(code);
        break;
    }
}

DebugFlow("Exiting " << func→name <<"\n");
DebugFlow(*symbtab);

symbtab→pop();

```

```

    /* End of function - fall through with no return values */
    curfuncnest--;
    return new FromNodeMB();
}

/*
 * Implementation of a DECL statement.
 */

flow_code_decl(Code* code)
{
    /*
     * Add the variable to the symbol table
     */
    SymbolEntry* newsym = symtab->add(memblock_get_data(code->args, char*, 0),
                                      memblock_get_data(code->args, int, 1),
                                      STAB_VAR);

    /*
     * Couldn't add it? Must be a duplicate symbol.
     */
    if (!newsym)
        error("Multiply defined variable %s",
              memblock_get_data(code->args, char*, 0));

    /*
     * Set all fromnode.nodeidx's to -1 so that it will be an
     * uninitialized variable.
     */
    FromNode tmp;
    tmp.nodeidx = -1;
    for (int i=0; i<memblock_get_data(code->args, int, 1); i++)
        newsym->fromnodes->add(&tmp);
    DebugFlow("Added " << *newsym << "\n");
}

/*
 * Implementation of an INPUT statement.
 */

flow_code_input(Code* code)
{
    /*
     * Add the variable to the symbol table
     */
    SymbolEntry* newsym = symtab->add(memblock_get_data(code->args, char*, 0),
                                      memblock_get_data(code->args, int, 1),
                                      STAB_INPUT);

    /*
     * Couldn't add it? Must be a duplicate symbol.
     */

```

```

if (!newsym)
    error("Multiply defined variable %s",
        memblock_get_data(code→args, char*, 0));

/*
 * Assign FromNodes for all bits because inputs are always
 * initialized.
 */
for (int i=0; i<memblock_get_data(code→args, int, 1); i++) {
    FromNode tmp;
    tmp.nodeidx = FLOW_NODE_INPUTS;
    tmp.pos = memblock_get_data(code→args, int, i+2);
    newsym→fromnodes→add(&tmp);
}
/*
 * Add the variable to the symbol table on level 0 also, for later
 * retrieval.
 */
newsym = symtab→add0(memblock_get_data(code→args, char *, 0),
                    memblock_get_data(code→args, int, 1),
                    STAB_INPUT);
for (i=0; i<memblock_get_data(code→args, int, 1); i++) {
    FromNode tmp;
    tmp.nodeidx = FLOW_NODE_INPUTS;
    tmp.pos = memblock_get_data(code→args, int, i+2);
    newsym→fromnodes→add(&tmp);
}
DebugFlow("Added " << *newsym << "\n");
}

/*
 * Implementation of an OUTPUT statement.
 */

flow_code_output(Code* code)
{
    /*
     * Add the variable to the symbol table
     */
    SymbolEntry* newsym = symtab→add(memblock_get_data(code→args, char *, 0),
                                    memblock_get_data(code→args, int, 1),
                                    STAB_OUTPUT);

    /*
     * Couldn't add it? Must be a duplicate symbol.
     */
    if (!newsym)
        error("Multiply defined variable %s",
            memblock_get_data(code→args, char*, 0));

    /*
     * Set all fromnode.nodeidx's to -1 so that it will be an

```

```

    * uninitialized variable. Set the pos to the output position so
    * we'll know which output we're talking about.
    */
    FromNode tmp;
    tmp.nodeidx = -1;
    for (int i=0; i<memblock_get_data(code→args, int, 1); i++) {
        tmp.pos = memblock_get_data(code→args, int, i+2);
        newsym→fromnodes→add(&tmp);
    }

    /*
    * Add the variable to the symbol table on level zero also for
    * later retrieval.
    */
    newsym = symtab→add0(memblock_get_data(code→args, char *, 0),
                        memblock_get_data(code→args, int, 1),
                        STAB_OUTPUT);
    for (i=0; i<memblock_get_data(code→args, int, 1); i++) {
        tmp.pos = memblock_get_data(code→args, int, i+2);
        newsym→fromnodes→add(&tmp);
    }

    DebugFlow("Added " <<*newsym<<"\n");
}

/*
* Implementation of an assignment statement.
*/

flow_code_assign(Code* code)
{
    /*
    * First, simulate the expression...
    */
    FromNodeMB* funcret = flow_simexpr(memblock_get_data(code→args,
                                                         Expr *, 0));

    /*
    * Now assign the returned FromNodes to the VarBits in order
    */
    int varbitsassigned = 0;
    for (int varbitsnum = 1; varbitsnum < code→args→size();
         varbitsnum++) {
        VarBits* destvarbits = (VarBits *)memblock_get_data(code→args,
                                                            VarBits*,
                                                            varbitsnum);

        intMB* realbits = destvarbits→expand();
        SymbolEntry* destsym = symtab→lookup_err(destvarbits→name);

        for (int i=0; i<realbits→size(); i++) {
            int bitnum = (*realbits)[i];
            if (bitnum < 0 || bitnum ≥ destsym→bitwidth)

```



```

        error("Out of bounds reference: %s<%d>", destsym→name,
            bitnum);
    if (varbitsassigned < funcret→size()) {
        if (destsym→attrib ≠ STAB_OUTPUT)
           >(*destsym→fromnodes)[bitnum] =
               >(*funcret)[varbitsassigned];
        else {
            Signal signal;
            signal.dest = FLOW_NODE_OUTPUTS;
            signal.destpos = (*destsym→fromnodes)[bitnum]→pos;
            signal.srcpos = (*funcret)[varbitsassigned]→pos;
            signal.goals = NULL;
            signal.goalhpos = NULL;
            signal.routegoal = NULL;
            dag→add_signal((*funcret)[varbitsassigned]→nodeidx,
                &signal);
        }
    }
    varbitsassigned++;
}
delete realbits;
}
if (funcret→size() ≠ varbitsassigned)
    error("Illegal number of bits in assignment: expecting %d, got %d",
        varbitsassigned, funcret→size());

delete funcret;
}

/*
 * Implementation of a RETURN statement.
 */

FromNodeMB *flow_code_return(Code* code)
{
    DebugFlow("Returning\n");

    /*
     * Create a FromNodeMB to for the returned FromNodes.
     */
    FromNodeMB* funcret = new FromNodeMB();
    /*
     * Iterate through the varbits list, adding FromNode's to the list as
     * we go.
     */
    VarBits** retvarbits;
    memblock_iter(code→args, retvarbits) {
        SymbolEntry* destsym = symtab→lookup_err((*retvarbits)→name);
        /*
         * Iterate through the bits in this varbits
         */
    }
}

```

```

    intMB* realbits = (*retvarbits)→expand();
    int* bitnum;
    memblock_iter(realbits, bitnum) {
        if (*bitnum < 0 || *bitnum ≥ destsym→bitwidth)
            error("Out of bounds reference: %s<%d>", destsym→name,
                *bitnum);
        funcret→add((*destsym→fromnodes)[*bitnum]);
    }
}
symtab→pop();

return funcret;
}

/*
 * Implementation of a TYPEOUT statement.
 */

flow_code_typeout(Code* code)
{
    cout << "% " << memblock_get_data(code→args, char*, 0) << "\n";
}

/*
 * Simulate an expression, and return a FromNodeMB of the returned
 * values.
 */

FromNodeMB* flow_simexpr(Expr* expr)
{
    DebugFlow("Evaluating " << *expr << "\n");

    FromNodeMB* ret = new FromNodeMB();

    if (expr→leaf) {
        SymbolEntry* sym = symtab→lookup_err(expr→varbits→name);
        if (sym→attrib == STAB_OUTPUT)
            error("Can't read value from an OUTPUT");
        intMB* realbits = expr→varbits→expand();
        int* bitnum;
        memblock_iter(realbits, bitnum) {
            if (*bitnum < 0 || *bitnum ≥ sym→bitwidth)
                error("Out of bounds reference: %s<%d>", sym→name, *bitnum);
            if ((*sym→fromnodes)[*bitnum]→nodeidx == -1)
                error("Reference to uninitialized bit %s<%d>",
                    expr→varbits→name, *bitnum);
            ret→add((*sym→fromnodes)[*bitnum]);
        }
        delete realbits;
        return ret;
    }
}

```

```

FromNodeMB args;
Expr** child;
memblock_iter(expr→children, child) {
    FromNodeMB* retargs = flow_simexpr(*child);
    FromNode* fromnode;
    memblock_iter(retargs, fromnode)
        args.add(fromnode);
    delete retargs;
}
Function* func = function_lookup(expr→op);
if (!func)
    error("Call to undeclared function %s", expr→op);
FromNodeMB* retargs = flow_simfunc(func, args);
return retargs;
}

/*
 * Dump the call stack out (for error processing).
 */

flow_print_stack(ostream& s)
{
    if (!curfuncnest)
        return;
    s << "Call stack:\n";

    int width = 0;
    for (int i=0; i<curfuncnest; i++) {
        if (i) {
            if (width > 73) {
                s << "\n->";
                width = 3;
            }
            else {
                s << " -> ";
                width += 4;
            }
        }
        if (width > 77 - strlen(funcstack[i]→name)) {
            s << "\n";
            width = 0;
        }
        s << funcstack[i]→name;
        width += strlen(funcstack[i]→name);
    }
    s << "\n";
}

```

## File FromNodeMB.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _FromNodeMB_h_
#define _FromNodeMB_h_

#include "MemBlock.h"
#include "SymbolTable.h"

class FromNodeMB : public MemBlock {
public:
    FromNodeMB() : (sizeof(FromNode)) {};
    FromNode** get_data() { return (FromNode**)data; }
    FromNodeMB* copy() { return (FromNodeMB*)((MemBlock*)this)→copy(); }
    FromNode* last() { return (FromNode*)(data+bsize*(num-lowbound-1)); }
    FromNode* operator [] (int n) { return (FromNode*)(data+bsize*(n-lowbound)); }
};

#endif /* _FromNodeMB_h_ */
```

## File front.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _front_h_
#define _front_h_

#include <setjmp.h>

class Arch;
class Array;
class DAG;
class MemBlock;

extern Arch* current_arch;
extern DAG* current_dag;
extern Array* current_array;
extern int verbose;

extern jmp_buf front_env;

char* get_param(char*, char*);
int get_param_int(char*, int);
float get_param_float(char*, float);

#endif /* _front_h_ */
```

## File front.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <File.h>
#include <PlotFile.h>
#include <ctype.h>
#include <std.h>
#include <stream.h>
#include "Arch.h"
#include "Array.h"
#include "DAG.h"
#include "MemBlock.h"
#include "archprods.h"
#include "config.h"
#include "flow.h"
#include "front.h"
#include "function.h"
#include "langprods.h"
#include "lex.h"
#include "lib.h"
#include "pr2dblock.h"
#include "util.h"

Arch* current_arch = NULL;
DAG* current_dag = NULL;
Array* current_array = NULL;

jmp_buf front_env;

int verbose;
static int script_echo;
static char* searchdirs[MAXLIBSEARCHDIRS];
static int ndirs;

static istream* in_files[MAXNESTFILES];
static int nfiles;

static char temp_procfile[100];

struct ScriptVar {
    char* name;
    char* value;
};
```

```

static MemBlock* script_vars;

void cmd_arch(int, char**);
void cmd_dir(int, char**);
void cmd_echo(int, char**);
void cmd_file(int, char**);
void cmd_help(int, char**);
void cmd_lib(int, char**);
void cmd_parse(int, char**);
void cmd_postscript(int, char**);
void cmd_quit(int, char**);
void cmd_route2d(int, char**);
void cmd_set(int, char**);
void cmd_verbose(int, char**);

struct {
    char* cmd;
    void (*func)(int, char**);
} command_table[] = {
    { "?", cmd_help },
    { "arch", cmd_arch },
    { "help", cmd_help },
    { "dir", cmd_dir },
    { "echo", cmd_echo },
    { "file", cmd_file },
    { "lib", cmd_lib },
    { "parse", cmd_parse },
    { "postscript", cmd_postscript },
    { "quit", cmd_quit },
    { "route2d", cmd_route2d },
    { "set", cmd_set },
    { "verbose", cmd_verbose },
    { 0, 0 }
};

#ifdef mips
void quiet_File_error_handler(char*)
{
    extern int errno;
    errno = 0;
}
#endif

main(int argc, char* argv[])
{
    srand(time(0));
    temp_procfile[0] = '\0';

    /* Make File errors be quiet */
    set_File_error_handler(quiet_File_error_handler);

```

```

lib_init();

script_vars = new MemBlock(sizeof(ScriptVar));
script_echo = 0;
verbose = 1;
searchdirs[0] = ".";
ndirs = 1;

nfiles = 1;
in_files[0] = &cin;

while (nfiles > 0) {
    while (in_files[nfiles-1]→good()) {
        if (temp_procfile[0])
            unlink(temp_procfile);
        temp_procfile[0] = '\0';
        char inbfr[STRSIZE];
        char* startbfr;
        if (in_files[nfiles-1] == &cin)
            cout << "CMD> ";
        in_files[nfiles-1]→getline(inbfr, sizeof inbfr);
        inbfr[strlen(inbfr)-1] = '\0';
        if (script_echo && in_files[nfiles-1] ≠ &cin)
            cout << "EXEC> " << inbfr << "\n";
        startbfr = inbfr;
        while (*startbfr && isspace(*startbfr))
            startbfr++;
        if (!*startbfr)
            continue;
        char *theargv[MAXARGS];
        int theargc = 0;
        for (char* ptr=startbfr; *ptr; ptr++) {
            if (*ptr ≠ ' ')
                continue;
            *ptr++ = '\0';
            while (*ptr && *ptr == ' ')
                ptr++;
            if (!*ptr)
                break;
            theargv[theargc] = ptr;
            theargc++;
        }
        if (setjmp(front_env) > 0) {
            if (nfiles > 1) {
                cout << "*** Aborting due to error **\n";
                for (int i=1; i<nfiles; i++)
                    if (in_files[i] ≠ &cin)
                        in_files[i]→close();
                nfiles = 1;
            }
            continue;
        }
    }
}

```



```

    }
    for (int i=0; command_table[i].cmd; i++) {
        if (!strcasecmp(startbfr, command_table[i].cmd)) {
            command_table[i].func(theargc, theargv);
            break;
        }
    }
    if (!command_table[i].cmd)
        cout << "Unknown command: " << startbfr << "\n";
}
if (verbose && in_files[nfiles-1] != &cin)
    cout << "Leaving called file " << in_files[nfiles-1]→name() <<
        "\n";
if (in_files[nfiles-1] != &cin)
    in_files[nfiles-1]→close();
nfiles--;
}
}

void cmd_help(int argc, char* argv[])
{
    cout << "Available commands:\n";
    cout << "help, ?           - Display this help message\n";
    cout << "arch [cppargs] <file>    - Parse architecture spec\n";
    cout << "dir [dir] ...           - Add [dir] to the library prefix list\n";
    cout << "dir                       - Show the library prefix list\n";
    cout << "echo [y|n]              - Turn command echo on or off\n";
    cout << "file <file>             - Load command file <file>\n";
    cout << "lib [lib] ...           - Add [lib] to the searched library list\n";
    cout << "lib                       - Show the searched library list\n";
    cout << "parse [cppargs] <file>   - Parse <file> into a dataflow graph\n";
    cout << "postscript <file>       - Dump the current array to file in PostScript\n";
    cout << "quit                    - Exit program\n";
    cout << "route2d                  - Route a 2D architecture\n";
    cout << "set <var> <val>         - Set variable <var> to <val>\n";
    cout << "set                       - Show all set variables\n";
    cout << "verbose [y|n]           - Turn verbose mode on or off\n";
}

void cmd_quit(int argc, char* argv[])
{
    exit(0);
}

static int get_yes_no(char* s)
{
    if (s[0] == 'n' || s[0] == 'N' || !strcasecmp(s, "off"))
        return 0;
    return 1;
}

```

```

void cmd_echo(int argc, char* argv[])
{
    script_echo = get_yes_no(argc ? argv[0] : "yes");

    if (verbose)
        if (script_echo)
            cout << "Echo on\n";
        else
            cout << "Echo off\n";
}

void cmd_verbose(int argc, char* argv[])
{
    verbose = get_yes_no(argc ? argv[0] : "yes");

    if (verbose)
        cout << "Verbose on\n";
    else
        cout << "Verbose off\n";
}

void cmd_file(int argc, char* argv[])
{
    if (argc != 1) {
        cout << "Illegal number of arguments\n";
        return;
    }
    if (nfiles == MAXNESTFILES) {
        cout << "Too many nested files!  Aborting to top level\n";
        for (int i=1; i<nfiles; i++)
            in_files[i]→close();
        nfiles = 1;
        return;
    }
    in_files[nfiles] = new istream(argv[0], io_readonly, a_useonly);
    if (!in_files[nfiles]→is_open()) {
        cout << "Can't open file " << argv[0] << " for input!  Aborting to top level\n";
        for (int i=1; i<nfiles; i++)
            in_files[i]→close();
        nfiles = 1;
        return;
    }
    nfiles++;
    if (verbose)
        cout << "Calling file " << in_files[nfiles-1]→name() << "\n";
}

void process_cpp(int* argcp, char* argv[], char* temp_procfile)
{
    char* cppargs[MAXCPPARGS];
    char* infile = 0;

```

```

int numcppargs = 0;
int j, pid, status;

cppargs[numcppargs++] = "cpp";

for (int i=0; i<*argcp; i++) {
    if (*argv[i] == '-') {
        if (argv[i][1] == 'D' || argv[i][1] == 'U' ||
            argv[i][1] == 'I') {
            cppargs[numcppargs++] = argv[i];
            for (j=i; j<*argcp-1; j++)
                argv[j] = argv[j+1];
            (*argcp)--;
            continue;
        }
    }
    else {
        if (infile != 0)
            error("Only one input file may be specified");
        infile = argv[i];
    }
    i++;
}

if (!infile)
    error("No input file specified");

cppargs[numcppargs++] = infile;
cppargs[numcppargs++] = temp_procfile;
cppargs[numcppargs] = 0;

pid = vfork();
if (pid == -1)
    fatal("No more processes!\n");
if (pid == 0) {
    execv(CPP, cppargs);
    fatal("Can't find %s", CPP);
    fflush(stdout);
    _exit(100);
}
while (pid != wait(&status))
    ;
if ((status=(status&0377)) != 0 && status != 14)
    fatal("Fatal error in /lib/cpp");
}

void cmd_parse(int argc, char* argv[])
{
    if (!current_arch) {
        warning("No current architecture");
        return;
    }
}

```

```

}

strcpy(temp_procfile, "/tmp/occXXXXXX");

/* Initialize everything */
flow_init();
function_init();
lprod_init();

if (setjmp(front_env)) {
    lex_close();
    return;
}
mktemp(temp_procfile);
process_cpp(&argc, argv, temp_procfile);

/* Parse it... */
if (lex_init(temp_procfile))
    return;
lex_select_lang();
yylangparse();
current_dag → make_hierarchy();
if (get_param_int("sim_show_hierarchy", 0)) {
    cout << "Hierarchy after vertical ordering:\n";
    cout << *current_dag << "\n";
}
}

void cmd_arch(int argc, char* argv[])
{
    strcpy(temp_procfile, "/tmp/occXXXXXX");

    /* Initialize everything */
    aprod_init();

    if (setjmp(front_env)) {
        lex_close();
        return;
    }
    mktemp(temp_procfile);
    process_cpp(&argc, argv, temp_procfile);

    /* Parse it... */
    if (lex_init(temp_procfile))
        return;
    lex_select_arch();
    yyarchparse();
    delete current_arch;
    current_arch = new_arch;
    if (get_param_int("show_arch", 0))
        cout << *current_arch;
}

```

```

}

void cmd_lib(int argc, char* argv[])
{
    char filename[STRSIZE], fullpath[STRSIZE];

    if (!argc) {
        lib_dump_files(cout);
        return;
    }

    if (!current_arch) {
        cout << "No current architecture!\n";
        return;
    }

    for (int i=0; i<argc; i++) {
        strcpy(filename, "klib");
        strcat(filename, argv[i]);
        istream* fp = 0;
        int is_loaded = 0;
        for (int dir = 0; dir < ndirs; dir++) {
            strcpy(fullpath, searchdirs[dir]);
            strcat(fullpath, "/");
            strcat(fullpath, filename);
            if (lib_is_loaded(fullpath)) {
                is_loaded = 1;
                cout << "Library " << fullpath << " already loaded\n";
                break;
            }
            fp = new istream(fullpath, "r");
            if (fp->is_open())
                break;
            delete fp;
            fp = 0;
        }
        if (!fp) {
            if (!is_loaded)
                cout << "Can't find library " << argv[i] << "\n";
            continue;
        }
        if (verbose)
            cout << "Reading definitions from " << fullpath << "\n";
        lib_add_file(fullpath, fp);
    }
}

void cmd_dir(int argc, char* argv[])
{
    if (!argc) {
        if (!ndirs) {

```

```

        cout << "No search directories defined\n";
        return;
    }
    cout << "Searched directory list:\n";
    for (int i=0; i<ndirs; i++)
        cout << searchdirs[i] << "\n";
    return;
}

for (int i=0; i<argc; i++) {
    if (ndirs == MAXLIBSEARCHDIRS) {
        cout << "Too many library search directories\n";
        return;
    }
    ndirs++;
    searchdirs[ndirs-1] = xsave_string(argv[i]);
}
}

void cmd_set(int argc, char* argv[])
{
    ScriptVar* var;
    char* value;

    if (!argc) {
        memblock_iter(script_vars, var)
            cout << var->name << " = " << var->value << "\n";
        return;
    }
    if (argc < 1 || argc > 2) {
        cout << "Illegal number of argument\n";
        return;
    }
    if (argc == 1)
        value = "1";
    else
        value = argv[1];

    memblock_iter(script_vars, var) {
        if (!strcasecmp(var->name, argv[0])) {
            delete var->value;
            var->value = xsave_string(value);
            return;
        }
    }
    ScriptVar newvar;
    newvar.name = xsave_string(argv[0]);
    newvar.value = xsave_string(value);
    script_vars->add(&newvar);
}

```

```

char* get_param(char* varname, char* dflt)
{
    ScriptVar* var;

    memblock_iter(script_vars, var)
        if (!strcasecmp(var->name, varname))
            return var->value;
    return dflt;
}

int get_param_int(char* varname, int dflt)
{
    ScriptVar* var;

    memblock_iter(script_vars, var)
        if (!strcasecmp(var->name, varname))
            return atoi(var->value);
    return dflt;
}

float get_param_float(char* varname, float dflt)
{
    ScriptVar* var;

    memblock_iter(script_vars, var)
        if (!strcasecmp(var->name, varname))
            return atof(var->value);
    return dflt;
}

void cmd_route2d(int argc, char** argv)
{
    extern Array* pr2dblock_array;

    if (!current_arch) {
        warning("No current architecture");
        return;
    }
    if (!current_dag) {
        warning("No current program");
        return;
    }

    if (pr2dblock_init(current_dag))
        return;
    pr2dblock_do_everything();
    current_array = pr2dblock_array;
}

void cmd_postscript(int argc, char** argv)

```

```
{
  if (argc  $\neq$  1) {
    cout  $\ll$  "No filename specified!\n";
    return;
  }
  if (!current_array) {
    cout  $\ll$  "No current array!\n";
    return;
  }
  current_array  $\rightarrow$  dump_ps(argv[0]);
}
```



## File function.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _function_h_
#define _function_h_

class Code;
class LibEntry;
class MemBlock;
class ostream;

class Function {
public:
    char* name;
    int inlib;
    LibEntry* libentry;
    MemBlock* code;
    MemBlock* formals;
public:
    friend ostream& operator << (ostream&, Function&);
};

class FormalArg {
public:
    char* name;
    int size;
};

void function_init();
void function_start(char*);
void function_add(Code*);
Function* function_lookup(char*);
Function* function_lookup_no_lib(char*);

#endif /* _function_h_ */
```

## File function.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "Code.h"
#include "MemBlock.h"
#include "function.h"
#include "langprods.h"
#include "lib.h"

MemBlock* functions = 0;
Function* current_function;

/*
 * Initialize list of functions
 */

void function_init()
{
    delete functions; /* XXX Memory leak */
    functions = new MemBlock(sizeof(Function));
}

/*
 * Enter a function
 */

void function_start(char* name)
{
    Function func;

    func.name = name;
    func.inlib = 0;
    func.code = new MemBlock(sizeof(Code));
    func.formals = new MemBlock(sizeof(FormalArg));
    functions->add(&func);
    current_function = functions->last();
}

/*
 * Add a Code (statement) to the current function
 */
```

```

void function_add(Code* c)
{
    c→lineno = lprod_current_line();
    c→filename = xsave_string(lprod_get_filename());
    current_function→code→add(c);
}

/*
 * Lookup a function by name - look it up in the libraries if it isn't
 * found locally.
 */

Function* function_lookup(char* name)
{
    Function *ret, newfunc;
    LibEntry* libentry;

    memblock_iter(functions, ret) {
        if (!strcmp(ret→name, name))
            return ret;
    }
    libentry = lib_lookup(name);
    if (!libentry)
        return 0;

    newfunc.name = xsave_string(name);
    newfunc.inlib = 1;
    newfunc.libentry = libentry;
    functions→add(&newfunc);
    return functions→last();
}

/*
 * Lookup a function by name - don't look in the libraries.
 */

Function* function_lookup_no_lib(char* name)
{
    Function *ret;

    memblock_iter(functions, ret) {
        if (!strcmp(ret→name, name))
            return ret;
    }
    return 0;
}

/*
 * Print a function
 */

```

```
ostream& operator << (ostream& s, Function& func)
{
    Code *c;

    s << "Function " << func.name << "\n";
    if (func.inlib)
        s << *func.libentry;
    else
        memblock_iter(func.code, c)
            s << *c << "\n";
}
```

## File inlines.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _inlines_h_
#define _inlines_h_

overload min;

inline int min(int x, int y)
{
    return x < y ? x : y;
}

#endif /* _inlines_h_ */
```

## File intMB.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _intSimpMB_h_
#define _intSimpMB_h_

#include "MemBlock.h"
// -1pt #include ""

#include <stream.h>

class intMB : public MemBlock {
public:
    intMB() : (sizeof(int)) {};
    int** get_data() { return (int**)data; }
    add(int n) { ((MemBlock*)this)→add(&n); }
    intMB* copy() { return (intMB*)((MemBlock*)this)→copy(); }
    int last() { return *((int*)(data+bsize*(num-1))); }
    int& operator [] (int n) { return *((int*)(data+bsize*n)); }
    friend ostream& operator << (ostream& s, intMB& mb) {
        for (int i=0; i<mb.num; i++) {
            s << *(int*)(mb.data+mb.bsize*i);
            if (i ≠ mb.num-1)
                s << ", ";
        }
        return s;
    }
};

#endif /* _intSimpMB_h_ */
```

## File KernNodeMB.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _KernNodeMB_h_
#define _KernNodeMB_h_

#include "MemBlock.h"
#include "Arch.h"

class KernNodeMB : public MemBlock {
public:
    KernNodeMB() : (sizeof(KernNode)) {};
    KernNode** get_data() { return (KernNode**)data; }
    KernNodeMB* copy() { return (KernNodeMB*)((MemBlock*)this)→copy(); }
    KernNode* last() { return (KernNode*)(data+bsize*(num-lowbound-1)); }
    KernNode* operator [] (int n) { return (KernNode*)(data+bsize*(n-lowbound)); }
};

#endif /* _KernNodeMB_h_ */
```

## File language.y

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

/* These three MUST be first! */
%token      L_INTEGER L_STRING L_ID

%token      L_ECHO
%token      L_DECL L_INPUT L_OUTPUT L_RETURN L_TYPEOUT

%start file

%union {
    char *s;
    int n;
}

%{
#include "langprods.h"
#include "lex.h"
#include "util.h"
%}

%%

file          : function_defs main_prog
              ;

function_defs : /* NULL */
              | function_defs function_def
              | function_defs L_ECHO L_STRING ';' { lprod_do_echo($3); }
              ;

function_def  : L_ID '(' { lprod_enter_function($1); } formal_args ')'
              | '{' statements '}'
              | lprod_exit_function();
              ;

formal_args   : /* NULL */
              | formal_arg
              | formal_args ',' formal_arg
              ;

formal_arg    : L_ID '<' L_INTEGER '>'
```



```

        { lprod_add_formalarg($1, $3); }
    ;

main_prog      : /* NULL - this is a reduce/reduce conflict; ignore */
    { error("No main body specified - nothing to compile"); }
    | { lprod_enter_main(); } statements { lprod_exit_main(); }
    ;

varbits_ref    : L_ID { lprod_varbits_start($1); } varbits_spec
    { $$ = lprod_varbits_return(); }
    ;

varbits_spec   : /* NULL */ { lprod_varbits_setall(); }
    | '<' varbits_part varbits_spec2 '>'
    ;

varbits_spec2  : /* NULL */
    | varbits_spec2 ',' varbits_part
    ;

varbits_part   : L_INTEGER { lprod_varbits_add($1); }
    | L_INTEGER ':' L_INTEGER { lprod_varbits_add_range($1, $3); }
    ;

statements     : /* NULL */
    | statements statement
    ;

statement      : { lprod_set_line(); } interior ';'
    ;

interior       : declaration
    | varbits_list '=' expr { lprod_add_assignment($1, $3); }
    | return
    ;

declaration    : L_DECL decl_list
    | L_INPUT input_list
    | L_OUTPUT output_list
    | L_TYPEOUT L_STRING { lprod_add_typeout($2); }
    | L_ECHO L_STRING { lprod_do_echo($2); }
    ;

decl_list      : decl_elem
    | decl_list ',' decl_elem
    ;

decl_elem      : L_ID '<' L_INTEGER '>'
    { lprod_add_decl($1, $3); }
    ;

input_list     : input_elem

```

```

        | input_list ',' input_elem
        ;

input_elem      : L_ID '<' L_INTEGER '>' '@' L_INTEGER
                { lprod_add_input($1, $3, $6); }
        | L_ID '<' L_INTEGER '>' '@' '[' position_list ']'
                { lprod_add_input_with_list($1, $3); }
        ;

output_list     : output_elem
                | output_list ',' output_elem
                ;

output_elem     : L_ID '<' L_INTEGER '>' '@' L_INTEGER
                { lprod_add_output($1, $3, $6); }
        | L_ID '<' L_INTEGER '>' '@' '[' position_list ']'
                { lprod_add_output_with_list($1, $3); }
        ;

position_list   : { lprod_positionlist_start(); } position positions
                ;

positions       : /* NULL */
                | positions ',' position
                ;

position        : L_INTEGER { lprod_positionlist_add($1); }
                ;

return          : L_RETURN varbits_list_bl
                { lprod_add_return($2); }
                ;

varbits_list_bl : { lprod_varbitslist_start(); } varbits_args_bl
                { $$ = lprod_varbitslist_return(); }
                ;

varbits_args_bl : /* NULL */
                | varbits_arg
                | varbits_args_bl ',' varbits_arg
                ;

varbits_list    : { lprod_varbitslist_start(); } varbits_args
                { $$ = lprod_varbitslist_return(); }
                ;

varbits_args    : varbits_arg
                | varbits_args ',' varbits_arg
                ;

varbits_arg     : varbits_ref { lprod_varbitslist_add($1); }

```

```

;
expr      : varbits_ref { $$ = (int)lprod_expr_make_varbits($1); }
| func_call
;

func_call : L_ID '(' { lprod_pushinit_func($1); } call_arg_list ')'
{ $$ = (int)lprod_popexpr_func(); }
;

call_arg_list : /* NULL */
| call_arg
| call_arg_list ',' call_arg
;

call_arg : expr { lprod_func_add_arg($1); }
;

%%

```

## File langprods.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _langprods_h
#define _langprods_h_

#include "Expr.h"
#include "MemBlock.h"
#include "VarBits.h"
#include "util.h"

void lprod_init();
void lprod_set_line();
int lprod_current_line();
char* lprod_get_filename();
void lprod_varbits_start(char*);
void lprod_varbits_setall();
void lprod_varbits_add(int);
void lprod_varbits_add_range(int, int);
VarBits* lprod_varbits_return();
Expr* lprod_expr_make_varbits(VarBits*);
void lprod_pushinit_func(char*);
Expr* lprod_popexpr_func();
void lprod_func_add_arg(Expr*);
void lprod_enter_function(char*);
void lprod_add_formalarg(char*, int);
void lprod_enter_main();
void lprod_exit_function();
void lprod_exit_main();
void lprod_positionlist_start();
void lprod_positionlist_add(int);
void lprod_varbitslist_start();
void lprod_varbitslist_add(VarBits*);
MemBlock* lprod_varbitslist_return();
void lprod_add_assignment(MemBlock*, Expr*);
void lprod_add_decl(char*, int);
void lprod_add_input(char*, int, int);
void lprod_add_input_with_list(char*, int);
void lprod_add_output(char*, int, int);
void lprod_add_output_with_list(char*, int);
void lprod_add_return(MemBlock*);
void lprod_add_typeout(char*);
void lprod_do_echo(char*);
```

```
int yylangparse();  
#endif /* _langprods_h_ */
```

## File langprods.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "Code.h"
#include "DAG.h"
#include "Expr.h"
#include "MemBlock.h"
#include "Node.h"
#include "SymbolTable.h"
#include "VarBits.h"
#include "config.h"
#include "flow.h"
#include "front.h"
#include "function.h"
#include "langprods.h"
#include "lex.h"
#include "util.h"

/*
 * These variables are for storing state during nested function calls.
 * funcargs[] is a set of Expr* arguments, and funcindex[] points into
 * funcargs[]. funcops[] is the set of operations for each nested
 * function.
 */

static int funccuridx, funclevel;
static Expr* funcargs[MAXFUNCARGS];
static int funcindex[MAXFUNCNEST];
static char* funcops[MAXFUNCNEST];

/*
 * cur_varbits is used to build a VarBits.
 */

static VarBits* cur_varbits = 0;

/*
 * varbitlist is used to build a list of VarBits's for a RETURN or
 * assignment statement.
 */

static MemBlock* varbitlist = 0;
```

```

/*
 * positionlist is used to build a list of positions for INPUTs or
 * OUTPUTs.
 */

MemBlock* positionlist = 0;

/*
 * These are used for storing the current line number and filename.
 */

static int current_line;
static char* current_filename;

/*
 * Initialize internal state...Performed at beginning of a file.
 */

void lprod_init()
{
    VarBits** vb;

    DebugLang(">> lprod_init\n");

    funcidx = 0;
    funclvl = 0;
    current_line = 0;
    current_filename = (char *)0;

    /* XXX delete cur_varbits;
    cur_varbits = 0;
    if (varbitslist) {
        memblock_iter(varbitslist, vb)
            delete (*vb);
        delete varbitslist;
        varbitslist = 0;
    } */
    delete positionlist;
    positionlist = 0;
}

/*
 * Save the current lex state for future use by function_add.
 */

void lprod_set_line()
{
    current_line = lex_current_line();
    delete current_filename;
    current_filename = xsave_string(lex_get_filename());
}

```

```

    DebugLang(">> lprod_set_line "<<current_filename<<"@"<<current_line<<"\n");
}

/*
 * Return the current lex state.
 */

int lprod_current_line()
{
    return current_line;
}

char* lprod_get_filename()
{
    return current_filename;
}

/*
 * Start the definition for a VarBits... like a or a<5> or a<5,8:10>.
 */

void lprod_varbits_start(char* name)
{
    cur_varbits = new VarBits(name, 0);

    DebugLang(">> lprod_varbits_start "<<name<<"\n");
}

/*
 * Set the current VarBits to indicate all bits of the variable should
 * be used.
 */

void lprod_varbits_setall()
{
    cur_varbits->setallbits(1);

    DebugLang(">> lprod_varbits_setall\n");
}

/*
 * Add an element to the current VarBits.
 */

void lprod_varbits_add(int elem)
{
    cur_varbits->addref(elem);

    DebugLang(">> lprod_varbits_add "<<elem<<"\n");
}

```



```

/*
 * Add a range of elements to the current VarBits.
 */

void lprod_varbits_add_range(int start, int end)
{
    int i;

    DebugLang(">> lprod_varbits_add_range "<<start<<" to "<<end<<"\n");

    if (end ≥ start)
        for (i=start; i≤end; i++)
            cur_varbits→addref(i);
    else
        for (i=start; i≥end; i--)
            cur_varbits→addref(i);
}

/*
 * Return the current VarBits.
 */

VarBits* lprod_varbits_return()
{
    DebugLang(">> lprod_varbits_return "<<*cur_varbits<<"\n");

    return cur_varbits;
}

/*
 * Return an Expr node for a VarBits (a leaf).
 */

Expr* lprod_expr_make_varbits(VarBits* bits)
{
    Expr* ret = new Expr(bits);

    DebugLang(">> lprod_expr_make_varbits "<<*bits<<" => "<<*ret<<"\n");
    return ret;
}

/*
 * Enter a new function call and save where we were before.
 */

void lprod_pushinit_func(char* op)
{
    DebugLang(">> lprod_pushinit_func "<<op<<"\n");
}

```

```

    if (funclevel == MAXFUNCNEST)
        error("Too many nested functions");

    funcops[funclevel] = op;
    funcindex[funclevel++] = funccuridx;
}

/*
 * Exit a function call def and return an expression for the function
 * call.
 */

Expr* lprod_popexpr_func()
{
    int i, start, end;
    Expr *ret, *newarg;

    if (funclevel ≤ 0)
        fatal("Too many function argument pops");

    ret = new Expr(funcops[--funclevel]);

    start = funcindex[funclevel];
    end = funccuridx;

    for (i=start; i<end; i++)
        ret→addchild(funcargs[i]);

    funccuridx = start;

    DebugLang(">> lprod_popexpr_func "<<*ret<<"\n");

    return ret;
}

/*
 * Add an argument to the current function call argument list.
 */

void lprod_func_add_arg(Expr* arg)
{
    if (funccuridx == MAXFUNCARGS)
        error("More function arguments than maximum allowed");

    funcargs[funccuridx++] = arg;

    DebugLang(">> lprod_func_add_arg "<<*arg<<"\n");
}

/*

```

```

* Enter a generic function.
*/

void lprod_enter_function(char* name)
{
    if (function_lookup_no_lib(name))
        error("Multiply defined function %s", name);

    function_start(name);

    DebugLang(">> lprod_enter_function "<<name<<"\n");
}

/*
* Add a formal argument to the current function.
*/

void lprod_add_formalarg(char* name, int size)
{
    extern Function *current_function;
    FormalArg arg;

    DebugLang(">> lprod_add_formalarg "<<name<<"<"<<size<<">\n");

    arg.name = name;
    arg.size = size;
    current_function->formals->add(&arg);
}

/*
* Enter the main function.
*/

void lprod_enter_main()
{
    DebugLang(">> lprod_enter_main\n");

    function_start("__main__");
}

/*
* Exit a generic function.
*/

void lprod_exit_function()
{
    extern Function* current_function;

    DebugLang(">> lprod_exit_function\n");
    DebugLang(*current_function);
}

```

```

    /* Don't do anything for now... */
}

/*
 * Exit the main function.
 */

void lprod_exit_main()
{
    extern Function* current_function;

    DebugLang(">> lprod_exit_main\n");
    DebugLang(*current_function);

    lex_close();

    current_dag = flow_simulate();
    if (get_param_int("show_nodes", 0)) {
        cout << "Node list:\n";
        cout << *current_dag << "\n";
    }
}

/*
 * Begin a position list for INPUT/OUTPUT statements
 */

void lprod_positionlist_start()
{
    DebugLang(">> lprod_positionlist_start\n");

    positionlist = new MemBlock(sizeof(int));
}

/*
 * Add a position to the position list
 */

void lprod_positionlist_add(int pos)
{
    DebugLang(">> lprod_positionlist_add "<<pos<<"\n");

    positionlist->add(&pos);
}

/*
 * Begin a varbits list
 */

void lprod_varbitslist_start()

```

```

{
    DebugLang(">> lprod_varbitslist_start\n");

    varbitslist = new MemBlock(sizeof(VarBits*));
}

/*
 * Add a VarBits to the current varbits list
 */

void lprod_varbitslist_add(VarBits* bits)
{
    VarBits** newbits;

    DebugLang(">> lprod_varbitslist_add "<<*bits<<"\n");

    varbitslist->add(&bits);
}

/*
 * Return the current varbitslist
 */

MemBlock* lprod_varbitslist_return()
{
    DebugLang(">> lprod_varbitslist_return\n");

    return varbitslist;
}

/*
 * Add an assignment to the current function. The result of the
 * expression is assigned to the given VarBits's in order.
 */

void lprod_add_assignment(MemBlock* bitslist, Expr* expr)
{
    Code* c;
    char* newarg;
    VarBits** bits;

    c = new Code(CODE_ASSIGN);
    c->addarg((int)expr);
    memblock_iter(bitslist, bits)
        c->addarg((int)*bits);
    delete bitslist;
    function_add(c);

    DebugLang(">> lprod_add_assignment "<<*c<<"\n");
}

```

```

/*
 * Add a declaration to the current function.
 */

void lprod_add_decl(char* name, int size)
{
    Code* c;
    char** tmp;

    c = new Code(CODE_DECL);
    c→addarg((int)name);
    c→addarg(size);
    function_add(c);

    DebugLang(">> lprod_add_decl "<<*c<<"\n");
}

/*
 * Add an INPUT to the main function.
 */

void lprod_add_input(char* name, int size, int position)
{
    extern Function* current_function;
    Code* c;
    int i;

    if (strcmp(current_function→name, "__main__"))
        error("INPUT may only be specified in the main body");

    c = new Code(CODE_INPUT);
    c→addarg((int)name);
    c→addarg(size);
    for (i=0; i<size; i++)
        c→addarg(i+position);
    function_add(c);

    DebugLang(">> lprod_add_input "<<*c<<"\n");
}

/*
 * Add an INPUT to the main function with an explicit position list.
 */

void lprod_add_input_with_list(char* name, int size)
{
    extern Function* current_function;
    Code* c;
    int* pos;

```

```

    if (strcmp(current_function→name, "__main__"))
        error("INPUT may only be specified in the main body");

    c = new Code(CODE_INPUT);
    c→addarg((int)name);
    c→addarg(size);
    if (size ≠ positionlist→size())
        error("Number of bits declared does not agree with positions specified");
    memblock_iter(positionlist, pos)
        c→addarg(*pos);
    function_add(c);

    DebugLang(">> lprod_add_input_with_list "<<*c<<"\n");
}

/*
 * Add an OUTPUT to the main function.
 */

void lprod_add_output(char* name, int size, int position)
{
    extern Function* current_function;
    Code* c;
    int i;

    if (strcmp(current_function→name, "__main__"))
        error("OUTPUT may only be specified in the main body");

    c = new Code(CODE_OUTPUT);
    c→addarg((int)name);
    c→addarg(size);
    for (i=0; i<size; i++)
        c→addarg(i+position);
    function_add(c);

    DebugLang(">> lprod_add_output "<<*c<<"\n");
}

/*
 * Add an OUTPUT to the main function with an explicit position list.
 */

void lprod_add_output_with_list(char* name, int size)
{
    extern Function* current_function;
    Code* c;
    char** tmp;
    int* pos;

    if (strcmp(current_function→name, "__main__"))
        error("OUTPUT may only be specified in the main body");

```

```

    c = new Code(CODE_OUTPUT);
    c→addarg((int)name);
    c→addarg(size);
    if (size ≠ positionlist→size())
        error("Number of bits declared does not agree with positions specified");
    memblock_iter(positionlist, pos)
        c→addarg(*pos);
    function_add(c);

    DebugLang(">> lprod_add_output_with_list "<<*c<<"\n");
}

/*
 * Add a RETURN statement (with the current varbits list as args) to
 * the code.
 */

void lprod_add_return(MemBlock* vb)
{
    Code* c;

    c = new Code(CODE_RETURN);
    c→type = CODE_RETURN;
    /* A little sleight of hand... */
    delete c→args;
    c→args = vb;
    function_add(c);

    DebugLang(">> lprod_add_return "<<*c<<"\n");
}

/*
 * Add a TYPEOUT statement to the code.
 */

void lprod_add_typeout(char* s)
{
    Code* c;

    c = new Code(CODE_TYPEOUT);
    c→type = CODE_TYPEOUT;
    c→args→add(&s);
    function_add(c);

    DebugLang(">> lprod_add_typeout \"\"<<s<<\"\"\n");
}

/*
 * Do a screen ECHO. This does not actually change the code at all,
 * but is merely used to display messages during parsing.

```



```
*/  
  
void lprod_do_echo(char* s)  
{  
    cout << "==" << s << "\n";  
}
```

## File lex.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _lex_h_
#define _lex_h_

class ostream;

int lex_init(char*);
void lex_close(), lex_printpos(ostream&);
void lex_set_file_line(char*, int);
void lex_select_lang(), lex_select_arch();

char* lex_get_filename();
int lex_current_line();

int yyarchlex();
int yylanglex();

#endif /* _lex_h_ */
```

## File lex.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <Obstack.h>
#include <ctype.h>
#include <stream.h>
#include "archy.tab.h"
#include "front.h"
#include "lex.h"
#include "langy.tab.h"
#include "util.h"

/*
 * These are local variables used to keep lex state.
 */

static istream* infile = 0;
static char filename[256];
static int lineno = -1;
static Obstack* symbols = 0;

/*
 * Initialize lex processing for a particular file.
 */

int lex_init(char* file)
{
  filename[0] = '\0';
  delete infile;
  infile = new istream(file, io_readonly, a_useonly);
  if (!infile->is_open()) {
    /* cpp will have printed an error message */
    delete infile;
    infile = 0;
    return 1;
  }
  delete symbols;
  symbols = new Obstack();
  return 0;
}

/*
 * Get the next character from the input file and return it. Handle
```

```

* line#/filename notations as provided by cpp. Return -1 on error.
*/

static int mygetchar()
{
    char curchar;
    static int prevchar = '\n';
    char smallbfr[256];

    for (;;) {
        infile→get(curchar);
        if (!infile→good())
            return -1;
        if (curchar == '\n')
            lineno++;
        if (prevchar == '\n' && curchar == '#') {
            if (!infile→getline(smallbfr, sizeof smallbfr))
                return -1;
            sscanf(smallbfr, "%d %s", &lineno, filename);
            continue;
        }
        return curchar;
    }
}

/*
* Close off the lex input file.
*/

void lex_close()
{
    if (infile) {
        infile→close();
        delete infile;
    }
    infile = 0;
    lineno = -1;
}

/*
* Print the file/line# location, if possible, for error reports.
*/

void lex_printpos(ostream& s)
{
    if (infile)
        s.form("%s, line %d: ", filename, lineno);
    else if (lineno ≠ -1)
        s.form("%s, line %d: ", filename, lineno);
}

```

```

/*
 * Set the current file/line# location to fake out future error
 * reports.
 */

void lex_set_file_line(char* s, int n)
{
    strcpy(filename, s);
    lineno = n;
}

/*
 * Return the current filename as reported by cpp.
 */

char* lex_get_filename()
{
    return filename;
}

/*
 * Return the current line# as reported by cpp.
 */

int lex_current_line()
{
    return lineno;
}

/*
 * This function should only be called by yyparse(). It returns the
 * next token on the input stream.
 */

static struct _Reserved {
    char *word;
    int value;
} *current_reserved_words,
lang_reserved_words[] = {
    "DECL", L_DECL,
    "INPUT", L_INPUT,
    "OUTPUT", L_OUTPUT,
    "RETURN", L_RETURN,
    "TYPEOUT", L_TYPEOUT,
    "ECHO", L_ECHO,
    0, 0
};
arch_reserved_words[] = {
    "DIMENS", A_DIMENS,
    "NODEDEF", A_NODEDEF,
    "INPUTS", A_INPUTS,

```

```

    "OUTPUTS", A_OUTPUTS,
    "FLAVOR", A_FLAVOR,
    "KERNEL", A_KERNEL,
    "TYPE", A_TYPE,
    "COORD", A_COORD,
    "KERN", A_KERN,
    "NODE", A_NODE,
    0, 0
};

void lex_select_lang()
{
    current_reserved_words = lang_reserved_words;
}

void lex_select_arch()
{
    current_reserved_words = arch_reserved_words;
}

static YYLANGSTYPE yylval;

yylex()
{
    int c;

    /* Skip over whitespace */
    for (;;) {
        if ((c = mygetchar()) == -1)
            return -1;
        if (isspace(c) || c == '\n')
            continue;
        break;
    }

    /* Is it going to be an INTEGER? */
    if (isdigit(c) || c == '-') {
        symbols→grow(c);
        for (;;) {
            c = mygetchar();
            if (!isdigit(c))
                break;
            symbols→grow(c);
        }
        char* ptr = symbols→finish(0);
        if (c ≠ -1)
            infile→unget(c);
        yylval.n = atoi(ptr);
        symbols→free(ptr);
        return L_INTEGER;
    }
}

```

```

/* Is it going to be a quoted string? */
if (c == '\\"') {
    for (;;) {
        c = mygetchar();
        if (c == '\\"' || c == -1)
            break;
        symbols→grow(c);
    }
    ylval.s = symbols→finish(0);
    return L_STRING;
}

/* How about an identifier or reserved word? */
if (isalpha(c)) {
    symbols→grow(c);
    for (;;) {
        c = mygetchar();
        if ((!isalnum(c) && c ≠ '_' ) || c == -1)
            break;
        symbols→grow(c);
    }
    ylval.s = symbols→finish(0);
    infile→unget(c);
    for (int i=0; current_reserved_words[i].word; i++)
        if (!strcasecmp(ylval.s, current_reserved_words[i].word)) {
            symbols→free(ylval.s);
            return current_reserved_words[i].value;
        }
    return L_ID;
}

/* Just a random character */
return c;
}

int yylanglex()
{
    int ret = yylex();
    ylval = ylval;
    return ret;
}

int yyarchlex()
{
    int ret = yylex();
    yyarchlval = *((YYARCHSTYPE*)&ylval);
    return ret;
}

```

## File lib.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _lib_h_
#define _lib_h_

class Array;
class intMB;
class istream;
class ostream;

struct LibFile {
    char* name;
    istream* fp;
};

class LibEntry {
public:
    LibFile* libfile;
    long pos;
    char* name;
    intMB* inputs;
    intMB* outputs;
    intMB* sizes;
    Array* array;
public:
    friend ostream& operator << (ostream&, LibEntry&);
};

void lib_init();
void lib_add_file(char*, istream*);
int lib_read_def(LibFile*);
void lib_bad_file();
int lib_convert_intstr(char*, intMB*, char*);
int lib_read_line(istream*, char*);
LibEntry* lib_lookup(char*);
int lib_lookup_alt_num(char*);
LibEntry* lib_lookup_alt(char*, int);
int lib_is_loaded(char*);
void lib_dump_files(ostream&);
void lib_dump(ostream&);
int lib_compute_output_bitpos(LibEntry*, int);
int lib_compute_input_bitpos(LibEntry*, int);
```



```
#endif /* _lib_h_ */
```

## File lib.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stdio.h>
#include <stream.h>
#include <ctype.h>
#include "Array.h"
#include "MemBlock.h"
#include "config.h"
#include "front.h"
#include "intMB.h"
#include "lib.h"
#include "util.h"

/*
 * Consists of LibFile's, not LibFile *'s
 */
static MemBlock* libfiles;

/*
 * Consists of LibEntry's, not LibEntry *'s
 */
static MemBlock* libroutines;

/*
 * Initialize library handling routines
 */

void lib_init()
{
    libfiles = new MemBlock(sizeof(LibFile));
    libroutines = new MemBlock(sizeof(LibEntry));
}

/*
 * Add a file to the library search list.
 */

void lib_add_file(char* name, istream* file)
{
    LibFile newfile;
    newfile.name = xsave_string(name);
    newfile.fp = file;
}
```

```

libfiles→add(&newfile);
LibFile* realentry = memblock_get_datap(libfiles, LibFile,
                                       libfiles→size()-1);
while (lib_read_def(realentry))
    ;
cout << "\n";
}

/*
 * Read a single routine definition from the specified library -
 * return zero if end-of-file is found.
 */

int lib_read_def(LibFile* lib)
{
    char funcname[STRSIZE], bfr[STRSIZE];

    if (!lib_read_line(lib→fp, funcname))
        return 0;

    LibEntry newentry;
    newentry.libfile = lib;
    newentry.name = xsave_string(funcname);
    LibEntry* oldentry;
    int foundone = 0;
    memblock_iter(libroutines, oldentry)
        if (!strcmp(funcname, oldentry→name)) {
            foundone = 1;
            break;
        }
    if (!foundone)
        cout << funcname << "...";

    if (!lib_read_line(lib→fp, bfr))
        lib_bad_file();

    newentry.inputs = new intMB();
    lib_convert_intstr(bfr, newentry.inputs, "INPUTS");

    if (!lib_read_line(lib→fp, bfr))
        lib_bad_file();

    newentry.outputs = new intMB();
    lib_convert_intstr(bfr, newentry.outputs, "OUTPUTS");

    if (!lib_read_line(lib→fp, bfr))
        lib_bad_file();

    newentry.sizes = new intMB();
    lib_convert_intstr(bfr, newentry.sizes, "SIZE");
}

```

```

newentry.array = new Array(2);
if (!newentry.array→read_ascii(lib→fp, (*newentry.sizes)[0],
                               (*newentry.sizes)[1]))
    lib_bad_file();

libroutines→add(&newentry);

return 1;
}

/*
 * Illegal library format...
 */

void lib_bad_file()
{
    error("Illegal library format!"); /* XXX More descriptive... */
}

/*
 * Convert a string of the form "KEYWORD 0 1 2 3" to a MemBlock representation
 */

int lib_convert_intstr(char* bfr, intMB* mb, char* keyword)
{
    char* ptr = bfr;
    if (strncasecmp(bfr, keyword, strlen(keyword)))
        error("Bad library entry - keyword %s not found", keyword);
    ptr += strlen(keyword)+1;

    while (*ptr) {
        mb→add(atoi(ptr));
        while (*ptr && !isspace(*ptr))
            ptr++;
        while (*ptr && isspace(*ptr))
            ptr++;
    }
}

/*
 * Read a line from the specified istream, ignoring blank lines and
 * lines that start with '#'. Return 0 on end-of-file. Assume the
 * buffer is STRSIZE bytes wide. Also strip the trailing newline off,
 * as well as leading and trailing spaces.
 */

int lib_read_line(istream* fp, char* s)
{
    for (;;) {
        if (!fp→good())
            return 0;
    }
}

```

```

        if (!fp→getline(s, STRSIZE))
            return 0;
        if (s[strlen(s)-1] == '\n')
            s[strlen(s)-1] = '\0';
        while (isspace(*s))
            strcpy(s, s+1);
        while (isspace(s[strlen(s)-1]))
            s[strlen(s)-1] = '\0';
        if (*s && s[0] ≠ '#')
            return 1;
    }
}

/*
 * Lookup a library routine
 */

LibEntry* lib_lookup(char* name)
{
    LibEntry* entry;

    memblock_iter(libroutines, entry) {
        if (!strcmp(entry→name, name))
            return entry;
    }
    return NULL;
}

/*
 * Find alternate library routines with the same name
 */

int lib_lookup_alt_num(char* name)
{
    int num = 0;
    LibEntry* entry;

    memblock_iter(libroutines, entry) {
        if (!strcmp(entry→name, name))
            num++;
    }
    return num;
}

/*
 * Lookup an alternate library routine
 */

LibEntry* lib_lookup_alt(char* name, int num)
{
    LibEntry* entry;

```

```

    memblock_iter(libroutines, entry) {
        if (!strcmp(entry->name, name))
            if (!num--)
                return entry;
    }
    return NULL;
}

/*
 * Find the position of an input bit
 */

lib_compute_input_bitpos(LibEntry* entry, int bit)
{
    return (*entry->inputs)[bit];
}

/*
 * Find the position of an output bit
 */

int lib_compute_output_bitpos(LibEntry* entry, int bit)
{
    return (*entry->outputs)[bit];
}

/*
 * Is a library already loaded?
 */

int lib_is_loaded(char* path)
{
    LibFile* lib;

    memblock_iter(libfiles, lib)
        if (!strcmp(lib->name, path))
            return 1;
    return 0;
}

/*
 * Dump information about all of the library files to a stream.
 */

void lib_dump_files(ostream& s)
{
    LibFile* lib;

    if (!libfiles->size()) {
        cout << "No libraries currently loaded\n";
    }
}

```

```

        return;
    }
    cout << "Currently loaded libraries:\n";
    memblock_iter(libfiles, lib)
        s << lib->name << "\n";
}

/*
 * Dump all library routine entries to a stream.
 */

void lib_dump(ostream& s)
{
    LibEntry* entry;

    memblock_iter(libroutines, entry)
        s << *entry;
}

/*
 * Print a library routine entry
 */

ostream& operator << (ostream& s, LibEntry& entry)
{
    return s << entry.libfile->name << " @ " << entry.pos << ": " <<
        entry.name << "\n";
}

```

## File MemBlock.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _memblock_h_
#define _memblock_h_

#define MEMBLOCK_ALLOCSIZE 16

typedef int (*PFRI)(void*, void*);

class MemBlock {
public:
    int lowbound;
    int num;
    int allocated;
    int bsize;
    char* data;
public:
    MemBlock(int size, int lowb=0);
    ~MemBlock();
    size() { return num; }
    next() { return lowbound+num; }
    low() { return lowbound; }
    get_allocated() { return allocated; }
    get_bsize() { return bsize; }
    void* get_data() { return data; }
    add(void*);
    add_low(void*);
    del(void*);
    del(int);
    reinit();
    sort(PFRI);
    sort_range(PFRI, int, int);
    MemBlock* copy();
    void* last() { return data+bsize*(num-lowbound-1); }
    void* operator [] (int n) { return data+bsize*(n-lowbound); }
};

#define memblock_get_datap(m,d,n) (((d *)((m)→data))+n)-(m)→low()
#define memblock_get_data(m,d,n) (((d *) (m)→data)[n)-(m)→low()])

#define memblock_iter(m,v) \
    for(v=(typeof(v))(m)→data;v≠(typeof(v))(m)→data+(m)→num;v++)
```



```
#endif /* _memblock_h_ */
```

## File MemBlock.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stdio.h>
#include <std.h>
#include "MemBlock.h"
#include "util.h"

MemBlock::MemBlock(int size, int lowb)
{
    num = 0;
    allocated = 0;
    bsize = size;
    lowbound = lowb;
    data = (char *)NULL;
}

MemBlock::~MemBlock()
{
    if (allocated)
        xfree(data);
}

MemBlock::add(void* newdata)
{
    if (num == allocated) {
        allocated += MEMBLOCK_ALLOCSIZE;
        if (num)
            data = (char *)xrealloc(data, bsize*allocated);
        else
            data = (char *)xmalloc(bsize*allocated);
    }
    bcopy(newdata, data+num*bsize, bsize);
    num++;
}

MemBlock::add_low(void* newdata)
{
    lowbound--;
    if (num == allocated) {
        allocated += MEMBLOCK_ALLOCSIZE;
        if (num)
            data = (char *)xrealloc(data, bsize*allocated);
    }
}
```

```

        else
            data = (char *)xmalloc(bsize*allocated);
    }
    bcopy(data, data+bsize, num*bsize);
    bcopy(newdata, data, bsize);
    num++;
}

MemBlock::del(void* ptr)
{
    bcopy(ptr+bsize, ptr, data-(char *)ptr+num*bsize);
    num--;
}

MemBlock::del(int n)
{
    n -= lowbound;
    bcopy(data+bsize*(n+1), data+bsize*n, (num-n-1)*bsize);
    num--;
}

MemBlock::reinit()
{
    if (allocated)
        xfree(data);
    num = 0;
    allocated = 0;
    lowbound = 0;
    data = (char *)NULL;
}

MemBlock* MemBlock::copy()
{
    register MemBlock* ret = new MemBlock(bsize);
    int i;

    ret->bsize = bsize;
    ret->allocated = allocated;
    ret->num = num;
    ret->lowbound = lowbound;
    ret->data = (char*)xmalloc(bsize*allocated);
    bcopy(data, ret->data, bsize*num);

    return ret;
}

MemBlock::sort(PFRI compfunc)
{
    qsort((void*)data, num, (unsigned)bsize, compfunc);
}

```

```
MemBlock::sort_range(PFRI compfunc, int start, int end)
{
    qsort((void*)(data+start*bsize), end-start+1, (unsigned)bsize, compfunc);
}
```

## File Node.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _node_h_
#define _node_h_

#include "intMB.h"
#include "lib.h"

class ostream;
struct Signal;

enum nodetype { NODE_INPUT, NODE_OUTPUT, NODE_BLOCK };

class Node {
private:
    int id;
    MemBlock* signals;
    MemBlock* backsignals;
public:
    char* name;
    nodetype type;
    LibEntry* lib;
    int vertmin, vertmax, curvert;
    int flag;
    int idx;
    intMB* pos;
    intMB* idealpos;
    intMB* reqright;
    intMB* reqleft;

    Node(nodetype, char*);
    Node(nodetype, LibEntry*);
    Node(Node&);
    ~Node();
    add_signal(Signal*);
    add_backsignal(Signal*);
    MemBlock* get_signals() { return signals; }
    MemBlock* get_backsignals() { return backsignals; }
    int width() { return type == NODE_BLOCK ?
        (*lib->sizes)[0]*2 + ((*lib->sizes)[1] > 1 ? 1 : 0) : 0; }
    int get_hpos() { return (*pos)[0]; /*return type == NODE_BLOCK ?
        (*pos)[0] + ((*lib->sizes)[1] < 2 ? 1 : 0) :
```

```

        (*pos)[0];*/ }
char* get_name();
friend ostream& operator << (ostream&, Node&);
};

struct Signal {
    int srcpos;
    int destpos;
    int src;
    int dest;
    intMB* goals;
    intMB* goalthpos;
    intMB* routegoal;
};

class RouteWire {
public:
    int hpos;
    int startvpos;
    int starthpos; /* starting hpos at node with higher level */
    int desthpos; /* ending hpos at node with lower level */
    int marked;
    int inactive;
    Node* srcnode; /* node with higher level */
    Node* destnode; /* node with lower level */
    Signal* signal;

    friend ostream& operator << (ostream&, RouteWire&);
};

inline GOAL(RouteWire* w, int l)
{
    if (l == w->destnode->curvert)
        return w->desthpos;
    if (l == w->srcnode->curvert)
        return w->starthpos;
    return (*w->signal->routegoal)[l-w->destnode->curvert-1];
}

#endif /* _node_h_ */

```

## File Node.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "Arch.h"
#include "MemBlock.h"
#include "Node.h"
#include "front.h"
#include "intMB.h"
#include "lib.h"

/*
 * Variable used to give a unique number to every new Node that's created.
 */

static int node_number = 0;

/*
 * Initialize a new Node.
 */

Node::Node(nodetype thetype, char* thename)
{
    type = thetype;
    signals = new MemBlock(sizeof(Signal));
    backsignals = new MemBlock(sizeof(Signal*));
    name = thename;
    lib = 0;
    id = node_number++;
    vertmin = vertmax = curvert = -1;
    flag = 0;
    pos = new intMB();
    for (int i=0; i<current_arch->dimens; i++)
        pos->add(0);
    idealpos = pos->copy();
    reqright = pos->copy();
    reqleft = pos->copy();
}

Node::Node(nodetype thetype, LibEntry* entry)
{
    type = thetype;
    signals = new MemBlock(sizeof(Signal));
```

```

    backsignals = new MemBlock(sizeof(Signal*));
    name = entry->name;
    lib = entry;
    id = node_number++;
    vertmin = vertmax = curvert = -1;
    flag = 0;
    pos = new intMB();
    for (int i=0; i<current_arch->dimens; i++)
        pos->add(0);
    idealpos = pos->copy();
    reqright = pos->copy();
    reqleft = pos->copy();
}

/*
 * Copy data from an existing Node.
 */

Node::Node(Node& oldnode)
{
    type = oldnode.type;
    name = oldnode.name;
    lib = oldnode.lib;
    id = oldnode.id;
    idx = oldnode.idx;
    vertmin = oldnode.vertmin;
    vertmax = oldnode.vertmax;
    curvert = oldnode.curvert;
    flag = 0;
    signals = oldnode.signals->copy();
    backsignals = new MemBlock(sizeof(Signal*));
    pos = oldnode.pos->copy();
    idealpos = oldnode.idealpos->copy();
    reqright = oldnode.reqright->copy();
    reqleft = oldnode.reqleft->copy();
}

/*
 * Destroy a Node.
 */

Node::~Node()
{
    Signal* signal;
    memblock_iter(signals, signal) {
        if (signal->goals)
            delete signal->goals;
        if (signal->goalhpos)
            delete signal->goalhpos;
        if (signal->routegoal)
            delete signal->routegoal;
    }
}

```



```

    }
    delete signals;
    delete backsignals;
    delete pos;
    delete idealpos;
    delete reqright;
    delete reqleft;
}

/*
 * Add a signal to a Node.
 */

Node::add_signal(Signal* signal)
{
    signals→add(signal);
}

/*
 * Add a backwards link to a Node.
 */

Node::add_backsignal(Signal* signal)
{
    backsignals→add(&signal);
}

/*
 * Dump a Node to a stream.
 */

static void node_dump_intmb(ostream& s, intMB* mb)
{
    s << "(";
    for (int i=0; i<current_arch→dimens; i++) {
        s << (*mb)[i];
        if (i ≠ current_arch→dimens-1)
            s << ",";
    }
    s << ")";
}

ostream& operator << (ostream& s, Node& node)
{
    s << "<Node " << node.idx << " " << node.get_name() << " (";
    switch (node.type) {
    case NODE_INPUT:
        s << "in)";
        break;
    case NODE_OUTPUT:
        s << "out)";
    }
}

```

```

        break;
    case NODE_BLOCK:
        s << "blk";
        break;
    }
    if (node.vertmin ≠ -1)
        s.form(" [%d:%d]", node.vertmin, node.vertmax);
    if (node.curvert ≠ -1) {
        s.form("0%d ipos (" , node.curvert);
        s << *node.idealpos << " ) pos (" << *node.pos << " ) reqleft (" <<
            *node.reqleft << ")\n\t\treqright (" << *node.reqright <<
                " )";
    }
    return s << ">";
}

/*
 * Return the name of a Node.
 */

char* Node::get_name()
{
    static char bfr[256];

    sprintf(bfr, "%s_%d", name, id);
    return bfr;
}

```

## File NodeDefMB.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _NodeDefMB_h_
#define _NodeDefMB_h_

#include "MemBlock.h"
#include "Arch.h"

class NodeDefMB : public MemBlock {
public:
    NodeDefMB() : (sizeof(NodeDef)) {};
    NodeDef** get_data() { return (NodeDef**)data; }
    NodeDefMB* copy() { return (NodeDefMB*)((MemBlock*)this->copy()); }
    NodeDef* last() { return (NodeDef*)(data+bsize*(num-lowbound-1)); }
    NodeDef* operator [] (int n) { return (NodeDef*)(data+bsize*(n-lowbound)); }
};

#endif /* _NodeDefMB_h_ */
```

## File pr2dblock.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _pr2dblock_h_
#define _pr2dblock_h_

class Array;
class DAG;
class Flavor;
class MemBlock;
class Node;

#define EVEN(x) (((x)&1)==0)
#define ODD(x) (((x)&1)≠0)

const INC=2;

extern DAG* pr2ddag;
extern int pr2dblock_min_wire_dist;
extern Array* pr2dblock_array;
extern Flavor* array_crossover, *array_passthru, *array_leftbroadcast,
    *array_rightbroadcast;

int pr2dblock_init(DAG*);
void pr2dblock_do_everything();

void pr2dblock_place_ideal();
void pr2dblock_compute_lrreq();
void pr2dblock_dump_routings();
void pr2dblock_route();
void pr2dblock_do_annealing();
int pr2dblock_get_cost(DAG*);
void pr2dblock_anneal_optimize();

int pr2dblock_output_bitpos(Node*, int);
int pr2dblock_input_bitpos(Node*, int);

void pr2dblock_display();

#ifdef IV
const pr2dblock_xsize = 1000;
const pr2dblock_ysize = 700;
const pr2dblock_xscale = 50;
```

```
const pr2dblock_yscale = 50;  
#endif  
  
#endif /* _pr2dblock_h_ */
```

## File pr2dblock.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "Arch.h"
#include "Array.h"
#include "DAG.h"
#include "MemBlock.h"
#include "Node.h"
#include "SPlotFile.h"
#include "archprods.h"
#include "config.h"
#include "front.h"
#include "intMB.h"
#include "pr2dblock.h"
#include "util.h"

MemBlock *routinglist;      /* Routing's */

DAG* pr2ddag;
Array* pr2dblock_array;

/*
 * User-settable minimum dist btwn wires
 */
int pr2dblock_min_wire_dist;

/*
 * Initialize any data structures...called before any other block routine
 */

Flavor* array_crossover, *array_passthru, *array_leftbroadcast,
      *array_rightbroadcast;

int pr2dblock_init(DAG* dag)
{
    DebugPR2D("pr2dblock_init\n");

    array_crossover = current_arch->lookup_flavor("XOVER");
    if (!array_crossover)
        error("No XOVER flavor in the current architecture");
    array_passthru = current_arch->lookup_flavor("PT");
    if (!array_passthru)
```

```

        error("No PT flavor in the current architecture");
array_leftbroadcast = current_arch→lookup_flavor("LBRD");
if (!array_leftbroadcast)
    error("No LBRD flavor in the current architecture");
array_rightbroadcast = current_arch→lookup_flavor("RBRD");
if (!array_rightbroadcast)
    error("No RBRD flavor in the current architecture");

pr2dblock_min_wire_dist = get_param_int("pr2dblock_wire_dist", 1);
if (pr2dblock_min_wire_dist < 0 || pr2dblock_min_wire_dist > 30) {
    warning("pr2dblock_wire_dist must be between 0 and 30 inclusive");
    return 1;
}
if (current_arch→dimens ≠ 2) {
    warning("Architecture must be two dimensional");
    return 1;
}
pr2ddag = dag;
pr2dblock_array = NULL;
return 0;
}

/*
 * Perform all of the block operations:
 * pr2dblock_place_ideal (pr2dblockplace.c):
 *     Place in ideal horizontal configuration
 * pr2dblock_compute_lrreq (pr2dblockplace.c):
 *     Compute routing spaces and displace blocks horizontally
 * pr2dblock_route (pr2dblockroute.c)
 *     Do the actual routing
 * pr2dblock_make_array (this file)
 *     Dump resultant array (including inputs and outputs)
 */

void pr2dblock_do_everything()
{
    DebugPR2D("pr2dblock_do_everything\n");

    char *optim = get_param("optimization", "");

    if (!strcmp(optim, "sa")) {
        pr2dblock_do_annealing();
        return;
    }

    /* No optimization! */
    pr2ddag→verify();
    pr2dblock_place_ideal();
    if (get_param_int("pr2d_show_ideal", 0))
        cout << "New hierarchy:\n" << *pr2ddag << "\n";
    pr2ddag→verify();
}

```

```

    pr2dblock_compute_lrreq();
#ifdef DEBUGPR2D
    if (get_param_int("pr2d_show_nodes", 0))
#endif
        cout << "New hierarchy:\n" << *pr2ddag;
#ifdef DEBUGPR2D
    if (get_param_int("pr2d_show_routings", 0))
#endif
        pr2dblock_dump_routings();
#ifdef IV
    if (get_param_int("pr2d_display", 0))
        pr2dblock_display();
#endif

    pr2ddag->verify();
    pr2dblock_array = new Array(2);
    pr2dblock_route();
    pr2ddag->verify();
    if (get_param_int("show_array", 0))
        cout << *pr2dblock_array;
}

/*
 * Simulated annealing support
 */

void pr2dblock_do_annealing()
{
    int cost, lastcost, firstcost;
    DAG *newdag, *olddag;

    int firstpass = 1;
    int totaltried;

    DebugPR2D("*** Doing Simulated Annealing!\n");

    float temp = get_param_float("sa_start_temp", 10.0);
    int numiters = get_param_int("sa_num_iters", 100);
    float tempmult = get_param_float("sa_temp_scale", .95);

    char* plotfilename = get_param("sa_plotfile", NULL);
    SPlotFile *plotfile = NULL;
    if (plotfilename) {
        char bfr[256];
        plotfile = new SPlotFile(plotfilename, io_writeonly, a_create);
        sprintf(bfr,
            "Simulated Annealing: Start temp %f, Iters %d, Mult %f",
            temp, numiters, tempmult);
        plotfile->init(7, 9.5, bfr);
        plotfile->new_curve();
        plotfile->curve_point_start();
    }
}

```



```

}

pr2dblock_place_ideal();
if (get_param_int("pr2d_show_ideal", 0))
    cout << "New hierarchy:\n" << *pr2ddag << "\n";
for (totaltried=1;totaltried++) {
    if ((totaltried%100)==0 && !verbose)
        cout << "Temp " << temp << " cost " << lastcost << "\n";
    pr2dblock_compute_lrrreq();
#ifdef DEBUGPR2D
    if (get_param_int("pr2d_show_nodes", 0))
#endif
        cout << "New hierarchy:\n" << *pr2ddag;

#ifdef DEBUGPR2D
    if (get_param_int("pr2d_show_routings", 0))
#endif
        pr2dblock_dump_routings();
#ifdef IV
    if (get_param_int("pr2d_display", 0))
        pr2dblock_display();
#endif

    if (pr2dblock_array) {
        delete pr2dblock_array;
        pr2dblock_array = NULL;
    }
    if (!strcmp(get_param("sa_cost", "size"), "size")) {
        pr2dblock_array = new Array(2);
        pr2dblock_route();
        if (get_param_int("show_array", 0))
            cout << *pr2dblock_array;
    }

    cost = pr2dblock_get_cost(pr2ddag);
    if (verbose)
        cout << "Temp " << temp << ", new cost " << cost << "...";
    if (!firstpass) {
        if (cost-lastcost <= 0 ||
            (cost-lastcost > 0 &&
             random()/2147483647.0 < exp((lastcost-cost)/temp))) {
            if (verbose)
                cout << "Accepting!\n";
            lastcost = cost;
            delete olddag;
        }
        else {
            if (verbose)
                cout << "Rejecting!\n";
            delete pr2ddag;
            pr2ddag = olddag;
        }
    }
}

```

```

    }
}
else {
    lastcost = cost;
    firstcost = cost;
}

if (plotfile) {
    plotfile→curve_point(totaltried, lastcost);
    if ((totaltried%100) == 0) {
        plotfile→curve_point_end();
        plotfile→new_curve();
        plotfile→curve_point_start();
        plotfile→curve_point(totaltried, lastcost);
    }
}

firstpass = 0;
olddag = pr2ddag;

temp *= tempmult;
if (totaltried == numiters)
    break;

pr2ddag = pr2ddag→copy();
pr2dblock_anneal_optimize();
}
if (plotfile) {
    plotfile→curve_point_end();
    plotfile→xaxis("Trial number");
    plotfile→yaxis("Cost");
    delete plotfile;
}
if (pr2dblock_array) {
    delete pr2dblock_array;
    pr2dblock_array = NULL;
}
pr2dblock_array = new Array(2);
pr2dblock_route();
if (get_param_int("show_array", 0))
    cout << *pr2dblock_array;
cout << "SA Optimization: " << totaltried <<
    " tried, initial cost " << firstcost << ", final cost " <<
    lastcost << "\n";
int xmin, xmax, ymin, ymax;
pr2dblock_array→getdims(&xmin, &xmax, &ymin, &ymax);
cout << "    Final array size: " << xmax-xmin << " by " <<
    ymax-ymin << " = " << (xmax-xmin)*(ymax-ymin) << "\n";
}

int pr2dblock_get_goal(Signal* signal, int lev)

```

```

{
    if (lev == NODEREF(pr2ddag→nodelist, signal→src)→curvert)
        return pr2dblock_output_bitpos(NODEREF(pr2ddag→nodelist,
            signal→src),
            signal→srcpos);
    if (lev == NODEREF(pr2ddag→nodelist, signal→dest)→curvert)
        return pr2dblock_input_bitpos(NODEREF(pr2ddag→nodelist,
            signal→dest),
            signal→destpos);
    return (*signal→routegoal)[lev-NODEREF(pr2ddag→nodelist,
        signal→src)→curvert-1];
}

int pr2dblock_get_cost(DAG* dag)
{
    int xmin, ymin, xmax, ymax;

    char* costfunc = get_param("sa_cost", "size");
    if (!strcmp(costfunc, "size")) { /* Total array size */
        pr2dblock_array→getdims(&xmin, &xmax, &ymin, &ymax);
        return (xmax-xmin)*(ymax-ymin);
    }
    if (!strcmp(costfunc, "crosses")) { /* Number of wire crosses */
        int crosses = 0;
        int hordist = 0;
        Node** node1;
        Signal* signal1;
        Node** node2;
        Signal* signal2;
        memblock_iter(pr2ddag→nodelist, node1) {
            memblock_iter((*node1)→get_signals(), signal1) {
                for (int lev=NODEREF(pr2ddag→nodelist, signal1→src)→curvert;
                    lev<NODEREF(pr2ddag→nodelist, signal1→dest)→curvert;
                    lev++) {
                    hordist += abs(pr2dblock_get_goal(signal1, lev)-
                        pr2dblock_get_goal(signal1, lev+1));
                    memblock_iter(pr2ddag→nodelist, node2) {
                        memblock_iter((*node2)→get_signals(), signal2) {
                            if (signal1 == signal2)
                                continue;
                            if (NODEREF(pr2ddag→nodelist, signal2→src)→
                                curvert > lev ||
                                NODEREF(pr2ddag→nodelist, signal2→dest)→
                                curvert ≤ lev)
                                continue;
                            int top1 = pr2dblock_get_goal(signal1, lev);
                            int top2 = pr2dblock_get_goal(signal2, lev);
                            int bot1 = pr2dblock_get_goal(signal1, lev+1);
                            int bot2 = pr2dblock_get_goal(signal2, lev+1);
                            if ((top1 ≤ top2 && bot1 ≥ bot2) ||
                                (top1 ≥ top2 && bot1 ≤ bot2))

```

```

        crosses++;
    }
}
}
}
return hordist+crosses/2;
}
if (!strcmp(costfunc, "hordist")) { /* Number of wire crosses */
    int hordist = 0;
    Node** node1;
    Signal* signal1;
    Node** node2;
    Signal* signal2;
    memblock_iter(pr2ddag->nodelist, node1) {
        memblock_iter((*node1)->get_signals(), signal1) {
            for (int lev=NODEREF(pr2ddag->nodelist, signal1->src)->curvert;
                lev<NODEREF(pr2ddag->nodelist, signal1->dest)->curvert;
                lev++) {
                hordist += abs(pr2dblock_get_goal(signal1, lev)-
                    pr2dblock_get_goal(signal1, lev+1));
            }
        }
    }
    return hordist;
}
error("Illegal cost function: %s", costfunc);
}

void pr2dblock_anneal_optimize()
{
    int level, start, end, mod;
    Node* node;

    for (;;) {
        switch (random() % 3) { /* Pick an optimization */
        case 0: /* Move a module's ideal pos slightly left or right */
            try_again:
                level = (random() % (pr2ddag->maxlevel-1))+1;
                start = pr2ddag->level_start(level);
                end = pr2ddag->level_end(level);
                mod = (random() % (end-start+1))+start;
                node = NODEREF(pr2ddag->nodelist, mod);
                if (verbose)
                    cout << "Picked node " << node->get_name() <<
                        " for moving ideal pos\n";
                int newhpos = (random() % 2)*2-1 + (*node->idealpos)[0];
                if (verbose)
                    cout << "Old pos " << (*node->idealpos)[0] << " new pos " <<
                        newhpos << "\n";
                /*

```

```

    * See if this will conflict with another module on this
    * level
    */
    for (int i=start; i<=end; i++) {
        if (i == mod)
            continue;
        Node* checknode = NODEREF(pr2ddag->nodelist, i);
        if (newhpos+node->width()-1 >= (*checknode->idealpos)[0] &&
            newhpos <= (*checknode->idealpos)[0]+checknode->width()-1)
            break;
    }
    if (i != end+1) {
        if (verbose)
            cout << "Conflict!\n";
        goto try_again;
    }
    (*node->idealpos)[0] = newhpos;
    (*node->pos)[0] = newhpos;
    return;
case 1: /* Transpose two modules */
    level = (random() % (pr2ddag->maxlevel-1))+1;
    start = pr2ddag->level_start(level);
    end = pr2ddag->level_end(level);
    if (start == end) /* Must be at least 2 mods on this level */
        break;
    mod = (random() % (end-start))+start;
    node = NODEREF(pr2ddag->nodelist, mod);
    Node* nextnode = NODEREF(pr2ddag->nodelist, mod+1);
    if (verbose)
        cout << "Picked node " << node->get_name() <<
            " for transposition with " << nextnode->get_name() << "\n";
    int lefthpos = (*node->idealpos)[0];
    int righthpos = (*nextnode->idealpos)[0]+
        (*nextnode->lib->sizes)[0]*2;
    (*node->idealpos)[0] = righthpos-(*node->lib->sizes)[0]*2;
    (*node->pos)[0] = (*node->idealpos)[0];
    (*nextnode->idealpos)[0] = lefthpos;
    (*nextnode->pos)[0] = lefthpos;
    return;
case 2: /* Switch library modules */
try_again2:
    mod = random() % (pr2ddag->nodelist->size());
    node = NODEREF(pr2ddag->nodelist, mod);
    if (node->type != NODE_BLOCK)
        goto try_again2;
    int numavail = lib_lookup_alt_num(node->lib->name);
    if (numavail < 2)
        break;
    int selected = random()%numavail;
    LibEntry* newentry = lib_lookup_alt(node->lib->name, selected);
    if (newentry == node->lib)

```

```

        goto try_again2;
    if (verbose)
        cout << "Picked node " << node->get_name() <<
            " to change module - selected " << selected << "\n";
    node->lib = newentry;
    return;
}
}
}

/*
 * Dump all of the routings
 */

void pr2dblock_dump_routings()
{
    Node** nodep;
    Signal* signal;

    cout << "Routings:\n";

    memblock_iter(pr2ddag->nodelist, nodep) {
        memblock_iter((*nodep)->get_signals(), signal) {
            Node* destnode = NODEREF(pr2ddag->nodelist, signal->dest);
            cout << "Route from " << (*nodep)->get_name() <<
                " @ " << signal->srcpos << " = " <<
                ((*nodep)->pos)[0]+signal->srcpos << " level " <<
                (*nodep)->curvert << " to " <<
                destnode->get_name() << " @ " << signal->destpos <<
                " = " << (*destnode->pos)[0]+signal->destpos <<
                " level " << destnode->curvert << "\n";
            cout << "    Goals: " << *signal->routegoal << "\n";
        }
    }
}

```

## File pr2dblockdisplay.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifdef IV

#include <stream.h>
#include <InterViews/frame.h>
#include <InterViews/graphic.h>
#include <InterViews/Graphic/label.h>
#include <InterViews/Graphic/lines.h>
#include <InterViews/Graphic/picture.h>
#include <InterViews/Graphic/polygons.h>
#include <InterViews/interactor.h>
#include <InterViews/painter.h>
#include <InterViews/transformer.h>
#include <InterViews/world.h>
#include "DAG.h"
#include "intMB.h"
#include "MemBlock.h"
#include "Node.h"
#include "pr2dblock.h"
#include "View.h"

static int maxx, minx;
Transformer* pr2dblock_trans;

static Graphic* make_rect(Node* node, int x, int y, FullGraphic* df)
{
    Coord left, bottom, right, top;

    Picture* pict = new Picture;
    Rect* rect;
    if (node->type == NODE_BLOCK)
        rect = new Rect(x, y, x + node->width()*pr2dblock_xscale/2,
                        y + pr2dblock_yscale, df);
    else
        rect = new Rect(x, y, x + maxx*pr2dblock_xscale/2,
                        y + pr2dblock_yscale, df);
    pict->Append(rect);

    Label* label = new Label(node->get_name(), df);
    label->GetBox(left, bottom, right, top);
    if (node->type == NODE_BLOCK)
```

```

        label→Translate(x+(node→width()*pr2dblock_xscale/2-(right-left))/2,
                        y+(pr2dblock_yscale-(top-bottom)));
    else
        label→Translate(x+maxx*pr2dblock_xscale/4-(right-left)/2,
                        y+(pr2dblock_yscale-(top-bottom)));
    pict→Append(label);
    return (Graphic*)pict;
}

void pr2dblock_display()
{
    World* world = new World("pr2dblock", 0, NULL);
    Picture* pict = new Picture;
    FullGraphic dfault;

    InitPPaint();

    dfault.FillBg(true);
    dfault.SetColors(pblack, pwhite);
    dfault.SetPattern(psolid);
    dfault.SetBrush(psingle);
    dfault.SetFont(pstdfont);

    int ml = pr2ddag→maxlevel;
    maxx = minx = 0;
    Node** nodep;
    Node* input_node;
    Node* output_node;
    memblock_iter(pr2ddag→nodelist, nodep) {
        if ((*nodep)→type == NODE_BLOCK) {
            int x = (*nodep)→get_hpos() * pr2dblock_xscale / 2;
            int y = (ml - (*nodep)→curvert) * pr2dblock_yscale * 2;
            if (((*nodep)→get_hpos()+(*nodep)→width()+1)/2 > maxx)
                maxx = ((*nodep)→get_hpos()+(*nodep)→width()+1)/2;
            if ((*nodep)→get_hpos()/2 < minx)
                minx = (*nodep)→get_hpos()/2;
            Graphic* rect = make_rect(*nodep, x, y, &dfault);
            pict→Append(rect);
        }
        else if ((*nodep)→type == NODE_INPUT)
            input_node = *nodep;
        else
            output_node = *nodep;
    }
    Signal* signal;
    memblock_iter((*nodep)→get_signals(), signal) {
        Node* srcnode = NODEREF(pr2ddag→nodelist, signal→src);
        Node* destnode = NODEREF(pr2ddag→nodelist, signal→dest);
        Coord* xs = new Coord[(destnode→curvert-srcnode→curvert)*2+1];
        Coord* ys = new Coord[(destnode→curvert-srcnode→curvert)*2+1];
        int n = 0;
        xs[n] = pr2dblock_output_bitpos(srcnode,

```



```

                signal→srcpos) *
                pr2dblock_xscale / 2 +
                pr2dblock_xscale / 3;
ys[n] = (ml - srcnode→curvert) * pr2dblock_yscale * 2;
n++;
for (int level=0; level<destnode→curvert-srcnode→curvert;
    level++) {
    int end;
    if (level == destnode→curvert-srcnode→curvert-1)
        end = pr2dblock_input_bitpos(destnode,
            signal→destpos);
    else
        end = (*signal→routegoal)[level];
    if (end > maxx)
        maxx = end;
    if (end < minx)
        minx = end;
    xs[n] = end * pr2dblock_xscale / 2 + pr2dblock_xscale / 3;
    ys[n] = (ml - level - srcnode→curvert - 1) *
        pr2dblock_yscale * 2 + pr2dblock_yscale;
    n++;
    if (level ≠ destnode→curvert-srcnode→curvert-1) {
        xs[n] = xs[n-1];
        ys[n] = ys[n-1]-pr2dblock_yscale;
        n++;
    }
}
pict→Append(new MultiLine(xs, ys, n, &dfault));
delete xs;
delete ys;
}
}

```

```

int x = input_node→get_hpos() * pr2dblock_xscale / 2;
int y = (ml - input_node→curvert) * pr2dblock_yscale * 2;
Graphic* rect = make_rect(input_node, x, y, &dfault);
pict→Append(rect);
x = output_node→get_hpos() * pr2dblock_xscale / 2;
y = (ml - output_node→curvert) * pr2dblock_yscale * 2;
rect = make_rect(output_node, x, y, &dfault);
pict→Append(rect);

```

```

Transformer* tr = new Transformer;
tr→Scale((float)pr2dblock_xsize/((maxx-minx)*pr2dblock_xscale/2),
    (float)pr2dblock_ysize/(ml*2*pr2dblock_yscale));
pict→SetTransformer(tr);
pr2dblock_trans = tr;
Graphics* dsp = new Graphics(pict);

```

```

world→InsertApplication(new ShadowFrame(dsp));
world→Run();

```

```
    delete world;  
}  
#endif
```

## File pr2dblockplace.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "DAG.h"
#include "Node.h"
#include "config.h"
#include "front.h"
#include "inlines.h"
#include "intMB.h"
#include "lib.h"
#include "pr2dblock.h"
#include "util.h"

struct hor_struct {
    int start;
    int end;
};

/*
 * Compute ideal horizontal placements for all of the Nodes. These
 * are placements that minimize routing distance and don't interfere
 * with each other. Horizontal placement is on absolute grid
 * coordinates, and is encoded for odd-level placement.
 *
 * We assume the DAG is sorted by vertical hierarchy position.
 */

void pr2dblock_place_ideal()
{
    DebugPR2D("pr2dblock_place_ideal\n");

    int pr2dblock_routing_pickpos(Node*, MemBlock*);
    Node *pr2dblock_pick_for_placement(MemBlock*, int, int);
    void pr2dblock_sort_positions(int);

#ifdef DEBUGPR2D
    if (verbose) {
        cout << "Placement Pass 1: ";
        cout.flush();
    }
#endif
}
/*
```

```

    * For each level...
    */
    for (int curlevel=0; curlevel ≤ pr2ddag→get_maxlevel(); curlevel++) {
        DebugPR2D("Level " << curlevel << "\n");
#ifdef DEBUGPR2D
        if (verbose) {
            cout << curlevel << "..";
            cout.flush();
        }
#endif
        /*
        * Start a new horizontal placement array, and unmark all of
        * the blocks on this level.
        */
        MemBlock *horarray = new MemBlock(sizeof(hor_struct));
        int levelstart = pr2ddag→level_start(curlevel);
        int levelend = pr2ddag→level_end(curlevel);
        for (int nodenum = levelstart; nodenum ≤ levelend; nodenum++)
            NODEREF(pr2ddag→nodelist, nodenum)→flag = 0;
        /*
        * Keep picking new blocks from this set until they're all
        * picked. For each figure out where it will best go based on
        * minimum routing distance.
        */
        Node* node;
        while (node = pr2dblock_pick_for_placement(pr2ddag→nodelist,
                                                    levelstart, levelend)) {
            DebugPR2D("Chose node " << *node << "\n");
            /*
            * An I/O block has a destined place already...
            */
            if (node→type ≠ NODE_BLOCK)
                continue;
            /*
            * Find location for minimum routing distance.
            */
            int horizmin = pr2dblock_routing_pickpos(node, horarray);
            DebugPR2D("Placing at " << horizmin << "\n");

            /*
            * We've got a fairly good location - install the block
            * there and mark the locations as used.
            */
            struct hor_struct nh;
            nh.start = horizmin;
            nh.end = horizmin+node→width()-1;
            horarray→add(&nh);
            (*node→idealpos)[0] = horizmin;
            (*node→pos)[0] = horizmin;
            (*node→reqleft)[0] = 0;
            (*node→reqright)[0] = 0;

```

```

    }
    pr2dblock_sort_positions(curlevel);
    delete horarray;
}
#ifdef DEBUGPR2D
    if (verbose) {
        cout << "\n";
        cout.flush();
    }
#endif
}

/*
 * Pick a Node for placement – for now, just pick the Nodes in
 * order. This is not optimal. XXX
 */

Node* pr2dblock_pick_for_placement(MemBlock* mb, int start, int end)
{
    for (int i=start; i<=end; i++) {
        if (!NODEREF(mb, i)→flag) {
            NODEREF(mb, i)→flag = 1;
            return NODEREF(mb, i);
        }
    }
    return 0;
}

int pr2dblock_output_bitpos(Node* node, int pos)
{
    if (node→type == NODE_BLOCK)
        return lib_compute_output_bitpos(node→lib, pos)+
            node→get_hpos();
    else
        return node→get_hpos()+pos;
}

int pr2dblock_input_bitpos(Node* node, int pos)
{
    if (node→type == NODE_BLOCK)
        return lib_compute_input_bitpos(node→lib, pos)+
            node→get_hpos();
    else
        return node→get_hpos()+pos;
}

/*
 * Compute the total routing distance for a Node placed at a position
 * in the array using a sum-of-squared-distance approach.
 */

```

```

pr2dblock_routing_dist(Node* node, int pos)
{
    /*
     * Find distance from all modules to this module
     */
    int length = 0;
    Signal** bsignal;
    memblock_iter(node→get_backsignals(), bsignal) {
        Node* fromnode = NODEREF(pr2ddag→nodelist, (*bsignal)→src);
        if (!fromnode→flag)
            fatal("pr2dblock_routing_dist - Reference to Node not yet picked");
        int inc = abs(pr2dblock_output_bitpos(fromnode,
            (*bsignal)→srcpos)-pos);

        length += inc*inc;
    }

    return length;
}

/*
 * Find the best horizontal location for a node.
 */

int pr2dblock_routing_pickpos(Node* node, MemBlock* hor)
{
    int prevlen = -1;
    int prevx;
    for (int x=0;;x++) {
        int lose = 0;
        hor_struct *hs;
        memblock_iter(hor, hs) {
            if (x+node→width()-1 ≥ hs→start && x ≤ hs→end)
                lose = 1;
        }
        if (lose)
            continue;
        if (!node→get_backsignals()→size())
            return x;
        int len = pr2dblock_routing_dist(node, x+node→width()/2);
        if (prevlen == -1) {
            prevlen = len;
            prevx = x;
        }
        if (len > prevlen)
            return prevx;
        prevlen = len;
        prevx = x;
    }
}

/*

```

```

* Sort the vertical ordering by horizontal position
*/

pr2dblock_sort_positions_comp(Node** n1, Node** n2)
{
    return ((*n1)→pos)[0] - ((*n2)→pos)[0];
}

void pr2dblock_sort_positions(int level)
{
    pr2ddag→nodelist→sort_range((PFRI)pr2dblock_sort_positions_comp,
                                pr2ddag→level_start(level),
                                pr2ddag→level_end(level));
    pr2ddag→translate_idx();
}

/*
* Make sure there is sufficient space between the modules of a given
* level.
*/

void pr2dblock_compute_lrreq()
{
    DebugPR2D("\nComputing left-right requirements\n");
#ifdef DEBUGPR2D
    if (verbose) {
        cout << "Placement Pass 2: ";
        cout.flush();
    }
    int outcount = 0;
#endif

    /*
    * First zero-out the reqleft and reqright fields
    */
    Node** nodep;
    memblock_iter(pr2ddag→nodelist, nodep) {
        ((*nodep)→reqleft)[0] = 0;
        ((*nodep)→reqright)[0] = 0;
    }

    /*
    * For each routing (which consists of source block/output-bit and
    * destination block/input-bit), figure out ideal bit hpos'es and
    * do a trial route...
    */
    Signal* signal;
    memblock_iter(pr2ddag→nodelist, nodep) {
        memblock_iter((*nodep)→get_signals(), signal) {
#ifdef DEBUGPR2D
            outcount++;

```

```

if (outcount > 100) {
    outcount = 0;
    if (verbose) {
        cout << ".";
        cout.flush();
    }
}
#endif
DebugPR2D("Route from " <<
    *NODEREF(pr2ddag→nodelist, signal→src) <<
    " at " << signal→srcpos << " to \n          " <<
    *NODEREF(pr2ddag→nodelist, signal→dest) << " at " <<
    signal→destpos << "\n");
/*
 * Figure out where we're coming from and where we're going
 */
Node* srcnode = NODEREF(pr2ddag→nodelist, signal→src);
Node* destnode = NODEREF(pr2ddag→nodelist, signal→dest);
/*
 * Figure out source hpos
 */
int srchpos = pr2dblock_output_bitpos(srcnode, signal→srcpos);
/*
 * Figure out destination hpos
 */
int desthpos = pr2dblock_output_bitpos(destnode, signal→destpos);
/*
 * Fill up the goal array so that it's the right size
 */
int zero=0;
signal→goals = new intMB;
signal→goalhpos = new intMB;
signal→routegoal = new intMB;
for (int i=srcnode→curvert+1; i<destnode→curvert; i++) {
    signal→goals→add(&zero);
    signal→goalhpos→add(&zero);
    signal→routegoal→add(&zero);
}
/*
 * Follow the route and mark nodes that need room around them.
 * Note that each level is already sorted in ascending
 * horizontal order. At this point a goal is a node number.
 * Positive means route around to the right, negative means
 * route around to the left.
 */
int hpos = srchpos;
int incperlevel = (desthpos-srchpos)/
    (destnode→curvert-srcnode→curvert);
int blockright;
for (int level = srcnode→curvert+1; level < destnode→curvert;
    level++, hpos += incperlevel) {

```



```

DebugPR2D("** Level " << level << " hpos " << hpos << "\n");
for (int nodnum = pr2ddag->level_start(level);
     nodnum <= pr2ddag->level_end(level); nodnum++) {
    Node* inway = NODEREF(pr2ddag->nodelist, nodnum);
    Node* nextnode;
    if (nodnum < pr2ddag->level_end(level))
        nextnode = NODEREF(pr2ddag->nodelist, nodnum+1);
    else
        nextnode = NULL;
    DebugPR2D("** Comparing " << inway->get_name() <<
              " and ");
#ifdef DEBUGPR2D
    if (nextnode)
        DebugPR2D(nextnode->get_name() << "\n");
    else
        DebugPR2D("nothing\n");
#endif
    if (inway->get_hpos() + inway->width()/2 > hpos) {
        (*inway->reqlist)[0] += pr2dblock_min_wire_dist;
        blockright = -(inway->idx+1);
        DebugPR2D("*** Left of left\n");
        break;
    }
    if (!nextnode) {
        (*inway->reqlist)[0] += pr2dblock_min_wire_dist;
        blockright = inway->idx;
        DebugPR2D("*** Right of left\n");
        break;
    }
    if (inway->get_hpos() + inway->width()/2 <= hpos &&
        nextnode->get_hpos() + nextnode->width()/2 >
        hpos) {
        (*inway->reqlist)[0] += pr2dblock_min_wire_dist;
        (*nextnode->reqlist)[0] += pr2dblock_min_wire_dist;
        blockright = inway->idx;
        DebugPR2D("*** Between\n");
        break;
    }
}
(*signal->goals)[level-srcnode->curvert-1] = blockright;
(*signal->goalhpos)[level-srcnode->curvert-1] = hpos;
}
}
}

/*
 * Now actually move the nodes around to make room for the routing.
 */

#ifdef DEBUGPR2D
    if (verbose) {

```

```

        cout << "*";
        cout.flush();
    }
#endif
    for (int level=0; level <= pr2ddag->maxlevel; level++) {
#ifdef DEBUGPR2D
        if (verbose) {
            cout << level << " . . ";
            cout.flush();
        }
#endif
        for (int nodnum=pr2ddag->level_start(level);
            nodnum < pr2ddag->level_end(level); nodnum++) {
            Node* node = NODEREF(pr2ddag->nodelist, nodnum);
            if (node->type != NODE_BLOCK)
                continue;
            Node* nextnode = NODEREF(pr2ddag->nodelist, nodnum+1);
            int distbtwn = nextnode->get_hpos() - node->get_hpos() -
                node->width();
            int distneeded = max((*nextnode->reqleft)[0],
                (*node->reqright)[0]);
            if (distbtwn < distneeded) {
                for (int movenum=nodnum+1;
                    movenum <= pr2ddag->level_end(level); movenum++) {
                    Node* nodemove = NODEREF(pr2ddag->nodelist, movenum);
                    (*nodemove->pos)[0] += distneeded - distbtwn;
                    DebugPR2D("Moving " << nodemove->get_name() <<
                        " right by " << distneeded - distbtwn <<
                        "\n");
                }
            }
        }
    }
}

/*
 * Well, we've got the blocks placed nicely now...go back and
 * assign goal columns for the routings. Go through all the
 * levels. For each level, go through the "spaces" to the right
 * of each block. Find all the routings that want to go through
 * that space, and assign them a column. Try to give them a
 * column that they ask for as long as its between the proper
 * blocks and in a reasonable position. Note that the levels are
 * still sorted by horizontal position.
 */

MemBlock* levs = new MemBlock(sizeof(MemBlock*));
for (int i=0; i<=pr2ddag->maxlevel; i++) {
    MemBlock* mb = new MemBlock(sizeof(hor_struct));
    levs->add(&mb);
}

```

```

#ifndef DEBUGPR2D
    if (verbose) {
        cout << "*";
        cout.flush();
    }
    outcount = 0;
#endif
    memblock_iter(pr2ddag->nodelist, nodep) {
        memblock_iter((*nodep)->get_signals(), signal) {
#ifndef DEBUGPR2D
            outcount++;
            if (outcount > 100) {
                outcount = 0;
                if (verbose) {
                    cout << ".";
                    cout.flush();
                }
            }
}
#endif
/*
 * For each routing, follow the goals and compute a
 * reasonable "real" hpos for the goal.
 */
Node* destnode = NODEREF(pr2ddag->nodelist, signal->dest);
for (int curlev=(*nodep)->curvert+1;
     curlev < destnode->curvert; curlev++) {
    MemBlock* curmb = memblock_get_data(levs, MemBlock*, curlev);
    int nodegoalnum = (*signal->goals)[curlev-
                                     (*nodep)->curvert-1];

    int dir;
    int srcpos;
    if (nodegoalnum < 0) {
        /* Go to the left of this node */
        dir = 1;
        Node* node = NODEREF(pr2ddag->nodelist, -nodegoalnum-1);
        srcpos = node->get_hpos()-(*node->reqleft)[0];
    }
    else {
        /* Go to the right of this node */
        dir = 1;
        Node* node = NODEREF(pr2ddag->nodelist, nodegoalnum);
        srcpos = node->get_hpos()+node->width();
    }
    for (; srcpos += dir) {
        hor_struct *hs;
        int lose = 0;
        memblock_iter(curmb, hs) {
            if (srcpos >= hs->start && srcpos <= hs->end)
                lose = 1;
        }
        if (!lose)

```

```

        break;
    }
    hor_struct nh;
    nh.start = srcpos;
    nh.end = srcpos+pr2dblock_min_wire_dist-1;
    curmb→add(&nh);
    (*signal→routegoal)[curlev-(*nodep)→curvert-1] = srcpos;
}
}
}
for (i=0; i≤pr2ddag→maxlevel; i++)
    delete memblock_get_data(levs, MemBlock*, i);
delete levs;
#ifdef DEBUGPR2D
    if (verbose) {
        cout << "\n";
        cout.flush();
    }
#endif
}

```

## File pr2dblockroute.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "Array.h"
#include "DAG.h"
#include "MemBlock.h"
#include "Node.h"
#include "config.h"
#include "front.h"
#include "intMB.h"
#include "lib.h"
#include "pr2dblock.h"
#include "util.h"

int pr2dblock_not_done(MemBlock*, int);
void pr2dblock_add_node_inputs(MemBlock*, Node*, int);
int pr2dblock_place_routed_blocks(MemBlock*, int, int);
void pr2dblock_route_active(MemBlock*, int, int);
void pr2dblock_sort_routes(MemBlock*);
void pr2dblock_place_node_in_array(Node*, int, int);

int gridtoarray(int h, int v)
{
    if (h <= 0)
        h--;
    if (EVEN(v))
        return h/2;
    return (h-1)/2;
}

/*
 * Now actually perform the bottom-up routing
 */

void pr2dblock_route()
{
    MemBlock* routewires;

    routewires = new MemBlock(sizeof(RouteWire));

    /*
     * Clear all of the block marked flags so we can mark which blocks
     */
}
```



```

        cout << vertlevel << "..";
        cout.flush();
    }
#endif
    }
    if (!pr2dblock_not_done(routewires, vertpos) && vertlevel < 1)
        break;
    DebugPR2DRoute("=====\n");
    pr2dblock_route_active(routewires, vertlevel, vertpos);
}
#ifdef DEBUGPR2DROUTE
    if (verbose) {
        cout << "\n";
        cout.flush();
    }
#endif
}

/*
 * Check to see if the routing is done...this is true when all the
 * wires are where they want to go and we're on an even position
 */

int pr2dblock_not_done(MemBlock* routewires, int vertpos)
{
    if (ODD(vertpos))
        return 1;

    RouteWire* wire;
    memblock_iter(routewires, wire) {
        if (wire->inactive)
            continue;
        if (wire->hpos != wire->desthpos)
            return 1;
    }
    return 0;
}

/*
 * Add a block's inputs to the routewire array at the given vertical position
 */

void pr2dblock_add_node_inputs(MemBlock* routewires, Node* node, int vertpos)
{
    DebugPR2DRoute("Adding node inputs for " << node->get_name() << "\n");

    Signal** back;
    memblock_iter(node->get_backsignals(), back) {
        RouteWire rw;
        rw.hpos = pr2dblock_input_bitpos(node, (*back)->destpos);
        rw.starthpos = rw.hpos;
    }
}

```

```

    rw.startvpos = vertpos;
    rw.destnode = NODEREF(pr2ddag→nodelist, (*back)→src);
    rw.srcnode = node;
    rw.desthpos = pr2dblock_output_bitpos(rw.destnode,
                                           (*back)→srcpos);

    rw.signal = *back;
    rw.marked = 0;
    rw.inactive = 0;
    routewires→add(&rw);
    DebugPR2DRoute(rw << "\n");
}
}
}

/*
 * Place a node in the array
 */

void pr2dblock_place_node_in_array(Node* node, int hpos, int vpos)
{
    int x, y;
    MemBlock *curlev;

    DebugPR2DRoute("** placing in hpos " << hpos << " vpos " << vpos << "\n");

    int height = (*node→lib→sizes)[1];

    for (y=0; y<height; y++) {
        for (x=0; x<(*node→lib→sizes)[0]*2; x+=2)
            pr2dblock_array→poke(gridtoarray(hpos+x+(height≠1), vpos-y),
                                  vpos-y,
                                  node→lib→array→peek(x, y));
    }
}

/*
 * If all the outputs from this block are here, and placing the block
 * down won't block any other wires, place the block.
 * XXX This could be made amazingly more efficient
 */

int pr2dblock_place_routed_blocks(MemBlock* routewires, int vertlevel,
                                  int vertpos)
{
    /*
     * See if all of the wires are at their goal columns
     */
    RouteWire* wire;
    memblockiter(routewires, wire) {
        if (wire→inactive)
            continue;
        if (wire→startvpos < vertpos &&

```



```

        wire→hpos ≠ GOAL(wire, vertlevel)) {
        DebugPR2DRoute("Wire " << *wire << " not finished\n");
        return 0;
    }
}

/*
 * Unmark all of the wires...
 */
memblock_iter(routewires, wire)
    wire→marked = 0;

/*
 * For each wire, see if all of the wires going to this wire's
 * block are at their destination locations
 */
for (;;) {
    int allmarked = 1;
    memblock_iter(routewires, wire) {
        if (wire→inactive)
            continue;
        if (!wire→marked) {
            allmarked = 0;
            break;
        }
    }
}
if (allmarked)
    break;

Node* node = wire→destnode;
DebugPR2DRoute("Checking wire to " << node→get_name() << "\n");

/*
 * Mark all of the wires that are going to this block and see
 * if they're all where they want to go.
 */
/* Note the first wire found will be "wire"! */
RouteWire* testwire;
memblock_iter(routewires, testwire) {
    if (testwire→inactive)
        continue;
    if (node == testwire→destnode)
        testwire→marked = 1;
}

/*
 * I/O blocks are completely ignored for placement! Otherwise
 * they won't all be on the same level.
 */
if (node→type ≠ NODE_BLOCK)
    continue;

```

```

/*
 * Is this block even on the right level?
 */
if (node→curvert ≠ vertlevel)
    continue;

/* Check even/odd placement */
int height = (*node→lib→sizes)[1];
if (height == 1 &&
    !((ODD(vertpos) && ODD(node→get_hpos())) ||
      (EVEN(vertpos) && EVEN(node→get_hpos()))))
    continue;
if (height > 1 &&
    !((ODD(vertpos) && EVEN(node→get_hpos())) ||
      (EVEN(vertpos) && ODD(node→get_hpos()))))
    continue;

/*
 * Now see if placing this node down will block any other
 * wires.
 */
int willblock = 0;
memblock_iter(routewires, testwire) {
    if (testwire→inactive || node == testwire→destnode)
        continue;
    if (((testwire→hpos ≥ node→get_hpos()) &&
         testwire→hpos ≤ node→get_hpos()+node→width()-1) ||
        (testwire→hpos < node→get_hpos()) &&
        GOAL(testwire, vertlevel) ≥ node→get_hpos()) ||
        (testwire→hpos > node→get_hpos()+node→width()-1 &&
         GOAL(testwire, vertlevel) ≤ node→get_hpos()+
         node→width()-1)) {
        willblock = 1;
        break;
    }
}
if (willblock)
    continue;

pr2dblock_place_node_in_array(node, node→get_hpos(), vertpos+
    (*node→lib→sizes)[1]-1);

(*node→pos)[1] = vertpos+(*node→lib→sizes)[1]-1;

pr2dblock_add_node_inputs(routewires, node, vertpos+
    (*node→lib→sizes)[1]);
for (int i=0; i<routewires→size();) {
    if (memblock_get_datap(routewires, RouteWire, i)→destnode ==
        node)
        routewires→del(i);
}

```

```

        else
            i++;
    }
    node->flag = 1;
}

int nplaced = 0;

Node** nodep;
memblock_iter(pr2ddag->nodelist, nodep) {
    if ((*nodep)->curvert == vertlevel && (*nodep)->flag &&
        ((*nodep)->pos)[1] < vertpos)
        nplaced++;
}

DebugPR2DRoute("nplaced = " << nplaced << " level = " << vertlevel <<
                "\n");

return nplaced == pr2ddag->level_end(vertlevel)-
        pr2ddag->level_start(vertlevel)+1;
}

/*
 * Sort the RouteWires into ascending horizontal sequence
 */

pr2dblock_sort_routes_comp(RouteWire* r1, RouteWire* r2)
{
    if (r1->inactive && r2->inactive)
        return 0;
    if (r1->inactive)
        return -1;
    if (r2->inactive)
        return 1;
    return r1->hpos - r2->hpos;
}

void pr2dblock_sort_routes(MemBlock* routewires)
{
    routewires->sort((PFRI)pr2dblock_sort_routes_comp);
}

/*
 * For each active wire, determine where it wants to go and try to
 * route it in that direction.
 */

void pr2dblock_route_active(MemBlock* routewires, int vertlevel, int vertpos)
{
    /*
     * First, sort the wires in ascending horizontal order so we can

```

```

    * easily check for two wires right next to each other.
    */
    pr2dblock_sort_routes(routewires);

    RouteWire* wire;
#ifdef DEBUGPR2DROUTE
    DebugPR2DRoute("Pos " << vertpos << ": ");
    memblock_iter(routewires, wire)
        DebugPR2DRoute(" " << wire->hpos << "(" << wire->inactive << ")->" <<
            GOAL(wire, vertlevel));
    DebugPR2DRoute("\n");
#endif

    for (int wirenum = 0; wirenum < routewires->size(); wirenum++) {
        wire = memblock_get_datap(routewires, RouteWire, wirenum);
        if (wire->startvpos > vertpos || wire->inactive)
            continue;
        RouteWire* nextwire;
        if (wirenum < routewires->size()-1)
            nextwire = memblock_get_datap(routewires, RouteWire, wirenum+1);
        else
            nextwire = NULL;
        int goal = GOAL(wire, vertlevel);
        int nextgoal;
        if (nextwire)
            nextgoal = GOAL(nextwire, vertlevel);
        if (nextwire && wire->hpos == nextwire->hpos-1 &&
            ((EVEN(vertpos) && EVEN(wire->hpos)) ||
             (ODD(vertpos) && ODD(wire->hpos)))) {
            if (wire->destnode == nextwire->destnode &&
                goal == nextgoal) {
                /*
                 * XXX Slight inefficiency here if they are within one
                 * of destination - won't necessarily pick right direction.
                 */
                if (wire->hpos < goal) {
                    pr2dblock_array->poke(gridtoarray(wire->hpos, vertpos),
                                           vertpos, array_leftbroadcast);
                    wire->hpos++;
                }
                else
                    pr2dblock_array->poke(gridtoarray(wire->hpos, vertpos),
                                           vertpos, array_rightbroadcast);
                nextwire->inactive = 1;
                wirenum++;
                continue;
            }
            if (goal == nextgoal && wire->destnode != nextwire->destnode)
                fatal("Routing wires with same goals but different destinations!");
            if ((wire->hpos < goal &&
                nextwire->hpos >= nextgoal) ||

```

```

    (nextwire→hpos > nextgoal &&
     wire→hpos ≤ goal)) {
    /*
     * The wire on the left wants to go right and the wire
     * on the right doesn't want to go right also, or vice
     * versa - do a cross over
     */
    pr2dblock_array→poke(gridtoarray(wire→hpos, vertpos),
                          vertpos, array_crossover);
    int temp = wire→hpos;
    wire→hpos = nextwire→hpos;
    nextwire→hpos = temp;
  }
  else {
    /* The wires can't move - keep them where they are */
    pr2dblock_array→poke(gridtoarray(wire→hpos, vertpos),
                          vertpos, array_passthru);
  }
  wirenum++;
}
else {
  if (wire→hpos > goal &&
      ((ODD(wire→hpos) && EVEN(vertpos)) ||
       (EVEN(wire→hpos) && ODD(vertpos)))) {
    /* It wants to and can go left */
    pr2dblock_array→poke(gridtoarray(wire→hpos, vertpos),
                          vertpos, array_crossover);
    wire→hpos--;
  }
  else if (wire→hpos < goal &&
           ((EVEN(wire→hpos) && EVEN(vertpos)) ||
            (ODD(wire→hpos) && ODD(vertpos)))) {
    /* It wants to and can go right */
    pr2dblock_array→poke(gridtoarray(wire→hpos, vertpos),
                          vertpos, array_crossover);
    wire→hpos++;
  }
  else {
    /*
     * The wire wants to stay right where it is or can't
     * go anywhere
     */
    pr2dblock_array→poke(gridtoarray(wire→hpos, vertpos),
                          vertpos, array_passthru);
  }
}
}
}
ostream& operator << (ostream& s, RouteWire& rw)
{

```

```
s << rw.srcnode->get_name() << " @ " << rw.startnpos <<
  " to " << rw.destnode->get_name() << " @ " << rw.destnpos <<
  " cur " << rw.hpos;
if (rw.inactive)
  s << " INACTIVE";
return s;
}
```

## File SPlotFile.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

/*
 * This is a derived work from the PlotFile class supplied with GNU
 * libg++ version 1.34.0
 */

#ifndef _splotfile_h_
#define _splotfile_h_

#include <File.h>

#define SYMBOL_TRIANGLE "triangle"
#define SYMBOL_DIAMOND "diamond"
#define SYMBOL_PLUS "plus"
#define SYMBOL_CROSS "cross"
#define SYMBOL_STAR "star"
#define SYMBOL_CIRCLE "circle"
#define SYMBOL_RECTANGLE "rectangle"
#define SYMBOL_SQUARE "square"
#define SYMBOL_CIRCLEFILLED "circlefilled"
#define SYMBOL_TRIANGLEFILLER "trianglefilled"
#define SYMBOL_DIAMONDFILLED "diamondfilled"
#define SYMBOL_SQUAREFILLED "squarefilled"
#define SYMBOL_RECTANGLEFILLED "rectanglefilled"
#define SYMBOL_NONE 0

#define TYPE_DOTS "dots"
#define TYPE_DASHES "dashes"
#define TYPE_STIPPLE "stipple"
#define TYPE_PAT "pat"
#define TYPE_BOLD "bold"
#define TYPE_FILL "fill"
#define TYPE_NONE 0

#define INTERP_LINEAR "linear"
#define INTERP_XHIST "xhistogram"
#define INTERP_YHIST "yhistogram"
#define INTERP_REGRESSION "regression"
#define INTERP_STEP "step"
#define INTERP_SMOOTH "smooth"
```

```

#define STYLE_INTEGER "integer"
#define STYLE_FLOATING "floating"
#define STYLE_SCIENTIFIC "scientific"
#define STYLE_EXPONENT "exponent"

#define ALIGN_CENTER "center"
#define ALIGN_LEFT "left"
#define ALIGN_RIGHT "right"

class SPlotFile : public File {
protected:
    SPlotFile& operator<<(char*);
    SPlotFile& operator<<(int);
    SPlotFile& operator<<(float);
public:
    File::open;      File::close;  File::raw;
    File::remove;   File::filedesc; File::is_open;
    File::iocount;  File::error;   File::name;
    File::setname;  File::rdstate;  File::put;
    File::eof;      File::fail;    File::bad;
    File::good;     File::clear;   File::failif;
    File::setbuf;   File::writable; File::readable;

    SPlotFile();
    SPlotFile(const char* filename, io_mode m, access_mode a);
    SPlotFile(const char* filename, const char* m);
    SPlotFile(int filedesc, io_mode m = io_writeonly);
    SPlotFile(FILE* fileptr);

    ~SPlotFile();
    operator void*();

    SPlotFile& init(float x, float y, char*);
    SPlotFile& comment(char* comment);
    SPlotFile& text(float x, float y, char* text, char* font=0,
                   int angle=0, char* align=0);

    SPlotFile& new_curve();
    SPlotFile& curve_symbol(char* sym, int symsize=2);
    SPlotFile& curve_type(char* type);
    SPlotFile& curve_label(char* label);
    SPlotFile& curve_font(char* font);
    SPlotFile& curve_interp(char* interp);
    SPlotFile& curve_gray(float gray);

    SPlotFile& curve_func(float xlow, float xhigh, float xdelta, char* func);
    SPlotFile& curve_point_start();
    SPlotFile& curve_point(float x, float y, float confidence=0);
    SPlotFile& curve_point_end();

    SPlotFile& xfont(char* font);

```



```
SPlotFile& xlog();
SPlotFile& xaxis(char* label);
SPlotFile& xaxis(char* label, float xmin, float xmax, float xinterval);
SPlotFile& xstyle(char* style, int precision=2);

SPlotFile& yfont(char* font);
SPlotFile& ylog();
SPlotFile& yaxis(char* label);
SPlotFile& yaxis(char* label, float ymin, float ymax, float yinterval);
SPlotFile& ystyle(char* style, int precision=2);
};

#endif /* _splotfile_h_ */
```

## File SPlotFile.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

/*
 * This is a derived work from the SPlotFile class supplied with GNU
 * libg++ version 1.34.0
 */

#include "SPlotFile.h"

SPlotFile::SPlotFile() {}

SPlotFile::~SPlotFile() {}

SPlotFile::SPlotFile(const char* filename, io_mode m, access_mode a)
:(filename, m, a)
{
    put("New graph\n");
}

SPlotFile::SPlotFile(const char* filename, const char* m) : (filename, m)
{
    put("New graph\n");
}

SPlotFile::SPlotFile(int filedesc, io_mode m = io_writable) : (filedesc, m)
{
    put("New graph\n");
}

SPlotFile::SPlotFile(FILE* fileptr) : (fileptr)
{
    put("New graph\n");
}

void* SPlotFile::operator void*()
{
    return (state & (_bad|_fail))? 0 : this;
}

SPlotFile& SPlotFile::operator << (char* s)
{
    put(s);
}
```

```

    return *this;
}

SPlotFile& SPlotFile::operator << (int n)
{
    put(itoa(n));
    return *this;
}

SPlotFile& SPlotFile::operator << (float n)
{
    char bfr[128];

    sprintf(bfr, "%f", n);
    put(bfr);
    return *this;
}

SPlotFile& SPlotFile::init(float x, float y, char* title)
{
    return (*this) << "size " << x << " by " << y << "\ngraph title " <<
        title << "\n";
}

SPlotFile& SPlotFile::comment(char* s)
{
    return (*this) << ";" << s << "\n";
}

SPlotFile& SPlotFile::text(float x, float y, char* text, char* font=0,
                          int angle=0, char* align=0)
{
    if (font && *font)
        (*this) << "text font " << font << "\n";
    if (angle)
        (*this) << "text angle " << angle << "\n";
    if (align && *align)
        (*this) << "text align " << align << "\n";
    return (*this) << "new text " << x << "," << y << "\n" << text << "\n\n";
}

SPlotFile& SPlotFile::new_curve()
{
    return (*this) << "New curve\ncurve confidence\n";
}

SPlotFile& SPlotFile::curve_symbol(char* sym, int symsize=2)
{
    if (sym && *sym)
        (*this) << "curve symbol " << sym << " Size " << symsize << "\n";
    return *this;
}

```

```

}

SPlotFile& SPlotFile::curve_type(char* type)
{
    if (type && *type)
        (*this) << "curve type " << type << "\n";
    return *this;
}

SPlotFile& SPlotFile::curve_label(char* label)
{
    if (label && *label)
        (*this) << "curve label " << label << "\n";
    return *this;
}

SPlotFile& SPlotFile::curve_font(char* font)
{
    if (font && *font)
        (*this) << "curve font " << font << "\n";
    return *this;
}

SPlotFile& SPlotFile::curve_interp(char* interp)
{
    if (interp && *interp)
        (*this) << "curve interpolation " << interp << "\n";
    return *this;
}

SPlotFile& SPlotFile::curve_gray(float gray)
{
    return (*this) << "curve gray " << gray << "\n";
}

SPlotFile& SPlotFile::curve_func(float xlow, float xhigh, float xdelta,
                                char* func)
{
    return (*this) << "function " << xlow << "," << xhigh << "," <<
        xdelta << ", def " << func << "\n";
}

SPlotFile& SPlotFile::curve_point_start()
{
    return (*this) << "New points\n";
}

SPlotFile& SPlotFile::curve_point(float x, float y, float confidence=0)
{
    if (confidence)
        return (*this) << x << "," << y << " ~" << confidence << "\n";
}

```

```

    return (*this) << x << ", " << y << "\n";
}

SPlotFile& SPlotFile::curve_point_end()
{
    return (*this) << "\n";
}

SPlotFile& SPlotFile::xfont(char* font)
{
    if (font && *font)
        (*this) << "X font " << font << "\n";
    return *this;
}

SPlotFile& SPlotFile::xlog()
{
    return (*this) << "X scale log\n";
}

SPlotFile& SPlotFile::xaxis(char* label)
{
    if (label && *label)
        (*this) << "X label " << label << "\n";
    return *this;
}

SPlotFile& SPlotFile::xaxis(char* label, float xmin, float xmax,
                           float xinterval)
{
    if (label && *label)
        (*this) << "X label " << label << "\n";
    (*this) << "X Minimum " << xmin << "\n";
    (*this) << "X Maximum " << xmax << "\n";
    return (*this) << "X Interval " << xinterval << "\n";
}

SPlotFile& SPlotFile::xstyle(char* style, int precision=2)
{
    if (style && *style)
        if (!strcmp(style, STYLE_INTEGER))
            (*this) << "X numberstyle integer\n";
        else
            (*this) << "X numberstyle " << style << " " << precision << "\n";
    return *this;
}

SPlotFile& SPlotFile::yfont(char* font)
{
    if (font && *font)
        (*this) << "Y font " << font << "\n";
}

```

```

    return *this;
}

SPlotFile& SPlotFile::ylog()
{
    return (*this) << "Y scale log\n";
}

SPlotFile& SPlotFile::yaxis(char* label)
{
    if (label && *label)
        (*this) << "Y label " << label << "\n";
    return *this;
}

SPlotFile& SPlotFile::yaxis(char* label, float ymin, float ymax,
                             float yinterval)
{
    if (label && *label)
        (*this) << "Y label " << label << "\n";
    (*this) << "Y Minimum " << ymin << "\n";
    (*this) << "Y Maximum " << ymax << "\n";
    return (*this) << "Y Interval " << yinterval << "\n";
}

SPlotFile& SPlotFile::ystyle(char* style, int precision=2)
{
    if (style && *style)
        if (!strcmp(style, STYLE_INTEGER))
            (*this) << "Y numberstyle integer\n";
        else
            (*this) << "Y numberstyle " << style << " " << precision << "\n";
    return *this;
}

```

## File SymbolTable.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _symboltable_h_
#define _symboltable_h_

class FromNodeMB;
class MemBlock;
class Node;
class ostream;

enum stab_type { STAB_VAR, STAB_FORMAL, STAB_INPUT, STAB_OUTPUT };

class FromNode {
public:
    int nodeidx;
    int pos;
    friend ostream& operator << (ostream& s, FromNode& f) {
        return s << "[" << f.nodeidx << " pos " << f.pos << "]";
    }
};

class SymbolEntry {
public:
    char* name;
    int level;
    stab_type attrib;
    int bitwidth;
    FromNodeMB* fromnodes;
public:
    friend ostream& operator << (ostream&, SymbolEntry&);
};

class SymbolTable {
protected:
    MemBlock* table;
    int level;
public:
    SymbolTable();
    ~SymbolTable();
    SymbolEntry* add(char*, int, stab_type);
    SymbolEntry* add0(char*, int, stab_type);
    SymbolEntry* lookup(char*);
};
```

```
SymbolEntry* lookup_err(char*);  
void push() { level++; }  
void pop();  
friend ostream& operator << (ostream&, SymbolTable&);  
};  
  
#endif /* _symboltable_h_ */
```



## File SymbolTable.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "FromNodeMB.h"
#include "MemBlock.h"
#include "Node.h"
#include "SymbolTable.h"
#include "util.h"

/*
 * Initialize a new symbol table.
 */

SymbolTable::SymbolTable()
{
    table = new MemBlock(sizeof(SymbolEntry));
    level = 0;
}

/*
 * Destroy a symbol table.  When we destroy a symbol table, we destroy
 * the contents of the symbol entries, too.
 */

SymbolTable::~SymbolTable()
{
    SymbolEntry *entry;

    memblock_iter(table, entry)
        delete entry->fromnodes;
    delete table;
}

/*
 * Add a symbol to a symbol table.  Assume that "name" is going to
 * stay allocated for life.
 */

SymbolEntry *SymbolTable::add(char* name, int width, stab_type attrib)
{
    SymbolEntry newsym, *oldsym;
```

```

memblock_iter(table, oldsym)
    if (!strcmp(oldsym→name, name) && oldsym→level == level)
        return 0;

newsym.name = name;
newsym.bitwidth = width;
newsym.level = level;
newsym.attrib = attrib;
newsym.fromnodes = new FromNodeMB();
table→add(&newsym);
return memblock_get_datap(table, SymbolEntry, table→size()-1);
}

/*
 * Add a symbol to a symbol table on level 0. Assume that "name" is
 * going to stay allocated for life.
 */

SymbolEntry *SymbolTable::add0(char* name, int width, stab_type attrib)
{
    int oldlevel;
    SymbolEntry* ret;

    oldlevel = level;
    level = 0;
    ret = add(name, width, attrib);
    level = oldlevel;

    return ret;
}

/*
 * Lookup a symbol by name – note dynamic scoping rules! Return 0 on
 * entry not found.
 */

SymbolEntry* SymbolTable::lookup(char* name)
{
    SymbolEntry* ret;
    int looklevel;

    /* Note dynamic scoping rules! */
    looklevel = level;

    while (looklevel ≥ 0) {
        memblock_iter(table, ret) {
            if (!strcmp(ret→name, name) && ret→level == looklevel)
                return ret;
        }
        looklevel--;
    }
}

```

```

    return 0;
}

/*
 * Lookup a symbol by name and give an error if it doesn't exist. The
 * error is fatal.
 */

SymbolEntry* SymbolTable::lookup_err(char* name)
{
    SymbolEntry* ret;

    ret = lookup(name);
    if (!ret)
        error("Reference to undeclared variable %s", name);

    return ret;
}

/*
 * Exit the current scoping level and delete all variables that were
 * defined on this level.
 */

void SymbolTable::pop()
{
    SymbolEntry* sym;

    memblock_iter(table, sym) {
        if (sym->level == level) {
            delete sym->fromnodes;
            table->del(sym);
            sym--;
        }
    }
    level--;
}

/*
 * Dump a symbol table to a stream. This entails dumping each entry
 * in the order in which the entries were defined.
 */

ostream& operator << (ostream& s, SymbolTable& stab)
{
    SymbolEntry *sym;

    memblock_iter(stab.table, sym)
        s << *sym << "\n";
    return s;
}

```

```

}

/*
 * Dump a SymbolEntry structure to a stream. Don't print a trailing
 * newline.
 */
ostream& operator << (ostream& s, SymbolEntry& ent)
{
    FromNode *fromnode;

    s.form("%s<%d> on %d attrib %d fromnodes", ent.name, ent.bitwidth,
           ent.level, ent.attrib);
    memblock_iter(ent.fromnodes, fromnode)
        if (fromnode->nodeidx != -1)
            s << " " << *fromnode;
        else
            s << " uninit";
    return s;
}

```

## File util.h

```
/*
 * Origami compiler
 *
 * By Robert S. French
 *
 * Copyright (c) 1988-1990, Massachusetts Institute of Technology
 */

#ifndef _util_h_
#define _util_h_

void fatal(char* ...);
void error(char* ...);
void warning(char* ...);

#define max(a,b) (((a)>(b))?(a):(b))

void *xmalloc(int);
void *xrealloc(void*, int);
void *xsave_string(char*);
void xfree(void*);

void yylangerror(char*);
void yyarcherror(char*);

extern "C" {
    int strcasecmp(char*, char*);
    int strncasecmp(char*, char*, int);
};

#endif /* _util_h_ */
```

## File util.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <std.h>
#include <stdio.h>
#include <stream.h>
#include "util.h"

void* xmalloc(int n)
{
    register void* ret;

    ret = (void*)malloc(n);
    if (!ret)
        fatal("Out of memory");
    return ret;
}

void* xrealloc(void* ptr, int n)
{
    register void* ret;

    ret = (void*)realloc(ptr, n);
    if (!ret)
        fatal("Out of memory");
    return ret;
}

void* xsave_string(char* s)
{
    char* ret = new char[strlen(s)+1];
    strcpy(ret, s);
    return (void*) ret;
}

void xfree(void* s)
{
    if (s)
        free(s);
}

void yylangerror(char* s)
{

```

```
    error(s);  
}  
  
void yyarcherror(char* s)  
{  
    error(s);  
}
```

## File varargs.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stdarg.h>
#include <stdio.h>
#include <stream.h>
#include "flow.h"
#include "front.h"
#include "lex.h"
#include "util.h"

vsprintf(char *str, char *fmt, va_list ap)
{
    FILE f;
    int len;

    f._flag = _IOWRT+_IOSTRG;
    f._ptr = (char *)str;
    f._cnt = 32767;
    len = _doprint(fmt, ap, &f);
    *f._ptr = 0;
    return (len);
}

void fatal(char* s, ...)
{
    char bfr[1024];
    va_list args;

    va_start(args, s);

    vsprintf(bfr, s, args);
    cerr << "INTERNAL FATAL error:\n" << bfr << "\n";
    exit(1);
    va_end(args);
}

void error(char* s, ...)
{
    char bfr[1024];
    va_list args;

    flow_print_stack(cerr);
}
```



```

lex_printpos(cerr);

va_start(args, s);

vsprintf(bfr, s, args);
cerr << bfr << "\n";
longjmp(front_env, 1);
va_end(args);
}

void warning(char* s, ...)
{
    char bfr[1024];
    va_list args;

    flow_print_stack(cerr);
    lex_printpos(cerr);

    va_start(args, s);

    vsprintf(bfr, s, args);
    cerr << "Warning: " << bfr << "\n";
    va_end(args);
}

```

## File VarBits.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifndef _varbits_h_
#define _varbits_h_

#include "intMB.h"

class ostream;

class VarBits {
public:
    char* name;
    int allbits;
    intMB* ref;
public:
    VarBits(char*, int);
    ~VarBits() { delete ref; }
    addref(int newref) { ref→add(newref); }
    setallbits(int i) { allbits = i; }
    intMB* expand();
    friend ostream& operator << (ostream&, VarBits&);
};

#endif /* _varbits_h_ */
```

## File VarBits.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#include <stream.h>
#include "Node.h"
#include "SymbolTable.h"
#include "VarBits.h"
#include "flow.h"
#include "intMB.h"

/*
 * Create a new VarBits.
 */

VarBits::VarBits(char* thename, int theallbits)
{
    name = thename;
    allbits = theallbits;
    ref = new intMB();
}

/*
 * Take a VarBits, and either return a copy of this->ref, or a new list
 * if this->allbits  $\neq$  0. The returned intMB should be deleted when
 * no longer needed.
 */

intMB* VarBits::expand()
{
    SymbolEntry* sym;
    intMB* ret;
    int *bit, *newbit, i;

    ret = new intMB();

    if (!allbits) {
        memblock_iter(ref, bit)
            ret→add(*bit);
        return ret;
    }

    sym = symtab→lookup_err(name);
```

```

    for (i=0; i<sym→bitwidth; i++)
        ret→add(i);

    return ret;
}

/*
 * Dump a VarBits to a stream.
 */

ostream& operator << (ostream& s, VarBits& bits)
{
    register int *bit;
    int first = 1;

    s << bits.name;
    if (bits.allbits)
        return s;
    s << "<";
    memblockiter(bits.ref, bit) {
        if (!first)
            s << " ";
        first = 0;
        s << *bit;
    }
    return s << ">";
}

```

## File View.h

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifdef IV
#ifndef _view_h_
#define _view_h_

#include <InterViews/graphic.h>
#include <InterViews/Graphic/grblock.h>

class Damage;
class MonoScene;
class Picture;

/*
 * Graphics class defines and implements the outer-level interactor for
 * the application. An instance of Graphic is inserted into the world.
 */
class Graphics : public MonoScene {
public:
    Graphics(Picture*);
protected:
    Picture* InitPicture();
};

/*
 * View is a GraphicBlock with its Handle member function redefined to detect
 * hits on the Graphic objects it contains.
 */
class View : public GraphicBlock {
public:
    View(Graphic*);
    virtual ~View();

    void Move(Graphic*, Event&);

    virtual void Handle(Event&);
    virtual void Update();
protected:
    virtual void Draw();
    virtual void Resize();
private:
    void Track(Event&, Rubberband&);
};
#endif
#endif
```

```
protected:
    Damage* damage;
};

#endif /* _view_h_ */
#endif /* IV */
```

## File View.cc

```
/*
 * Origami compiler
 *
 * By Robert S. French <rfrench@athena.mit.edu>
 *
 * Copyright (c) 1988-1990, Robert S. French
 */

#ifdef IV

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <InterViews/bitmap.h>
#include <InterViews/frame.h>
#include <InterViews/graphic.h>
#include <InterViews/panner.h>
#include <InterViews/raster.h>
#include <InterViews/rect.h>
#include <InterViews/sensor.h>
#include <InterViews/transformer.h>
#include <InterViews/tray.h>
#include <InterViews/world.h>
#include <InterViews/Graphic/damage.h>
#include <InterViews/Graphic/ellipses.h>
#include <InterViews/Graphic/grblock.h>
#include <InterViews/Graphic/label.h>
#include <InterViews/Graphic/lines.h>
#include <InterViews/Graphic/picture.h>
#include <InterViews/Graphic/polygons.h>
#include <InterViews/Graphic/rasterrect.h>
#include <InterViews/Graphic/splines.h>
#include <InterViews/Graphic/stencil.h>
#include "pr2dblock.h"
#include "View.h"

Graphics::Graphics (Picture* pict)
{
    View* view = new View(pict);
    Tray* interior = new Tray(view);
    Frame* pannerFrame = new Frame(new Panner(view));

    interior->Align(BottomRight, pannerFrame);
    interior->Propagate(false);
    Insert(interior);
}

```

```

View::View (Graphic* g) : (nil, g)
{
    input = new Sensor(updownEvents);
    input→Catch(KeyEvent);
    damage = nil;
}

View::~~View ()
{
    delete damage;
}

void View::Handle (Event& e)
{
    Coord slop = round(cm/10);
    Picture* pict = (Picture*) graphic;
    Graphic* g;

    if (e.eventType == DownEvent) {
        BoxObj b = BoxObj(e.x-slop, e.y-slop, e.x+slop, e.y+slop);
        g = pict→LastGraphicIntersecting(b);
        if (g ≠ nil) {
            switch (e.button) {
                case LEFTMOUSE:
                    Move(g, e); break;
#ifdef XXX
                case MIDDLEMOUSE:
                    Scale(g, e); break;
                case RIGHTMOUSE:
                    Rotate(g, e); break;
#endif
            }
        }
        } else if (e.eventType == KeyEvent && *e.keystring == 'q') {
        e.target = nil;
    }
}

void View::Move (Graphic* g, Event& e)
{
    Coord l, b, r, t, newl, newb;
    float dx, dy, mag = GetMagnification();

    damage→Incur(g);
    g→GetBox(l, b, r, t);
    SlidingRect sr(output, canvas, l, b, r, t, e.x, e.y);

    Track(e, sr);
    sr.GetCurrent(newl, newb, r, t);
    dx = (newl - l) / mag;

```



```

    dy = (newb - b) / mag;
    g→Translate(dx, dy);
    g→Touch();
    damage→Incur(g);
    Update();
}

void View::Update ()
{
    UpdatePerspective();
    damage→Repair();
}

void View::Draw ()
{
    damage→Reset();
    GraphicBlock::Draw();
}

void View::Resize ()
{
    GraphicBlock::Resize();
    if (damage == nil) {
        damage = new Damage(canvas, output, GetGraphic());
    }
}

void View::Track (Event& e, Rubberband& r)
{
    Coord x, y;
    float mag = GetMagnification();

    r.Draw();

    y = e.y;
    Listen(allEvents);
    do {
        if (e.eventType == MotionEvent) {
            extern Transformer* pr2dblock_trans;
            x = e.x;
            pr2dblock_trans→InvTransform(x, y);
            int inc = (int)((x+pr2dblock_xscale)/(pr2dblock_xscale*2));
            x = (int)(inc*pr2dblock_xscale*2);
            pr2dblock_trans→Transform(x, y);
            r.Track(x, y);
        }
        Read(e);
    } while (e.eventType ≠ UpEvent);
    Listen(input);

    r.Erase();
}

```

```
}
```

```
#endif
```