The Design and Implementation of a Spatial Partitioner for use in a Runtime Reconfigurable System

by

C. David Moye

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE
in
Computer Engineering

APPROVED:
Dr. Peter M. Athanas, Committee Chairman
Dr. Charles Nunnally, Committee Member
Dr. Mark Jones, Committee Member

Blacksburg, Virginia
Keywords: CCM, RTR, Spatial Partitioning

The Design and Implementation of a Spatial Partitioner for use in a Runtime Reconfigurable System

by

C. David Moye

Chairman: Dr. Peter Athanas

Electrical Engineering Department

(ABSTRACT)

Microprocessors have difficulties addressing the demands of today's high-performance embedded applications. ASICs are a good solution to the speed concerns, but their cost and time to market can make them impractical for some needs. Configurable Computing Machines (CCMs) provide a cost-effective way of creating custom components; however, oftentimes it would be better if there were a way to change the configuration of the CCM as a program is executing. An efficient way of doing this is with Runtime Reconfigurable (RTR) computing architectures.

In an RTR system, one challenging problem is the assignment of operators onto the array of processing elements (PEs) in a way as to simultaneously minimize both the number of PEs used and the number of interconnections between them for each configuration. This job is automated through the use of a software program referred to as the Spatial Partitioner.

The design and implementation of the Spatial Partitioner is the subject of this work. The Spatial Partitioner developed herein uses an iterative, recursive algorithm along with cluster refinement to find a reasonably efficient allocation of operators onto the target platform in a reasonable amount of time. Information about the topology of the target platform is used throughout the execution of the algorithm to ensure that the resulting solution is legal in terms of layout.

Acknowledgments

First and foremost, I would like to thank my advisor, Dr. Peter Athanas, for his never-ending confidence in my ability to undertake and complete this project. When I was deciding to return to school after nearly three years in the work force, it was his assurances that told me I was making the right decision. Without his guidance, I may not have returned to school and without his limitless patience, this work may never have become a reality.

I would also like to thank my wife, Katie. It has been her constant support, love and understanding that have made all things possible for me. It's not just any wife that would support her husband in a decision to quit a steady job and return to student life full-time; but then again, she is not just any wife. It is to her that I dedicate this research.

Table of Contents

Table of Figures

Table of Algorithms

Table of Equations

Table of Tables

1. Introduction

Microprocessors alone have severe difficulties addressing the needs of today's high-performance embedded applications. Some of these applications, for example image processing, Fast Fourier Transforms (FFTs) and cryptographic systems, function most effectively when they include Application Specific Integrated Circuits (ASICs) as part of their implementation. ASICs have, in general, a very short production life. Since ASICs are generally produced in relatively small quantities, they are responsible for a disproportionately large percentage of the cost associated with these systems.

In an effort to address this concern and to provide for future extensibility, some complex systems now employ Configurable Computing Machine (CCM) architectures. These CCM platforms are typically composed of Field-Programmable Gate Arrays (FPGAs), Field-Programmable Interconnection Devices (FPIDs), and Static Random-Access Memories (SRAM). An FPGA is a relatively inexpensive commodity device that can be reprogrammed very quickly via machine instructions.

1.1 Configurable Computing Machines

CCMs capture the salient characteristics of microprocessor-based systems and ASIC systems. A CCM, like a microprocessor, can be reconfigured for many different applications. While CCMs are also typically faster than microprocessors, they are more rigid in their

configurability. On the other hand, CCMs are much more flexible than ASICs (which cannot be reconfigured) but are significantly slower.

A CCM consists of four main parts:

1. Configurable Processing Elements (PEs) - PEs are the primary items in CCM architectures. They are made of simple processors that can be reconfigured to fit a programmer's needs. PEs are usually implemented with FPGAs since FPGAs are both reprogrammable (in terms of operation and interconnectivity) and readily available.
2. Memory - Some sort of static memory is needed to store the results of the calculations and actions performed by the PEs.
3. Customizable Communication Networks - In CCMs that contain multiple PEs, a customizable method of communicating between PEs is required.
4. Input/Output Blocks (IOBs) - These are the devices the CCM uses to communicate with the outside world. IOBs allow communication between the CCM and the host processor and provide an interface for programming.

General-purpose microprocessors have the constraint of being static. Once their architectures have been put into place, they are unchangeable. If a designer wants to include a microprocessor as a part of their design, they must be able to work within the limitations of this fixed architecture.

CCMs typically have no inherent architecture. This provides the designer with the flexibility of choosing an architecture that is specifically tailored to their needs. In this respect, CCMs can be used much like ASICs, but with the advantage of reprogrammability.

One of the problems associated with CCMs is partitioning an application in a way that most efficiently utilizes the FPGA array. This is a difficult problem since both the PEs that need to be programmed and the number of interconnections between these PEs need to be minimized simultaneously. The tool completed as a part of this work not only provides an efficient solution to this problem, but is also designed to be used as part of a Runtime Reconfiguration (RTR) system.

1.2 Runtime Reconfiguration Systems

Traditional CCM platforms consist of an array of FPGAs that is programmed in unison to execute a given task [http://www.io.com/~guccione/index.html]. Many of today's applications, for example FFTs, can be scaled to an arbitrarily large size. This scaling property guarantees that no matter how many FPGAs are included in a CCM system, there exists some problem that is too large to fit completely onto that system. The predominately accepted solution to this issue is the use of RTR systems.

One advantage of RTR systems is the concept of *virtual hardware*. Virtual hardware can best be explained as using an RTR system to replace several ASICs. Instead of using a chain of ASICs to perform a given task, the same CCM can be used, in different configurations, to perform the same task. This is similar to a computer's use of *virtual memory*, which consists of *pages* of RAM that can be swapped in and out of the active memory on demand. Simply reprogram the CCM to match the function of each ASIC in turn and tie the outputs from the previous configuration to the inputs of the next. There is, of course, the problem of losing some of the benefits associated with chaining ASICs together in a pipeline. That is, if a series of ASICs were used, while one ASIC was performing its tasks, the previous ASIC in the pipeline could already be processing the next set of inputs. This allows for a certain amount of parallelism. Even though this pipeline loss may exist, the cost difference is usually enough to compensate for this performance degradation (especially since a single CCM can usually be configured to behave as the entire pipeline of ASICs).

RTR systems can also be reprogrammed in a number of ways. From very small, fine-grain, adjustments, such reprogramming a single PE or IOB, to extensive, coarse-grain, adjustments such as reprogramming the entire platform. This varying granularity of reconfiguration makes RTR systems a flexible alternative to the general-purpose microprocessor or ASIC.

There is, however, overhead associated with using an RTR system. For example, in an RTR system, there is time overhead associated with reprogramming the platform that would not

be necessary if ASICs were used. Also, there is the need for a host that handles the reprogramming efforts.

In an RTR system, a large problem is divided into several small pieces, each of which fits onto the CCM platform. These partitions are then arranged in order of execution. The first partition is loaded onto the array and executed. The necessary results of this execution are stored and the second partition is loaded onto the array and executed. This process continues until all partitions have been loaded and executed. From this explanation, there are clearly two distinct types of partitioning that must occur: temporal and spatial.

The Temporal Partitioner divides the job into the time-based partitions mentioned above. The Spatial Partitioner (which is the subject of this work) divides each temporal partition into pieces that fit onto one FPGA in the CCM. It does this in a manner that minimizes both the number of FPGAs needed for each temporal configuration and the number of interconnections between these FPGAs. The high-level diagram shown inFigure 1-1 helps to explain this process.

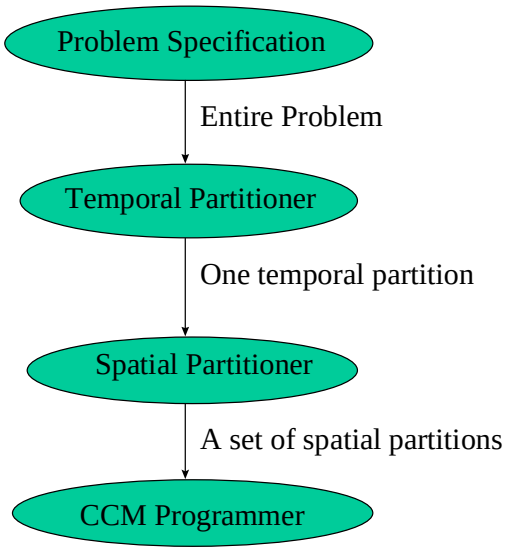


Figure 1-1: Flow of control for CCM programming.

1.3 Research Goal

The goal of the research associated with this thesis is to produce a spatial partitioner that functions in concert with a temporal partitioner and a CCM programmer. Both the Temporal Partitioner and the CCM programmer are being developed by other researchers in the Virginia Tech Configurable Computing Team. The Spatial Partitioner is designed to be a stand-alone executable that accepts as parameters two input files and an output file. The input files contain a textual description of the graph representing the temporal partition to be divided (described in Section 4.4.2) and a textual description of the topology of the target platform (described in Section 4.4.1). The output file contains a similar textual description (described in Section 4.4.3) of a graph. The nodes of the graph contained in the output file represent clusters of operations that fit onto one FPGA and the edges of the output graph represent the interconnections required between these clusters.

The author's specific contributions to this project are outlined below.

- The design of the grammar for the input files
- The design and implementation of the parser for reading the input files
- The design and implementation of the data structure which stores the topology of the target platform
- The design and implementation of the data structure which stores the input graph
- The design and implementation of the iterative, recursive spatial partitioning algorithm
- The design of the grammar for the output file
- The conjoining of all above contributions into a Spatial Partitioner class (implemented in C++) which reads the inputs, partitions the input graph in such a way as to fit onto the target platform in a manner which minimizes the number of interconnections between PEs and writes the results into the output file

1.4 Thesis Organization

Chapter 2 contains information pertaining to the work that has been done related to spatial partitioning. It is in this chapter that alternate implementation methods are discussed. Chapter 2 also provides important background information in the area of CCM technologies and the various methods of spatial partitioning that are currently in use.

In Chapter 3, the approach chosen for this work is explained. The reasons for this selection, as well as the benefits and drawbacks associated with this choice, are discussed in detail. The closing sections of this chapter are dedicated to an outline of the modifications and extensions to the foundation algorithms that were used as a part of this work.

The implementation details of the chosen approach are discussed in Chapter 4. In this chapter, the algorithms used are explained via pseudocode, and the interfaces between the Spatial Partitioner and both the Temporal Partitioner and the CCM programmer are defined. Also found in this chapter is a discussion of some of the constant choices.

In Chapter 5, the development environment is discussed and the experimental results are shown. In addition to the outputs of the Spatial Partitioner, Chapter 5 also contains timing measurements associated with the test samples used for validation of the Spatial Partitioner.

Chapter 6 contains suggestions for areas where future contributions can be made to the Spatial Partitioner and discusses the reasons why these contributions would be of benefit.

Chapter 7 contains the references.

Two Appendices, A and B, contains the source code and the files associated with an MCNC circuit.

2. Background

The automated division of a nontrivial graph into a minimum number of like-sized sections while simultaneously minimizing the number of edges between these sections is a sizable undertaking. In fact, no polynomial-time algorithm has been found which guarantees the discovery of the optimal grouping [8]. Due to the NP-complete nature of this problem, all algorithms that propose to find this sort of grouping are, in reality, only finding the best grouping that can be determined in a reasonable time. There are two main categories of algorithms for partitioning a graph: those which include operator replication, and those which do not. The algorithms of both types that were investigated as a part of the research associated with this work are explained in the following sections.

2.1 Methods Which Employ Operator Replication

Operator Replication is the reproduction of an operator located on one FPGA onto another FPGA in an effort to minimize the interconnections that occur between the two. This is made most clear with a diagram.

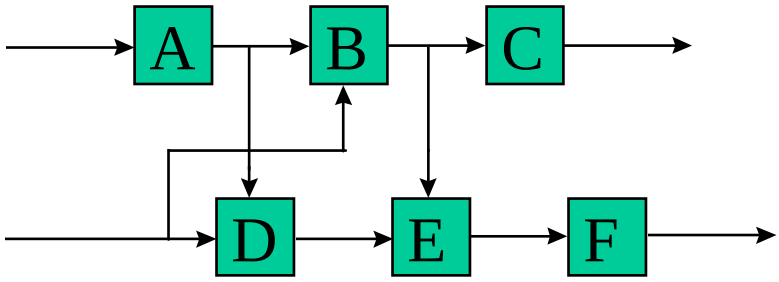


Figure 2-1: An unpartitioned set of operators [4].

Assuming that operators A, B and C fit nicely onto one FPGA and D, E and F fit onto a second FPGA with some room left over, there are two possible partitions. The first of these, the one without operator replication, is shown in Figure 2-2.

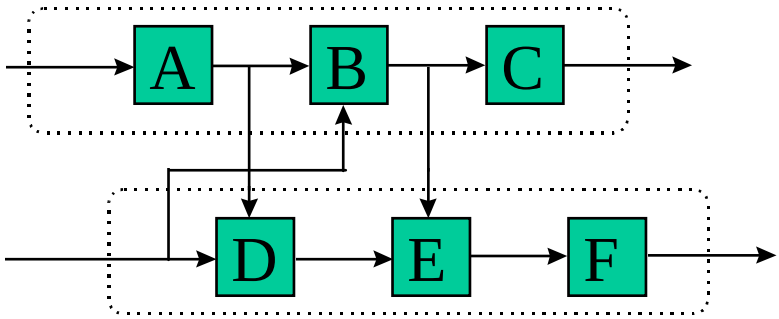


Figure 2-2: The operators partitioned without operator replication [4].

It is easy to see that this choice of partition, while valid, creates three interconnections between the top and bottom partition. It is important to minimize interconnections since decoding logic is required to route the signal to the appropriate FPGA. In addition to this decoding logic, logic within the FPGA to route the signal to the CLB containing the operator is also required. If Operator B were replicated, the number of interconnections could be reduced significantly as is shown in Figure 2-3.

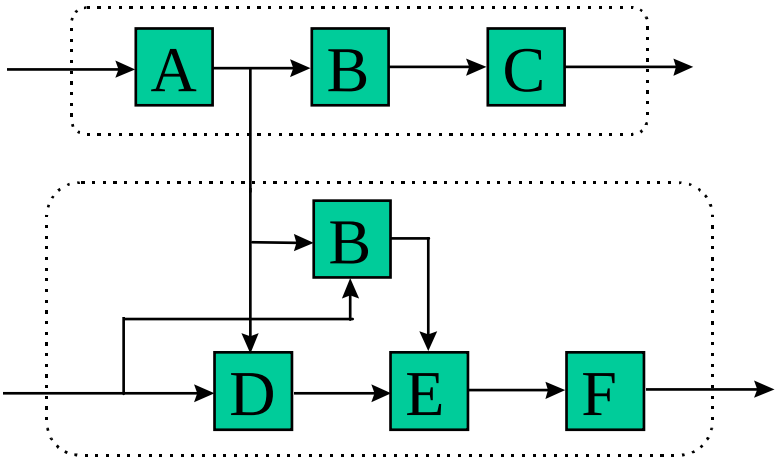


Figure 2-3: The operators partitioned using operator replication [4].

While this partitioning choice relies on Operator B being of a size that will fit into the lower partition, it does reduce the number of interconnections by 67%. If each operator is small (in terms of CLBs required) when compared to the number of CLBs available on each FPGA, then this method can save on the number of interconnections required by a relatively large amount.

The drawback associated with operator replication is not made obvious until it is time to configure the CCM to the next temporal arrangement. At this time, if Operator B is no longer needed, then two FPGAs must be reprogrammed as opposed to one. This increase in the number of FPGAs that needs to be reprogrammed, coupled with the fact that FPGAs must currently be programmed in serial, leads to more time spent programming the CCM and less time spent in execution, which leads to larger overall run times.

Two algorithms that contained operator replication were investigated. The first, developed in [4], uses "traditional" operator replication. Traditional operator replication (as described above) does not rely upon knowledge of the input/output dependencies of the operator being replicated. The second, developed in [1], uses "functional" replication. Functional replication relies on some knowledge pertaining to the input/output dependencies of the operator being replicated in order to impose some heuristics on the replication of that operator. In other

words, it uses this I/O dependency information to determine if there would be an increase or a reduction in the number of interconnections achieved by replicating the operator under inspection.

Since the Spatial Partitioner is developed as a part of a larger RTR system, the choice was made to use an algorithm that did not include operator replication as one of its partitioning considerations. This choice was based upon the increase in the number of FPGAs that need to be reprogrammed between temporal partitions and the resulting increase in run-times which was discussed above. The relationship used to validate this choice is:

$$ax + r = a(x + e) + \frac{r}{d}$$

Equation 2-1: The Break-Even Equation

In this equation, a represents the time required for a single reconfiguration, x is the number of configurations required without operator replication, e is the additional number of reconfigurations required due to the use of operator replication, r is the overall runtime of the system with operator replication and d is the amount by which the runtime is reduced if operator replication is allowed due to the ability to further lessen the number of interconnections between the PEs. By collecting values and rearranging the above equation, Equation 2-2 can be reached.

$$e = \frac{r(d-1)}{da}$$

Equation 2-2: The Simplified Break-Even Equation

In this form, the number of additional reconfigurations forced by operator replication that represents the break-even point for using versus not using operator replication can be easily seen. Since it was assumed that the number of reconfigurations forced by operator replication would

exceed the value on the right-hand side of Equation 2-2, the decision to use an algorithm that does not include operator replication is validated.

2.2 Methods Which do not Employ Operator Replication

As stated in Section 2.1, methods that do not include operator replication can result in an increased number of interconnections between PEs. The primary advantage associated with using such an algorithm is minimizing the number of PEs that need to be reprogrammed when the temporal partition changes.

Given a network, *N*, consisting of a set of cells, *C*, and a set of interconnections, *I*, the solution to the partitioning problem consists of finding a division of *C* into two sets such that the number of interconnections in *I* which have cells in both sets is minimal [3]. This problem can be extended from two-way partitioning (bipartitioning) to *n*-way partitioning simply by recursively calling the two-way partitioning algorithm [7]. This approach will be discussed in greater detail in Chapter 3. For now, we will discuss only bipartitioning algorithms.

In bipartitioning algorithms, the first step (and in a few, degenerate, cases the last) is making sure that the entire temporal partition does not fit onto one PE. If it does, then the partition is trivial, as only one set is needed, and the number of interconnections is therefore zero. Assuming that a nontrivial case exists, the number of partitions, *p*, is greater than one. Also, assuming that each partition needs to communicate with at least one other partition, the number of interconnections, *I*, is at least *p-1*, and probably much larger.

The simplest solution to the partitioning problem would be to require the Temporal Partitioner to create partitions that contain no more operators than there are PEs. The Spatial Partitioner would then just put each operator onto its own PE (assuming that the communications between operators matched the topology of the target CCM). Obviously, this is a waste of processing resources, and would not be seriously considered. Therefore, the job of the Spatial Partitioner becomes the process of determining the best division of the temporal partition that is possible in a reasonable time. This is usually accomplished in an iterative fashion that moves operators, one at a time, from their original partition to the other. For each operator, this move is

first simulated and a *cell gain* is computed. At the end of this pass, the operator with the highest gain is moved and the process repeats until no operator has a positive gain. Gain is usually a function of both the number of interconnections and the size of the larger partition. The size of the larger partition is considered because it is desirable for each partition to be of a similar size. This *balance* is important in order to maintain an even distribution of processing resources.

In a partitioning algorithm that utilizes the above process, the initial division is important. Some initial conditions will provide better final results than others. However, it is not possible to determine the absolute best starting point without knowing what the best result is (which is NP-complete), and therefore, some algorithms (e.g. [6] and [7]) employ a startup sequence that randomly partitions the operator space into various numbers of partitions of varying sizes. The partitioning algorithm is then run on each of these starting partitions and the best result is saved. This process takes longer, but results in a better final partition space [6].

The approach used in [6] also goes one step further in the *cell gain* area by using *cluster refinement*. Cluster refinement is an investigation of the operators in cluster A that share an interconnection with an operator in cluster B, determining a *closeness factor* and then moving the "closest" pair of operators to the same cluster. This process is done before each pass of gain computations. This cluster refinement method is illustrated in Figure 2-4.

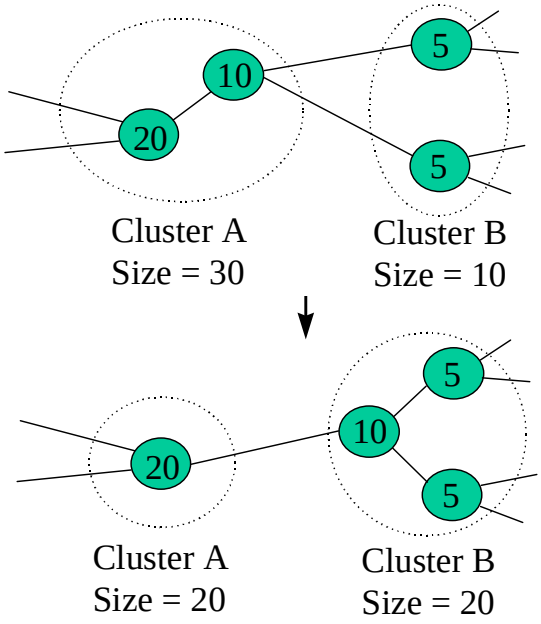


Figure 2-4: An example of cluster refinement.

In the process of cluster refinement, operators are only permitted to be moved to a less-filled (smaller) cluster. This restriction is imposed in order to maintain balance among the cluster sizes [6]. Cluster refinement continues until no further improvement is possible.

Since even the best clustering result may not produce the best final result, the algorithm proposed in [6] partitions starting from several different clustering results and chooses the best solution. These clustering results are also repeated from several different random initial partitions.

3. Approach

The approach selected is an extension of the Gradual, Constraint-Enforcing Partitioning (GCEP) algorithm discussed in [6] which will perform *n*-way partitioning with additional modifications to allow for target platform topology recognition. The addition of the closeness factor used in the cluster refinement portion of this algorithm, coupled with the iterative, hierarchical overall algorithm provided the most seamless and well-organized approach.

3.1 Explanation of Modifications

Several modifications were made to [6] in order to adapt the algorithm for use as a part of the proposed RTR system. These modifications were considered, primarily, because the GCEP algorithm made assumptions about the topology of the target platform. In [6], the author assumed that all PEs were connected to all other PEs. This assumption is not very realistic on the majority of CCM platforms. Under this assumption, the number of interconnections with every PE would be *n-1* where *n* is the number of PEs on the board for a total of *n(n-1)*. This can, needless to say, be a very large number of interconnections on a board of nontrivial size.

3.1.1 Introduction of Topology Recognition

In a non-theoretical system, topology recognition is an important trait of any algorithm. Many CCMs have a limited amount of interconnection logic available in their architectures. Not all PEs have access to input/output pins, nor are all PEs able to communicate with all other PEs.

In fact, in many CCM architectures, the PEs are arranged such that only neighboring PEs are able to communicate with one another and only the top and bottom rows of the array have access to input/output pins (Figure 3-1). This imposes serious restrictions on the placement of operators onto the array.

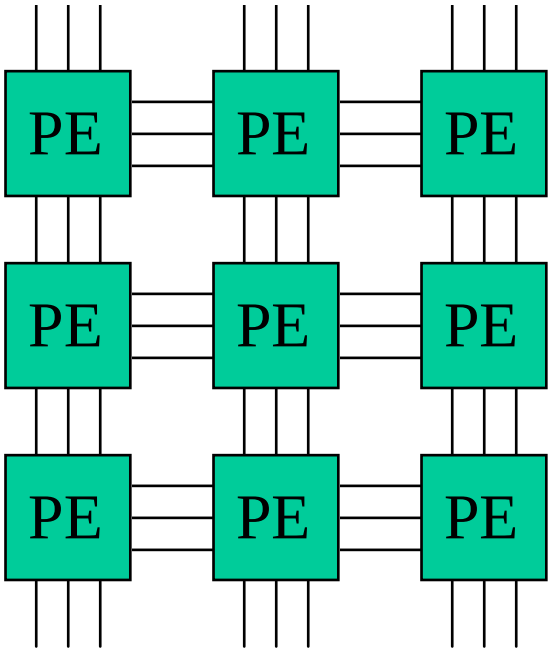


Figure 3-1: An example PE configuration in a CCM.

The Spatial Partitioner that is the subject of this work begins with an legal initial operator configuration that can be placed onto the topology of the target platform and makes all decisions regarding relocation based upon that arrangement. It only allows operators to be moved if they fit, in terms of available CLBs, available IOBs and interconnection logic, onto the target PE. As a result, the output is guaranteed to be a configuration that minimizes both the number of interconnections and the average operator density in such a way that the minimal arrangement found can be placed onto the target platform.

An example of an illegal operator relocation is shown inFigure 3-2.

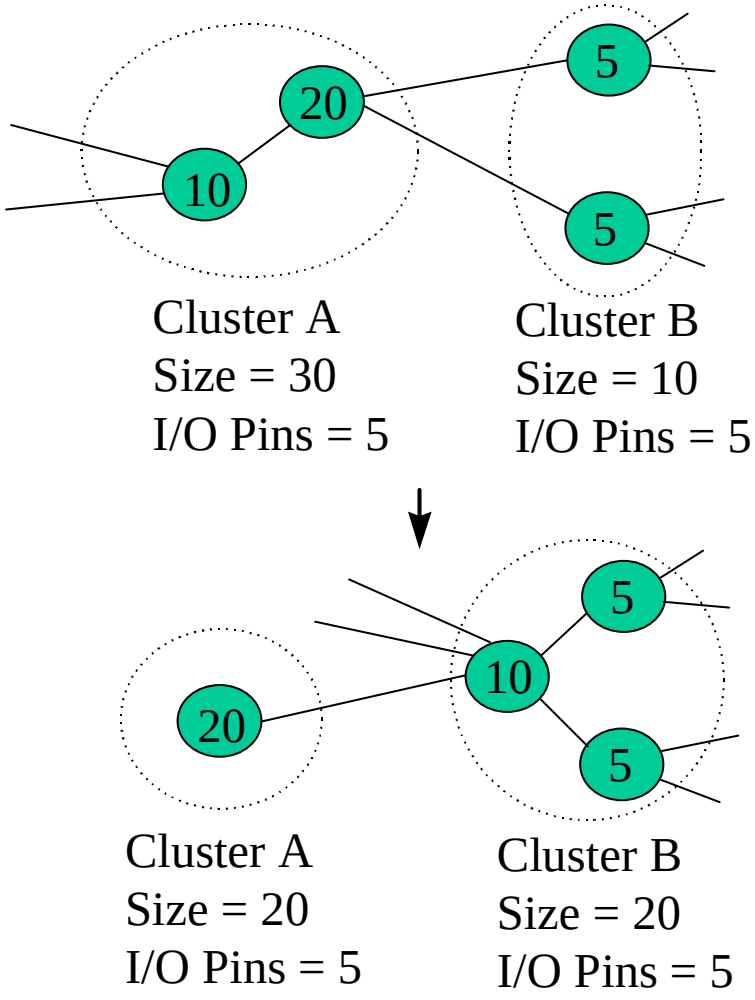


Figure 3-2: An example of an illegal operator relocation.

As shown in Figure 3-2, even though the relocation of the size 10 operator reduces the number of interconnections between Cluster A and Cluster B and also reduces the size difference, it causes the number of I/O pins used by Cluster B (6) to exceed the number of I/O pins available on Cluster B (5). This configuration is, therefore, illegal and would not be allowed by the Spatial Partitioner.

3.1.2 Removal of Random Initial Configuration

The addition of target platform topology recognition added considerable constraints to the positioning of the operators by the algorithm. As a result of these imposed restrictions, the algorithm no longer generates a purely random starting arrangement. The number of valid configurations is so much less than the number of invalid configurations that a large number of random arrangements had to be created during each pass before one was found that fit the target platform. It was determined through testing that the generation of legal random arrangements added an unacceptable amount of time to the overall run time of the Spatial Partitioner for any problem of nontrivial size. As a result, the generation of random initial configurations during each pass was removed. However, in order to maintain the iterative nature of the algorithm, each pass still generates starting arrangements of varying sizes (if possible).

3.1.3 Removal of the Flatten Clusters Step

The algorithm discussed in [6] has a step that is performed after the minimal arrangement is found that converts the internal representation of the operator clusters into a representation suitable for output. The class used to represent a cluster in this work needed no such "flattening"; hence, this step could be removed.

3.1.4 Extension to *n*-way Partitioning

In [6] and [7], the authors put forth an excellent algorithm for bipartitioning a graph. However, since in an RTR system there are many more than two FPGAs, it would be much more efficient to partition into multiple pieces. As a result, the algorithms put forth in [6] and [7] were extended to *n*-way partitioning. This allows the Spatial Partitioner to make a more intelligent decision regarding optimal operator placement.

4. Implementation

Following the approach outlined in Chapter 3, an algorithm for performing spatial partitioning can be derived. This algorithm finds the best solution that can be found in a given number of trials and initial conditions. The solution is minimized in terms of average operator density per device and interconnections required between PEs. The solution is also a legal configuration that will fit onto the topology of the target platform. The overall algorithm is put forth in Section 4.1. Portions of this algorithm that require further discussion are detailed in additional sections. The descriptions of the input and output files are also contained in sections found in this chapter.

4.1 Overall Spatial Partitioning Algorithm

The first algorithm discussed is a high-level overview of the Spatial Partitioning algorithm. It contains references to other algorithms that are described in subsequent sections.

1. Read the topology configuration from input file (Section4.4.1).
2. Read the temporal partition from input file (Section4.4.2).
3. If there are operators in the input graph then:
    a. For some loop count:
        i. Fit the input graph onto the topology of the target platform using a different number of clusters than the last time (Algorithm 4-2).

ii. `cut = gcep()` (Algorithm 4-8).

iii. If `(cut < mincut)` then save this result in `partition`.

b. `gcep(partition)` (Algorithm 4-8).

4. Save the results in the output file (Section 4.4.3).
5. Exit with `EXIT_SUCCESS` if a legal partition can be found, `EXIT_FAILURE` if it cannot.

Algorithm 4-1: Overall spatial partitioning algorithm.

4.2 Make Clusters Algorithm

One essential algorithm in the Spatial Partitioner is the algorithm responsible for mapping the input graph to an initial, legal, configuration and performing cluster refinement on that configuration. This algorithm, the Make Clusters Algorithm (Algorithm 4-2), is discussed in this section. The Make Clusters Algorithm contains references to other algorithms that are discussed in subsections of this section.

1. Initialize the cluster list (Algorithm 4-3).
2. If (`insertVertices()` (Algorithm 4-4)) then:
   a. Set `maxAttempts = 5` and `attempts = 0`.
   b. While (# of non-empty clusters > desired # of clusters) and (# attempts <= maxAttempts):
      i. Find candidates (Algorithm 4-5).
      ii. Merge closest (Algorithm 4-6).
      iii. `attempts++`
   c. Find candidates (Algorithm 4-5).
   d. Refine clusters (Algorithm 4-7).

Algorithm 4-2: Make clusters algorithm.

4.2.1 The Initialize Cluster List Algorithm

Initializing the cluster list is the process of creating an underlying graph that matches the topology of the target platform. This is done at this early stage so that the initial configuration of the operators on the PEs can be compared to the topology. This simple algorithm (Algorithm 4-3) is shown below.

1. For each cluster found in the topology:
   a. Make a copy of that cluster.
   b. Remove the topology identifier from the copy.
   c. Add the copy to the vertex list.

Algorithm 4-3: The initialize cluster list algorithm.

4.2.2 The Insert Vertices Algorithm

As can be seen in Algorithm 4-4, inserting the vertices is a recursive process. This approach was chosen because if, at any point, it is not possible to fit a new vertex into the arrangement due to the positioning of the previous vertices, it becomes necessary to "back out" of the algorithm to the most recent, previous, placement decision and try a different placement. Suffice it to say that the following, recursive, algorithm is performed for each vertex in turn until either all vertices have been added, or it has been determined that no arrangement exists which accommodates all the restrictions of placement. In this latter case, the Spatial Partitioner exits with `EXIT_FAILURE` and the Temporal Partitioner subdivides the graph into a smaller section and reinvokes the Spatial Partitioner with this new partition.

1. Set `added = FALSE` and `clusterCount = 1`.
2. While (not `added` and (`clusterCount` <= `clusters.GetCount()`)):

a. `recipientCluster = clusters.Find(clusterCount)`.

b. Set `fits = legalToAdd(vertex, recipientCluster)`.

c. If (`fits`) then:

i. Add vertex to list.

ii. If (not (`added = insertVertices`(the next vertex))) then:

a) Remove vertex from list.

b) Set `fits = FALSE`.

d. If (not `fits`) then:

i. For each neighbor of `recipientCluster`:

a) If `legalToAdd(vertex, neighboringCluster)`:

i) Add vertex to list.

ii) If (not (`added = insertVertices`(the next vertex))) then:

(a) Remove vertex from list

e. If (not `added`) then:

i. `clusterCount++`.

3. If (not added) then:

a. Return `FALSE`.

4. Return `TRUE`.

Algorithm 4-4: The insert vertices algorithm.

4.2.3 The Find Candidates Algorithm

The Find Candidates Algorithm (Algorithm 4-5) iterates through the cluster list, evaluates and sets the "closeness" of each pairing of clusters using the following equation:

$$\text{closenessValue} = \frac{\alpha \times \text{numConns}}{\min(\text{numNets in c}, \text{numNets in d})} - \frac{\text{size of c} + \text{size of d}}{\text{average cell size}}$$

Equation 4-1: The Closeness Equation

In the above equation, α is a constant that is used to weight the importance of the number of interconnections over the change in cell sizes with respect to the average cell size in the cluster list.

The overall Find Candidates Algorithm is presented below.

1. For each cluster in the cluster list, *c*:
    a. For each cluster remaining in the cluster list, *d*:
        i. If the two are connected then:
            a) If they are "more closely" connected than the other pairings with *c*:
                i) Set the closeness of the two to their "closeness" value
                ii) Set the `closestTo` field of *c* to *d* and vice-versa.

Algorithm 4-5: The find candidates algorithm.

4.2.4 The Merge Closest Algorithm

The Merge Closest Algorithm (Algorithm 4-6) is a simple algorithm that is used after the Find Candidates Algorithm (Algorithm 4-6) to merge the two most closely connected clusters in the cluster list. This step is necessary in order to reduce the number of clusters that are investigated in the Refine Clusters Algorithm (Algorithm 4-7).

1. Find the two most closely connected clusters in an iterative fashion.
2. Move all the operators from the *from* cluster to the *to* cluster.

Algorithm 4-6: The merge closest algorithm.

4.2.5 The Refine Clusters Algorithm

The Refine Clusters Algorithm (Algorithm 4-7) is used once the Merge Closest Algorithm (Algorithm 4-6) has been run in order to "refine" the positioning of the operators within the clusters. This repositioning is used to further reduce the number of interconnections and the average size of the clusters. This algorithm is perhaps the most instrumental in creating a high-quality final result [6].

1. For each cluster in the cluster list, *c*:
    a. For each vertex in that cluster, *v*:
        i. For each cluster remaining in the cluster list, *d*:
            a) If (`legalToAdd(v, d)` and (*c* and *d* are neighbors):
                i) Calculate the gain achieved by moving *v* from *c* to *d*.
                ii) If this is the maximal gain so far, save *v*, *c* and *d*.
            b) Move the vertex with the maximal gain from *c* to *d*.

Algorithm 4-7: The refine clusters algorithm.

4.3 Gradual, Constraint-Enforcing Partitioning (GCEP) Algorithm

The GCEP algorithm is the main partitioning algorithm. The algorithms discussed prior to this section (excluding Algorithm 4-1) are primarily supporting algorithms used for setup purposes. The GCEP algorithm (Algorithm 4-8) is responsible for the main partitioning decisions and for the primary relocation of operators and clusters. The GCEP algorithm used in this work is an extension of the algorithm put forth in [6] from a bipartitioning algorithm to an *n*-way partitioner. This extension is accomplished simply by calling GCEP on each pair of "closest" clusters. Since the algorithm is called in an iterative fashion, it produces the best results at each stage and then continues to the next pass. This provides the best output that can be obtained using the above algorithms.

1. For each cluster in the cluster list, *c*, and the cluster closest to *c*, *d*:
2. Set `done = FALSE`.
3. While (not `done`):
    a. Set `noMoreGain = FALSE`.
    b. `calculateGains(`*c*, *d*`)`.
    c. While (`noMoreGain`):
        i. Set `toMove` = vertex with largest gain in *c*, *v*.
        ii. If size of *c* > `minSize` then:
            a) Move *v* from *c* to *d*.
            b) `calculateGains(`*c*, *d*`)`.
    d. If `bal <= givenBAL` then set `done = TRUE`.
    e. Otherwise, set `bal = max(bal * reductionRatio, givenBAL)`.

Algorithm 4-8: GCEP algorithm.

The GCEP algorithm shown above uses some constants and variables that have not been discussed. These expressions, and the values chosen are outlined in Table 4-1.

Constant/Variable Name	***Value or Expression***	***Description of Constants***
`midSize`	Average size of c and d.	
`givenBAL`	`midSize`/(size of c)	
`reductionRatio`	`0.9`	The amount by which `bal` is reduced during each pass.
`minSize`	`midSize - (midSize * bal)`	
`bal`	`2 * givenBAL`	

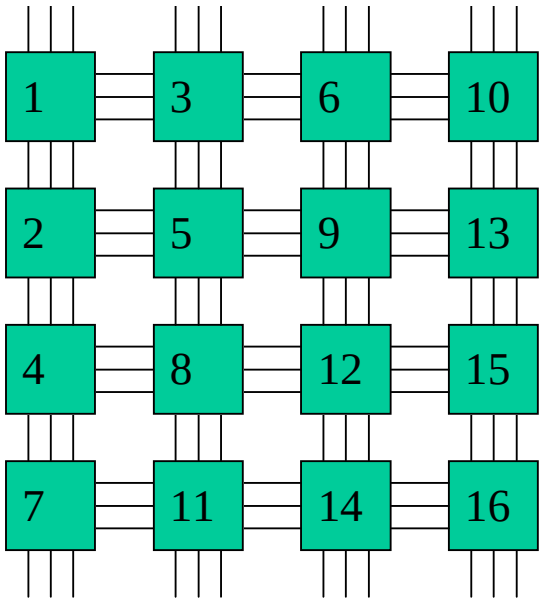
Table 4-1: Explanation of constants and variables.

4.4 Description of the Input and Output Files

The easiest way to explain the structure of the three files that are either read or written by the Spatial Partitioner is by example. The next three subsections will detail the grammar used in these three files by showing an example graph and then a listing of the file that represents each graph.

4.4.1 Description of the Topology Input File

The format of the topology input file (`platform.top`) is a simple grammar. However, it is best described by example. Figure 4-1 shows an example topology. In this diagram, the number of external pins and interconnections between PEs is not shown to scale. Please reference the table associated with the figure for precise details.



External Pin Counts and Sizes

PENumber	Input Only	Output Only	Bidirectional	CLBs
1	5	5	10	10
2	0	0	0	10
3	5	5	10	10
4	0	0	0	10
5	0	0	0	10
6	5	5	10	10
7	5	5	10	10
8	0	0	0	10
9	0	0	0	10
10	5	5	10	10
11	5	5	10	10
12	0	0	0	10
13	0	0	0	10
14	5	5	10	10
15	0	0	0	10
16	5	5	10	10

Figure 4-1: A sample platform topology.

In this topology, each PE has a size (in CLBs) of 10. Only the PEs along the top and bottom rows have external pins. Each PE on these rows has 5 input-only pins, 5 output-only pins

and 10 pins which can be used for either input or output (bidirectional pins). The `platform.top` which represents this topology is shown inFigure 4-2.

```
1:10:5:5:10:2,3;
3:10:5:5:10:1,5,6;
6:10:5:5:10:3,10;
10:10:5:5:10:6,13;

2:10:0:0:0:1,4,5;
5:10:0:0:0:2,3,8,9;
9:10:0:0:0:5,6,12,13;
13:10:0:0:0:9,10,15;

4:10:0:0:0:2,7,8;
8:10:0:0:0:4,5,11,12;
12:10:0:0:0:8,9,14,15;
15:10:0:0:0:12,13,16;

7:10:5:5:10:4,11;
11:10:5:5:10:7,8,14;
14:10:5:5:10:11,12,16;
16:10:5:5:10:14,15;
```

Figure 4-2: The `platform.top` file which represents the sample topology.

Each semi-colon-delimited line in the `platform.top` file begins with the PE number. After the colon signifying the end of the PE number comes the size, in CLBs, of the PE. The next three colon-separated entries are the number of input-only pins, the number of output-only pins and the number of bidirectional pins, respectively. The final item on each line in the topology file is a comma-separated list of PE numbers which have a connection to this PE. The information contained in this topology description file is used to determine the legality of operator placement

and movement throughout the execution of the spatial partitioning algorithm. It is, therefore, important that the information contained in this file is accurate.

4.4.2 Description of the Graph Input File

The format of the graph input file (`graph.file`) (which is created by the Temporal Partitioner) is also a very simple grammar. Again, it is best described by example. Figure 4-3 shows an example input graph. This graph is a good example because it contains a large amount of interconnection dependencies between the operators.

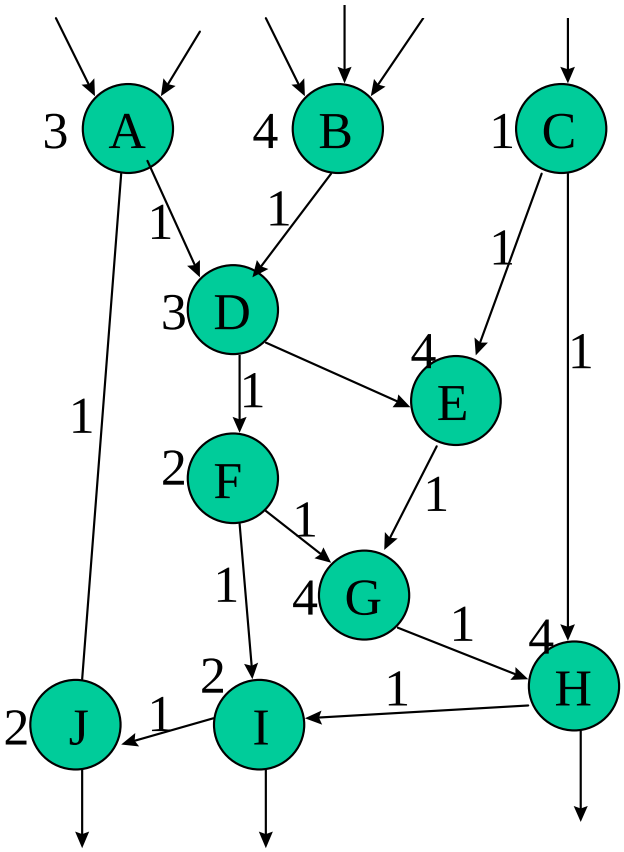


Figure 4-3: An example input graph.

The numbers located to the side of the vertices represent the size, in CLBs, of each vertex. The numbers located on each edge represent the corresponding edge weights. Finally, the arrows are directed along the data flow. The contents of the text file which describes the above diagram is shown below in Figure 4-4.

```
"Test Graph"
{
      A:ADD:3:2:0,
      B:ADDC:4:3:0,
      C:NOT:1:1:0,
      D:SUB:3:0:0,
      E:MULT:4:0:0,
      F:SPLIT:2:0:0,
      G:DIV:4:0:0,
      H:MOD:4:0:1,
      I:OR:2:0:1,
      J:AND:2:0:1;

      AD:A:0:D:0:E:1,
      AJ:A:0:J:0:E:1,
      BD:B:0:D:0:E:1,
      CE:C:0:E:0:E:1,
      CH:C:0:H:0:E:1,
      DE:D:0:E:0:E:1,
      DF:D:0:F:0:E:1,
      EG:E:0:G:0:E:1,
      FG:F:0:G:0:E:1,
      FI:F:0:I:0:E:1,
      GH:G:0:H:0:E:1,
      HI:H:0:I:0:E:1,
      IJ:I:0:J:0:E:1;
}
```

Figure 4-4: The contents of the sample `graph.file`.

In the `graph.file`, the name of the graph, enclosed in quotation marks, is the first thing recognized in the file. Next, the parser looks for an open brace, "{", followed by the first line of data. The first section of the data contains a comma-separated list of vertices. The description for each vertex is as follows:

Vertex Name:Vertex Type:Size:Inputs Required:Outputs Required

Each vertex is separated from each other vertex by a comma and there is a semi-colon following the final vertex entry.

After the vertices, the edges are listed. Again, each edge is separated from each other edge by a comma and there is a semi-colon following the last edge. The information contained in the edge area is as follows:

Edge Name:Source:Source Port:Sink:Sink Port:Edge Type:Weight

The final token which is recognized in the`graph.file` is the closing brace, "}", which signifies the end of the input graph.

4.4.3 Description of the Output File

The output file (`clusters.file`) which is created when the program ends is very similar in structure to the two input files. Below is an example of the output file generated by the Spatial Partitioner when run using the above files as inputs.

```
1:A,B,C,I;
2:D,E,F;
3:H,J;
5:G;
```

Figure 4-5: A sample `clusters.file`.

In the `clusters.file`, each line shows the contents of a PE on the target platform. The first number represents the PE number assigned in the `platform.top` file. After the first colon delimiter, there is a comma-separated list of vertices, which map to the vertex names found in `graph.file`, which are to be placed into each PE.

5. Results

5.1 Development Environment

The Spatial Partitioner was implemented in C++ using MFC and Microsoft™ Visual Developer Studio™ 5.0. The development platform was a Pentium Pro™ 200 with 32 MB of RAM. While the Spatial Partitioner was developed to function as a stand-alone application, it is entirely encapsulated into a series of C++ classes and can, therefore, be integrated into future designs where it is not necessarily stand-alone.

5.2 Functional Testing

The Spatial Partitioner was testing using several input graphs of varying complexity in concert with several different target platforms. In each case, the Spatial Partitioner found a configuration which minimized the number of interconnections between PEs if such a configuration existed. The table below shows a synopsis of this testing (using the `platform.top` file found in Figure 4-4).

# of Vertices	*# of Edges*	*Minimum Cut*	*PEs Required*	*Run Time (ms)*
10	13	0	1	34
20	18	0	2	110
50	27	5	4	210
100	54	15	8	300

Table 5-1: Summary of test results.

Below is the `platform.top` file associated with the above timing measurements. It is similar to the one shown in Figure 4-2, however the default size for all PEs is 50 instead of 10.

```
1:50:5:5:10:2,3;
3:50:5:5:10:1,5,6;
6:50:5:5:10:3,10;
10:50:5:5:10:6,13;

2:50:0:0:0:1,4,5;
5:50:0:0:0:2,3,8,9;
9:50:0:0:0:5,6,12,13;
13:50:0:0:0:9,10,15;

4:50:0:0:0:2,7,8;
8:50:0:0:0:4,5,11,12;
12:50:0:0:0:8,9,14,15;
15:50:0:0:0:12,13,16;

7:50:5:5:10:4,11;
11:50:5:5:10:7,8,14;
14:50:5:5:10:11,12,16;
16:50:5:5:10:14,15;
```

Figure 5-1: `platform.top` file used for above timing measurements.

In addition to the above battery of testing, the Spatial Partitioner was run against one "real-world" circuit. This is circuit c1355nr from the MCNC sample circuit listings. Using the input files located in Appendix B, the Spatial Partitioner arrived at the result shown in Appendix B.3 (3 PEs, 88 interconnections) in 313511 milliseconds (just over 5 minutes).

6. Areas for Future Work

There are several ways in which this Spatial Partitioner could be improved. The sections below give some suggestions for future improvement.

6.1 Using Windows NT Overlapped I/O for File I/O

Windows NT provides a mechanism for allowing a program to continue executing during a lengthy read or write command. This allows the user to issue a read or a write request and then continue processing until the operating system signals the program that the operation has completed. At this time, the program can pick up the data and process it. This would allow for some pipelining in the reading of the input files and the writing of the output files which would reduce the overall execution time.

6.2 Further Adjustment of the Closeness Equation and the Constants Therein

The closeness equation, Equation 4-1, could be adjusted to maximize efficiency for each specific problem by allowing the user to include the constants and some other adjustment factors as parameters to the program. This added flexibility would empower the user with the ability to specifically tailor the closeness calculations to match their desired results.

6.3 Inclusion of a Netlist-to-Graph Converter

The existence of a program which converted standard netlists to the format readable by the Spatial Partitioner would allow for a much-shortened setup time. During the functional testing phase of this project, all input graphs were constructed by hand (with the exception of circuit c1355nr, for which a simple MCNC-SP translator was written). This was a very time-consuming process, and a netlist-to-graph converter would greatly speed up this process.

6.4 The Addition of "Fixed" Operators

The Spatial Partitioner currently does not support the idea of partial reconfiguration. That is, reconfiguration of only a subset of the PEs. The inclusion of a *fixed* flag on certain operators which are known to be present in at least two consecutive temporal partitions would enable the Spatial Partitioner to group those operators together in such a way as to minimize the number of PEs which need to be reconfigured between temporal sessions. Since PEs must currently be programmed in a serial fashion, this has the potential of greatly reducing the reprogramming time and, therefore, allowing the target to spend less of a percentage of its time in a program cycle and a larger percentage in active execution.

6.5 Cost Functions for PEs

On the same target platform, there may be different generations of FPGAs or even FPGAs which are better suited for some tasks than others on the same board. As a result, a valuable addition to the Spatial Partitioning algorithm would be the inclusion of a "cost" function for each PE. If this addition were made, the Spatial Partitioner could prefer certain types of PEs for certain types of operators, and could utilize the most efficient (lowest cost function) PEs first in order to increase overall performance of the application.

7. References

[1] Roman Kuznar, Franc Brglez and Baldomir Zajc. *A Unified Cost Model for Min-Cut Partitioning with Replication Applied to Optimization of Large Heterogeneous FPGA Partitions.* ACM/IEEE 31st Design Automation Conference, Pages 271-276, 1994.

[2] Laura A. Sanchis. *Multiple-Way Network Partitioning*. IEEE Transactions on Computers, Volume 38, Number 1, Pages 62-74, January 1989.

[3] C. M. Fiduccia and R. M. Mattheyses. *A Linear-Time Heuristic for Improving Network Partitions.* ACM/IEEE 19th Design Automation Conference, Pages 175-180, 1982.

[4] Nozomu Togawa, Masao Sato and Tatsuo Ohtsuki. *A Performance-Oriented Circuit Partitioning Algorithm with Logic-Block Replication for Multi-FPGA Systems*. Proceedings of IEEE Asia Pacific Conference on Circuits and Systems, Pages 294-297, November 1996.

[5] U. Ober and M. Glesner. *Multiway Netlist Partitioning onto FPGA-based Board Applications*. ACM/IEEE 32nd Design Automation Conference, Pages 150-155, 1995.

[6] Namhoon Kim, Chunghee Kim and Hyunchul Shin. *An Improved Partitioning Method Using Cluster Refinement*. Proceedings of IEEE Asia Pacific Conference on Circuits and Systems, Pages 302-305, November 1996.

[7] Hyunchul Shin and Chunghee Kim. *A Simple Yet Effective Technique for Partitioning*. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume 1, Number 3, Pages 380-386, September 1993.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

Appendix A Spatial Partitioner Source Code

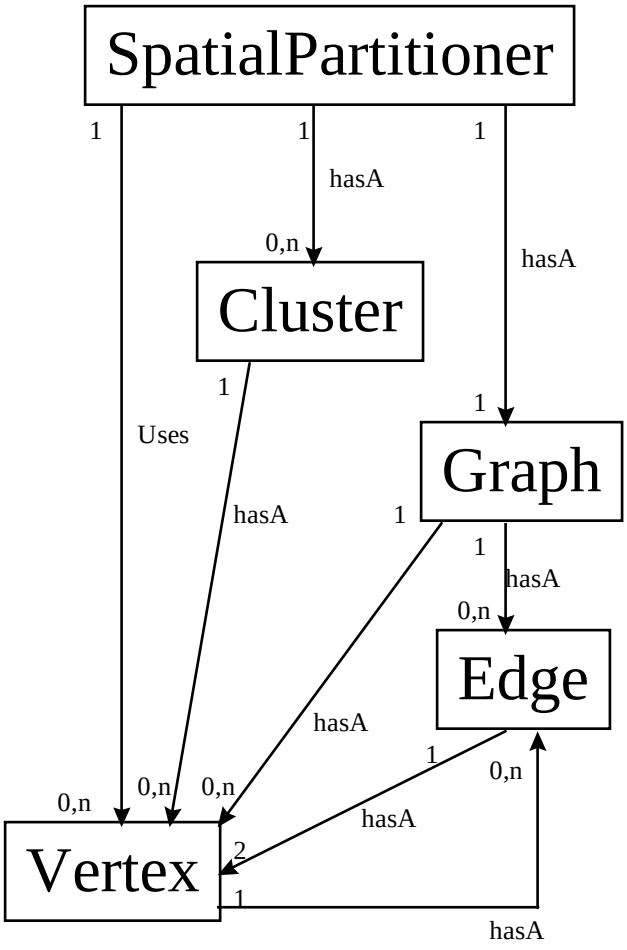


Figure A-1: Object Relationship Diagram

A.1 The Tools Library

A.1.1 RTRDefs.h

```
/*****************************************************************************************
**                                                                                      **
** RTRDefs.h                                                                            **
**                                                                                      **
** This file contains all of the constants which are used by the MFC library in order to  **
** provide the user-defined classes with unique identifiers.                            **
**                                                                                      **
*****************************************************************************************/
#ifndef RTRDEFS_H
#define RTRDEFS_H

#include <afx.h>

const UINT GRAPH_BASE_ID = 0x100;

const UINT EDGE_ID   = GRAPH_BASE_ID + 1;
const UINT GRAPH_ID  = GRAPH_BASE_ID + 2;
const UINT VERTEX_ID = GRAPH_BASE_ID + 3;

const UINT SPATIALPARTITIONER_BASE_ID = 0x200;

const UINT SPATIALPARTITIONER_ID = SPATIALPARTITIONER_BASE_ID + 1;
const UINT CLUSTER_ID            = SPATIALPARTITIONER_BASE_ID + 2;

#endif RTRDEFS_H
```

A.1.2 osindent.h

```
/*****************************************************************************************
**                                                                                      **
** osindent.h                                                                           **
**                                                                                      **
** This file contains the wrappers used in the various printOn() methods around iomanip.  **
** These wrappers provide a much more efficient way to output a series of spaces of       **
** varying length.                                                                      **
**                                                                                      **
*****************************************************************************************/
#ifndef _OSINDENT_H
#define _OSINDENT_H

#include <iostream.h>
#include <iomanip.h>

// print the right number of spaces, given by i
ostream& indent(ostream& os, int i);

#define omanip_int OMANIP(int)

inline omanip_int indent(int i)
{
        return omanip_int(&indent, i);
}

// indentation increment
const int INDENT_INC = 3;

// return the indentation for a given level
inline int inlevel(int i)
{
        return i * INDENT_INC;
}

#endif _OSINDENT_H
```

A.1.3 osindent.cpp

```
/*****************************************************************************************
**                                                                                      **
** osindent.cpp                                                                         **
**                                                                                      **
** This file contains the wrappers used in the various printOn() methods around iomanip.  **
** These wrappers provide a much more efficient way to output a series of spaces of       **
** varying length.                                                                      **
**                                                                                      **
*****************************************************************************************/
#include "osindent.h"
#include <ostream.h>

// print the right number of spaces, given by i
ostream& indent(ostream& os, int i)
{
        for (int j = 0; j < i; j++)
        {
                os << ' ';
        }

        return os;
}
```

A.2 The Graph Library

A.2.1 Vertex.h

```
/*********************************************************************************************
**                                                                                          **
** Vertex.h                                                                                 **
**                                                                                          **
** This class represents a single Vertex which is designed to have Edges connected to it  **
** and to be contained in a Graph for use in a SpatialPartitioner which uses Clusters as  **
** containers.                                                                              **
**                                                                                          **
*********************************************************************************************/
#ifndef VERTEX_H
#define VERTEX_H

#include <afx.h>
#include <afxtempl.h>
#include <iostream.h>

#include "Edge.h"

class Vertex;

typedef CList<Vertex*, Vertex*> VertexList;

class Vertex : public CObject
{
	DECLARE_SERIAL(Vertex)

	public:
		// Default ctor
		Vertex(const CString& name_ = "", const CString& type_ = "", INT weight = 0,
			UINT inputsRequired_ = 0U, UINT outputsRequired_ = 0U);
		// Copy ctor
		inline Vertex(const Vertex& rvalue_);
		// Default dtor
		virtual ~Vertex();

		BOOL addIncomingEdge(Edge* e_);
		BOOL addOutgoingEdge(Edge* e_);

		inline UINT getDegree() const;
		inline UINT getInDegree() const;
		inline UINT getOutDegree() const;
		inline UINT getInputsRequired() const;
```

```
                inline UINT getOutputsRequired() const;
                inline EdgeList* getIncomingEdgeList() const;
                inline EdgeList* getOutgoingEdgeList() const;
                inline const CString& getName() const;
                inline const CString& getType() const;
                inline INT getWeight() const;
                inline BOOL isValid() const;
                // These next two methods tell whether or not this Vertex has been added to a
Cluster
                inline VOID setAdded(BOOL val);
                inline BOOL isAdded() const;
                inline VOID setClusterNumber(UINT num_);
                inline UINT getClusterNumber() const;
                // These next two methods set the gain associated with moving this Vertex to
another
                // Cluster
                inline VOID setGain(INT);
                inline INT getGain() const;

                VOID printOn(ostream& os_ = cout, size_t indentLevel = 5) const;
                VOID Serialize(CArchive& ar_);

                inline Vertex& operator=(const Vertex& rvalue_);
                inline BOOL operator==(const Vertex& rvalue_) const;
                inline BOOL operator!=(const Vertex& rvalue_) const;

        protected:
                BOOL valid;

        private:
                VOID deepCopy(const Vertex& rvalue_);
                VOID printEdgeList(ostream& os_, size_t indentFiveMore_, const EdgeList* list_,
                        const CString& listName_) const;

                BOOL added;
                UINT clusterNumber;
                CString name;
                CString type;
                UINT inDegree;
                UINT outDegree;
                UINT inputsRequired;
                UINT outputsRequired;
                INT gain;
                INT weight;
                EdgeList* incomingEdges;
                EdgeList* outgoingEdges;
};
```

```
inline
Vertex::Vertex(const Vertex& rvalue_) :
        incomingEdges(NULL), outgoingEdges(NULL)
{
        deepCopy(rvalue_);
}

inline
VOID
Vertex::setClusterNumber(UINT val)
{
        clusterNumber = val;
}

inline
UINT
Vertex::getClusterNumber() const
{
        return clusterNumber;
}

inline
UINT
Vertex::getDegree() const
{
        return getInDegree() + getOutDegree();
}

inline
UINT
Vertex::getInDegree() const
{
        return inDegree;
}

// sets whether or not this vertex is part of a cluster
inline
VOID
Vertex::setAdded(BOOL val)
{
        added = val;
}

// returns whether or not this vertex is part of a cluster
inline
BOOL
```

```
Vertex::isAdded() const
{
        return added;
}

inline
UINT
Vertex::getOutDegree() const
{
        return outDegree;
}

inline
UINT
Vertex::getInputsRequired() const
{
        return inputsRequired;
}

inline
UINT
Vertex::getOutputsRequired() const
{
        return outputsRequired;
}

inline
EdgeList*
Vertex::getIncomingEdgeList() const
{
        return incomingEdges;
}

// sets the gain associated with moving this vertex from one cluster to another
inline
VOID
Vertex::setGain(INT gain_)
{
        gain = gain_;
}

// returns the gain associated with moving this vertex from one cluster to another
inline
INT
Vertex::getGain() const
{
        return gain;
```

```
}

inline
EdgeList*
Vertex::getOutgoingEdgeList() const
{
        return outgoingEdges;
}

inline
const CString&
Vertex::getName() const
{
        return name;
}

inline
const CString&
Vertex::getType() const
{
        return type;
}

inline
INT
Vertex::getWeight() const
{
        return weight;
}

inline
BOOL
Vertex::isValid() const
{
        return valid;
}

inline
Vertex&
Vertex::operator=(const Vertex& rvalue_)
{
        deepCopy(rvalue_);

        return *this;
}

inline
```

```
BOOL
Vertex::operator==(const Vertex& rvalue_) const
{
        return name == rvalue_.name;
}

inline
BOOL
Vertex::operator!=(const Vertex& rvalue_) const
{
        return !operator==(rvalue_);
}

#endif VERTEX_H
```

A.2.2 Vertex.cpp

```
/*********************************************************************************************
**                                                                                          **
** Vertex.cpp                                                                               **
**                                                                                          **
** This class represents a single Vertex which is designed to have Edges connected to it  **
** and to be contained in a Graph for use in a SpatialPartitioner which uses Clusters as  **
** containers.                                                                              **
**                                                                                          **
*********************************************************************************************/
#include "osindent.h"
#include "RTRDefs.h"
#include "Vertex.h"

IMPLEMENT_SERIAL(Vertex, CObject, VERTEX_ID)

// Default ctor
Vertex::Vertex(const CString& name_, const CString& type_, INT weight_, UINT inputsRequired_,
                        UINT outputsRequired_) :
        name(name_),                type(type_),                inputsRequired(inputsRequired_),
outputsRequired(outputsRequired_),
        inDegree(0), outDegree(0), weight(weight_), gain(0), added(FALSE), clusterNumber(0),
        valid(FALSE)
{
        incomingEdges = new EdgeList;
        outgoingEdges = new EdgeList;

        valid = TRUE;
}

// Default dtor
Vertex::~Vertex()
{
}

// copies everything
VOID
Vertex::deepCopy(const Vertex& rvalue_)
{
        if (&rvalue_ != this)
        {
                name = rvalue_.name;
                added = rvalue_.added;
                clusterNumber = rvalue_.clusterNumber;
                type = rvalue_.type;
```

```
                inDegree = rvalue_.inDegree;
                outDegree = rvalue_.outDegree;
                inputsRequired = rvalue_.inputsRequired;
                outputsRequired = rvalue_.outputsRequired;
                weight = rvalue_.weight;

                delete incomingEdges;
                incomingEdges = new EdgeList;
                incomingEdges->AddHead((EdgeList*)(rvalue_.incomingEdges));  //  have  to  cast  away
const

                delete outgoingEdges;
                outgoingEdges = new EdgeList;
                outgoingEdges->AddHead((EdgeList*)(rvalue_.outgoingEdges));  //  have  to  cast  away
const

                gain = rvalue_.gain;
                valid = rvalue_.valid;
        }
}

BOOL
Vertex::addIncomingEdge(Edge* e_)
{
        incomingEdges->AddHead(e_);
        inDegree++;

        return TRUE;
}

BOOL
Vertex::addOutgoingEdge(Edge* e_)
{
        outgoingEdges->AddHead(e_);
        outDegree++;

        return TRUE;
}

VOID
Vertex::Serialize(CArchive& ar)
{
        CObject::Serialize(ar);

        incomingEdges->Serialize(ar);
        outgoingEdges->Serialize(ar);
```

```
	if (ar.IsStoring())
	{
		ar << name;
		ar << clusterNumber;
		ar << added;
		ar << type;
		ar << inDegree;
		ar << outDegree;
		ar << inputsRequired;
		ar << outputsRequired;
		ar << weight;
		ar << gain;
		ar << valid;
	}
	else
	{
		ar >> name;
		ar >> clusterNumber;
		ar >> added;
		ar >> type;
		ar >> inDegree;
		ar >> outDegree;
		ar >> inputsRequired;
		ar >> outputsRequired;
		ar >> weight;
		ar >> gain;
		ar >> valid;
	}
}

VOID
Vertex::printEdgeList(ostream& os, size_t indentFiveMore, const EdgeList* list,
                                        const CString& listName) const
{
	if (list->GetCount())
	{
		Edge* e;
		Edge* tail = list->GetTail();
		POSITION pos = list->GetHeadPosition();
		UINT idx = 0;

		do
		{
			e = list->GetNext(pos);
			os << indent(indentFiveMore) << listName << "[" << idx++ << "] = ";
			e->printOn(os, indentFiveMore);
		} while ((e != tail) && (pos != NULL));
```

```
	}
}

VOID
Vertex::printOn(ostream& os, size_t indentLevel) const
{
	UINT indentFiveMore = indentLevel + 5;

	os << "Vertex {" << endl;

	os << indent(indentFiveMore) << "name = " << name << endl;
	os << indent(indentFiveMore) << "added = " << added << endl;
	os << indent(indentFiveMore) << "clusterNumber = " << clusterNumber << endl;
	os << indent(indentFiveMore) << "type = " << type << endl;
	os << indent(indentFiveMore) << "inDegree = " << inDegree << endl;
	if (inDegree > 0)
	{
		printEdgeList(os, indentFiveMore, incomingEdges, "incomingEdge");
	}
	os << indent(indentFiveMore) << "outDegree = " << outDegree << endl;
	if (outDegree > 0)
	{
		printEdgeList(os, indentFiveMore, outgoingEdges, "outgoingEdge");
	}

	os << indent(indentFiveMore) << "inputsRequired = " << inputsRequired << endl;
	os << indent(indentFiveMore) << "outputsRequired = " << outputsRequired << endl;
	os << indent(indentFiveMore) << "weight = " << weight << endl;
	os << indent(indentFiveMore) << "gain = " << gain << endl;
	os << indent(indentFiveMore) << "valid = ";
	if (valid)
	{
		os << "TRUE";
	}
	else
	{
		os << "FALSE";
	}
	os << endl;

	os << indent(indentLevel) << "}" << endl;
}

// these three templated functions are for use by MFC during various calls made by CList or other
// container classes
template <> void AFXAPI ConstructElements<Vertex>(Vertex* newVertices, int nCount)
{
```

```
    for (int i = 0; i < nCount; i++, newVertices++)
    {
        new(newVertices) Vertex;
    }
}

template <> void AFXAPI DestructElements<Vertex>(Vertex* newVertices, int nCount)
{
    for (int i = 0; i < nCount; i++, newVertices++)
    {
        delete(newVertices);
    }
}

template <> void AFXAPI SerializeElements<Vertex> (CArchive& ar, Vertex* newVertices, int nCount)
{
    for (int i = 0; i < nCount; i++, newVertices++)
    {
        newVertices->Serialize(ar);
    }
}
```

A.2.3 Edge.h

```
/*********************************************************************************************
**                                                                                          **
** Edge.h                                                                                   **
**                                                                                          **
** This class represents a single Edge which is designed to have two attached Vertexes     **
** and to be contained in a Graph for use in a SpatialPartitioner which uses Clusters as   **
** containers.                                                                              **
**                                                                                          **
*********************************************************************************************/
#ifndef EDGE_H
#define EDGE_H

#include <afxtempl.h>
#include <iostream.h>

class Edge;
class Vertex;

typedef CList<Edge*, Edge*> EdgeList;

class Edge : public CObject
{
	DECLARE_SERIAL(Edge)

	public:
		// Default ctor
		Edge(Vertex* source_ = NULL, UINT srcPort_ = 0U, Vertex* sink_ = NULL, UINT
snkPort_ = 0U,
			const CString& name_ = "", const CString& type_ = "", INT weight_ = 0);
		// copy ctor
		inline Edge(const Edge&);
		// Default dtor
		virtual ~Edge();

		inline Vertex* getSource() const;
		inline Vertex* getSink() const;
		inline UINT getSourcePort() const;
		inline UINT getSinkPort() const;
		inline const CString& getName() const;
		inline const CString& getType() const;
		inline INT getWeight() const;
		inline BOOL isValid() const;
```

```
                VOID printOn(ostream& os_ = cout, size_t indentLevel = 5) const;
                VOID Serialize(CArchive& ar_);

                inline Edge& operator=(const Edge& rvalue_);
                inline BOOL operator==(const Edge&  rvalue_) const;
                inline BOOL operator!=(const Edge& rvalue_) const;

        protected:
                BOOL valid;

        private:
                VOID deepCopy(const Edge& rvalue_);

                Vertex* source;
                UINT sourcePort;
                Vertex* sink;
                UINT sinkPort;
                CString name;
                CString type;
                INT weight;
};

// copy ctor
inline
Edge::Edge(const Edge& rvalue_) :
        source(NULL), sink(NULL)
{
        deepCopy(rvalue_);
}

inline
Vertex*
Edge::getSource() const
{
        return source;
}

inline
UINT
Edge::getSourcePort() const
{
        return sourcePort;
}

inline
Vertex*
Edge::getSink() const
```

```
{
        return sink;
}

inline
UINT
Edge::getSinkPort() const
{
        return sinkPort;
}

inline
const CString&
Edge::getName() const
{
        return name;
}

inline
const CString&
Edge::getType() const
{
        return type;
}

inline
INT
Edge::getWeight() const
{
        return weight;
}

inline
BOOL
Edge::isValid() const
{
        return valid;
}

inline
Edge&
Edge::operator=(const Edge& rvalue_)
{
        deepCopy(rvalue_);

        return *this;
}
```

```
inline
BOOL
Edge::operator==(const Edge& rvalue_) const
{
        return name == rvalue_.name;
}

inline
BOOL
Edge::operator!=(const Edge& rvalue_) const
{
        return !operator==(rvalue_);
}

#endif EDGE_H
```

A.2.4 Edge.cpp

```
/*********************************************************************************************
**                                                                                          **
** Edge.cpp                                                                                 **
**                                                                                          **
** This class represents a single Edge which is designed to have two attached Vertexes     **
** and to be contained in a Graph for use in a SpatialPartitioner which uses Clusters as   **
** containers.                                                                              **
**                                                                                          **
*********************************************************************************************/
#include "osindent.h"
#include "RTRDefs.h"
#include "Edge.h"
#include "Vertex.h"

IMPLEMENT_SERIAL(Edge, CObject, EDGE_ID)

// Default ctor
Edge::Edge(Vertex*  source_,  UINT  sourcePort_,  Vertex*  sink_,  UINT  sinkPort_,  const  CString&
name_,
                  const CString& type_, INT weight_) :
        source(source_), sourcePort(sourcePort_), sink(sink_), sinkPort(sinkPort_), name(name_),
        type(type_), weight(weight_), valid(FALSE)
{
        if (source)
        {
                source->addOutgoingEdge(this);
        }
        if (sink)
        {
                sink->addIncomingEdge(this);
        }

        valid = TRUE;
}

// Default dtor
Edge::~Edge()
{
}

// copies EVERYTHING
VOID
Edge::deepCopy(const Edge& rvalue_)
{
```

```
        if (&rvalue_ != this)
        {
                name = rvalue_.name;
                type = rvalue_.type;
                source = rvalue_.source;
                sourcePort = rvalue_.sourcePort;
                sink = rvalue_.sink;
                sinkPort = rvalue_.sinkPort;
                weight = rvalue_.weight;
                valid = rvalue_.valid;
        }
}

VOID
Edge::Serialize(CArchive& ar)
{
        CObject::Serialize(ar);

        if (ar.IsStoring())
        {
                ar << source;
                ar << sourcePort;
                ar << sink;
                ar << sinkPort;
                ar << name;
                ar << type;
                ar << weight;
                ar << valid;
        }
        else
        {
                ar >> source;
                ar >> sourcePort;
                ar >> sink;
                ar >> sinkPort;
                ar >> name;
                ar >> type;
                ar >> weight;
                ar >> valid;
        }
}

VOID
Edge::printOn(ostream& os, size_t indentLevel) const
{
        UINT indentFiveMore = indentLevel + 5;
```

```
        os << "Edge {" << endl;

        os << indent(indentFiveMore) << "name = " << name << endl;
        os << indent(indentFiveMore) << "type = " << type << endl;
        os << indent(indentFiveMore) << "source = " << source->getName() << endl;
        os << indent(indentFiveMore) << "sourcePort = " << sourcePort << endl;
        os << indent(indentFiveMore) << "sink = " << sink->getName() << endl;
        os << indent(indentFiveMore) << "sinkPort = " << sinkPort << endl;
        os << indent(indentFiveMore) << "weight = " << weight << endl;
        os << indent(indentFiveMore) << "valid = ";
        if (valid)
        {
                os << "TRUE";
        }
        else
        {
                os << "FALSE";
        }
        os << endl;

        os << indent(indentLevel) << "}" << endl;
}

// these three templated functions are for use by MFC during various calls made by CList or other
// container classes
template <> void AFXAPI ConstructElements<Edge>(Edge* newEdges, int nCount)
{
    for (int i = 0; i < nCount; i++, newEdges++)
    {
        new(newEdges) Edge;
    }
}

template <> void AFXAPI DestructElements<Edge>(Edge* newEdges, int nCount)
{
    for (int i = 0; i < nCount; i++, newEdges++)
    {
        delete(newEdges);
    }
}

template <> void AFXAPI SerializeElements<Edge> (CArchive& ar, Edge* newEdges, int nCount)
{
    for (int i = 0; i < nCount; i++, newEdges++)
    {
        newEdges->Serialize(ar);
    }
```

```
}
```

A.2.5 Graph.h

```
/*********************************************************************************************
**                                                                                          **
** Graph.h                                                                                  **
**                                                                                          **
** This class represents a Graph which is designed to contain Edges and Vertexes and to    **
** used in a SpatialPartitioner which uses Clusters as containers.                          **
**                                                                                          **
*********************************************************************************************/
#ifndef GRAPH_H
#define GRAPH_H

#include <afx.h>
#include <iostream.h>

#include "Vertex.h"

class Edge;

class Graph : public CObject
{
        DECLARE_SERIAL(Graph)

        public:
                // Default ctor
                Graph(const CString& name_ = "");
                // Copy ctor
                inline Graph(const Graph& rvalue_);
                // Default dtor
                virtual ~Graph();

                BOOL addVertex(Vertex* v_);
                BOOL addEdge(Edge* e_);

                VOID clearEdgeList();
                VOID clearVertexList();

                inline double getAverageVertexWeight() const;
                inline UINT getOrder() const;
                inline UINT getSize() const;
                inline VertexList* getVertices() const;
                Vertex* getVertexByName(const CString& name_);
                Edge* getEdgeByName(const CString& name_);

                inline BOOL isValid() const;
```

```
		VOID Serialize(CArchive& ar_);

		inline Graph& operator=(const Graph& rvalue_);
		inline BOOL operator==(const Graph& rvalue_);
		inline BOOL operator!=(const Graph& rvalue_);

		VOID printOn(ostream& os_ = cout, size_t indentLevel = 0) const;

		// Reads the graph from the specified data file
		VOID fromFile(const CString& filename_);

	protected:
		BOOL valid;

	private:
		VOID deepCopy(const Graph& rvalue_);
		VOID printEdgeList(ostream& os_, size_t indentFiveMore_, const EdgeList* list_,
			const CString& name_) const;
		VOID  printVertexList(ostream&  os_,  size_t  indentFiveMore_,  const  VertexList*
list_,
			const CString& name_) const;

		CString name;
		EdgeList* edges;
		double avgVertexWeight;
		UINT order;
		UINT size;
		VertexList* vertices;
};

// Copy ctor
inline
Graph::Graph(const Graph& rvalue_) :
	edges(NULL), vertices(NULL)
{
	deepCopy(rvalue_);
}

inline
UINT
Graph::getOrder() const
{
	return order;
}

inline
```

```
UINT
Graph::getSize() const
{
	return size;
}

inline
double
Graph::getAverageVertexWeight() const
{
	return avgVertexWeight;
}

inline
VertexList*
Graph::getVertices() const
{
	return vertices;
}

inline
Graph&
Graph::operator=(const Graph& rvalue_)
{
	deepCopy(rvalue_);

	return *this;
}

inline
BOOL
Graph::operator==(const Graph& rvalue_)
{
	return name == rvalue_.name;
}

inline
BOOL
Graph::operator!=(const Graph& rvalue_)
{
	return !operator==(rvalue_);
}

#endif GRAPH_H
```

A.2.6 Graph.cpp

```
/*********************************************************************************************
**                                                                                          **
** Graph.cpp                                                                                **
**                                                                                          **
** This class represents a Graph which is designed to contain Edges and Vertexes and to    **
** used in a SpatialPartitioner which uses Clusters as containers.                          **
**                                                                                          **
*********************************************************************************************/
#include "Graph.h"
#include "osindent.h"
#include "RTRDefs.h"

IMPLEMENT_SERIAL(Graph, CObject, GRAPH_ID)

// Default ctor
Graph::Graph(const CString& name_) : name(name_), order(0), size(0), edges(NULL), vertices(NULL),
        avgVertexWeight(0), valid(FALSE)
{
        edges = new EdgeList;
        vertices = new VertexList;

        valid = !name.IsEmpty();
}

// Default dtor
Graph::~Graph()
{
        edges->RemoveAll();
        delete edges;
        vertices->RemoveAll();
        delete vertices;
}

BOOL
Graph::addEdge(Edge* e_)
{
        BOOL retval = FALSE;

        if (!getEdgeByName(e_->getName()))
        {
                if (getVertexByName(e_->getSource()->getName())
                        && getVertexByName(e_->getSink()->getName()))
                {
                        edges->AddTail(e_);
```

```
                        size++;
                        retval = TRUE;
                }
        }

        return retval;
}

BOOL
Graph::addVertex(Vertex* v_)
{
        BOOL retval = FALSE;

        if (!getVertexByName(v_->getName()))
        {
                vertices->AddTail(v_);
                if (order)
                {
                        avgVertexWeight = (avgVertexWeight * order + v_->getWeight()) / (order +
1);
                }
                else
                {
                        avgVertexWeight = v_->getWeight();
                }
                order++;
                retval = TRUE;
        }

        return retval;
}

// copies EVERYTHING
VOID
Graph::deepCopy(const Graph& rvalue_)
{
        if (&rvalue_ != this)
        {
                name = rvalue_.name;
                edges->RemoveAll();
                edges->AddHead((EdgeList*)(rvalue_.edges)); // have to cast away const
                order = rvalue_.order;
                avgVertexWeight = rvalue_.avgVertexWeight;
                size = rvalue_.size;
                vertices->RemoveAll();
                vertices->AddHead((VertexList*)(rvalue_.verti ces)); // have to cast away const
        }
```

```
}

VOID
Graph::Serialize(CArchive& ar_)
{
	edges->Serialize(ar_);
	vertices->Serialize(ar_);

	if (ar_.IsStoring())
	{
		ar_ << name;
		ar_ << avgVertexWeight;
		ar_ << order;
		ar_ << size;
	}
	else
	{
		ar_ >> name;
		ar_ >> avgVertexWeight;
		ar_ >> order;
		ar_ >> size;
	}
}

VOID
Graph::printVertexList(ostream& os, size_t indentLevel, const VertexList* list,
                                       const CString& listName) const
{
	if (list->GetCount())
	{
		Vertex* v;
		Vertex* tail = list->GetTail();
		POSITION pos = list->GetHeadPosition();
		UINT idx = 0;

		do
		{
			v = list->GetNext(pos);
			os << indent(indentLevel) << listName << "[" << idx++ << "] = ";
			v->printOn(os, indentLevel);
		} while ((v != tail) && (pos != NULL));
	}
}

VOID
Graph::printEdgeList(ostream& os, size_t indentLevel, const EdgeList* list,
                                     const CString& listName) const
```

```
{
	if (list->GetCount())
	{
		Edge* e;
		Edge* tail = list->GetTail();
		POSITION pos = list->GetHeadPosition();
		UINT idx = 0;

		do
		{
			e = list->GetNext(pos);
			os << indent(indentLevel) << listName << "[" << idx++ << "] = ";
			e->printOn(os, indentLevel);
		} while ((e != tail) && (pos != NULL));
	}
}

VOID
Graph::printOn(ostream& os, size_t indentLevel) const
{
	UINT indentFiveMore = indentLevel + 5;

	os << "Graph {" << endl;

	os << indent(indentFiveMore) << "name = " << name << endl;
	os << indent(indentFiveMore) << "order = " << order << endl;
	os << indent(indentFiveMore) << "avgVertexWeight = " << avgVertexWeight << endl;
	if (order > 0)
	{
		printVertexList(os, indentFiveMore, vertices, "vertex");
	}
	os << indent(indentFiveMore) << "size = " << size << endl;
	if (size > 0)
	{
		printEdgeList(os, indentFiveMore, edges, "edge");
	}
	os << indent(indentFiveMore) << "valid = ";
	if (valid)
	{
		os << "TRUE";
	}
	else
	{
		os << "FALSE";
	}
	os << endl;
```

```
	os << indent(indentLevel) << "}" << endl;
}

Vertex*
Graph::getVertexByName(const CString& name_)
{
	Vertex* retval = NULL;

	if (vertices->GetCount())
	{
		Vertex* v;
		Vertex* tail = vertices->GetTail();
		POSITION pos = vertices->GetHeadPosition();
		UINT idx = 0;

		do
		{
			v = vertices->GetNext(pos);
			if (v->getName() == name_)
			{
				retval = v;
			}
		} while (((v != tail) && (pos != NULL)) && (!retval));
	}

	return retval;
}

Edge*
Graph::getEdgeByName(const CString& name_)
{
	Edge* retval = NULL;

	if (edges->GetCount())
	{
		Edge* e;
		Edge* tail = edges->GetTail();
		POSITION pos = edges->GetHeadPosition();
		UINT idx = 0;

		do
		{
			e = edges->GetNext(pos);
			if (e->getName() == name_)
			{
				retval = e;
			}
```

```
                } while (((e != tail) && (pos != NULL)) && (!retval));
        }

        return retval;
}

// The name of the graph (enclosed in "s is the first thing not ignored in the file)
// after the name, the reader scans for the rest of the info (enclosed in {})
// Vertices are listed first and they look like:
// Name:Type:Weight:Inputs Required:Outputs Required
// They are ,-separated and the last one has a ; after it.
// Edges come next and they look like:
// Name:Source:Source Port:Sink:Sink Port:Type:weight
// They are ,-separated and the last one has a ; after it.
VOID
Graph::fromFile(const CString& filename)
{
        const size_t bufSize = 1024;

        enum StateType { GETTING_GRAPH_NAME = 0, GETTING_VERTICES = 1, GETTING_EDGES = 2,
                FINISHING = 3 };

        CFile file;
        CHAR buffer;
        StateType state = GETTING_GRAPH_NAME;
        CHAR string[bufSize];
        INT strIdx = 0;
        CString graphName;

        file.Open(filename, CFile::modeRead);

        // Read the graph name
        while (buffer != '"')
        {
                file.Read(&buffer, 1);
        }

        file.Read(&buffer, 1);

        do
        {
                string[strIdx++] = buffer;
                file.Read(&buffer, 1);
        } while (buffer != '"');

        graphName = CString(string, strIdx);
```

```
// go to the first {
while (buffer != '{')
{
	file.Read(&buffer, 1);
}

file.Read(&buffer, 1);

// read all of the vertices
while (buffer != ';')
{
	// skip leading whitespace
	while (buffer == '\t' || buffer == '\f' || buffer == '\r' || buffer == '\n'
		|| buffer == ' ')
	{
		file.Read(&buffer, 1);
	}

	strIdx = 0;
	while (buffer != ':')
	{
		string[strIdx++] = buffer;
		file.Read(&buffer, 1);
	}

	CHAR* vertexName = new CHAR[strIdx + 1];
	strncpy(vertexName, string, strIdx);
	vertexName[strIdx] = '\0';

	file.Read(&buffer, 1);

	strIdx = 0;
	while (buffer != ':')
	{
		string[strIdx++] = buffer;
		file.Read(&buffer, 1);
	}

	CHAR* vertexType = new CHAR[strIdx + 1];
	strncpy(vertexType, string, strIdx);
	vertexType[strIdx] = '\0';

	file.Read(&buffer, 1);

	UINT clbs = 0;
	while (buffer != ':')
	{
```

```
                clbs = clbs * 10 + buffer - '0';
                file.Read(&buffer, 1);
        }

        file.Read(&buffer, 1);

        UINT ins = 0;
        while (buffer != ':')
        {
                ins = ins * 10 + buffer - '0';
                file.Read(&buffer, 1);
        }

        file.Read(&buffer, 1);

        UINT outs = 0;
        while ((buffer != ';') && (buffer != ','))
        {
                outs = outs * 10 + buffer - '0';
                file.Read(&buffer, 1);
        }

        if (buffer == ','
        {
                file.Read(&buffer, 1);
        }

        addVertex(new Vertex(vertexName, vertexType, clbs, ins, outs));
}

file.Read(&buffer, 1);

// read all of the edges
while (buffer != ';')
{
        // skip leading whitespace
        while (buffer == '\t' || buffer == '\f' || buffer == '\r' || buffer == '\n'
                || buffer == ' ')
        {
                file.Read(&buffer, 1);
        }

        strIdx = 0;
        while (buffer != ':')
        {
                string[strIdx++] = buffer;
                file.Read(&buffer, 1);
```

```
}

CHAR* edgeName = new CHAR[strIdx + 1];
strncpy(edgeName, string, strIdx);
edgeName[strIdx] = '\0';

file.Read(&buffer, 1);

strIdx = 0;
while (buffer != ':')
{
        string[strIdx++] = buffer;
        file.Read(&buffer, 1);
}

CHAR* sourceName = new CHAR[strIdx + 1];
strncpy(sourceName, string, strIdx);
sourceName[strIdx] = '\0';
Vertex* source = getVertexByName(sourceName);

file.Read(&buffer, 1);

UINT sourcePort = 0;
while (buffer != ':')
{
        sourcePort = sourcePort * 10 + buffer - '0';
        file.Read(&buffer, 1);
}

file.Read(&buffer, 1);

strIdx = 0;
while (buffer != ':')
{
        string[strIdx++] = buffer;
        file.Read(&buffer, 1);
}

CHAR* sinkName = new CHAR[strIdx + 1];
strncpy(sinkName, string, strIdx);
sinkName[strIdx] = '\0';
Vertex* sink = getVertexByName(sinkName);

file.Read(&buffer, 1);

UINT sinkPort = 0;
while (buffer != ':')
```

```
                {
                        sinkPort = sinkPort * 10 + buffer - '0';
                        file.Read(&buffer, 1);
                }

                file.Read(&buffer, 1);

                strIdx = 0;
                while (buffer != ':')
                {
                        string[strIdx++] = buffer;
                        file.Read(&buffer, 1);
                }

                CHAR* edgeType = new CHAR[strIdx + 1];
                strncpy(edgeType, string, strIdx);
                edgeType[strIdx] = '\0';

                file.Read(&buffer, 1);

                UINT size = 0;
                while ((buffer != ';') && (buffer != ','))
                {
                        size = size * 10 + buffer - '0';
                        file.Read(&buffer, 1);
                }

                if (buffer == ','
                {
                        file.Read(&buffer, 1);
                }

                if (source && sink)
                {
                        addEdge(new Edge(source, sourcePort, sink, sinkPort, edgeName, edgeType,
size));
                }
        }

        // There's nothing important after the edges, so just bail out
        file.Close();
}

// these three templated functions are for use by MFC during various calls made by CList or other
// container classes
template <> void AFXAPI ConstructElements<Graph>(Graph* newGraphs, int nCount)
{
```

```
    for (int i = 0; i < nCount; i++, newGraphs++)
    {
        new(newGraphs) Graph;
    }
}

template <> void AFXAPI DestructElements<Graph>(Graph* newGraphs, int nCount)
{
    for (int i = 0; i < nCount; i++, newGraphs++)
    {
        delete(newGraphs);
    }
}

template <> void AFXAPI SerializeElements<Graph> (CArchive& ar, Graph* newGraphs, int nCount)
{
    for (int i = 0; i < nCount; i++, newGraphs++)
    {
        newGraphs->Serialize(ar);
    }
}
```

A.3 The Spatial Partitioner Library

A.3.1 Cluster.h

```
/*********************************************************************************************
**                                                                                          **
** Cluster.h                                                                                **
**                                                                                          **
** This class represents a Cluster which is designed to contain Vertexes and to be used   **
** in a SpatialPartitioner as the semiatomic structures which represent one processing    **
** element.
**                                                                                          **
*********************************************************************************************/
#ifndef CLUSTER_H
#define CLUSTER_H

#include <afx.h>
#include <afxtempl.h>
#include <iostream.h>

#include "Vertex.h"

class Cluster;

typedef CList<Cluster*, Cluster*> ClusterList;
typedef CList<UINT, UINT> UINTList;

class Cluster : public CObject
{
        DECLARE_SERIAL(Cluster);

        // Constant used in the closeness calculation
        enum {ALPHA = 60};

        public:
                // Default ctor
                Cluster(VertexList* vertices_ = NULL, UINT number_ = 0U, UINT clbs_ = 0U,
                        UINT inPins_ = 0U, UINT outPins_ = 0U, UINT biPins_ = 0U,
                        UINTList* neighbors_ = NULL);
                // ctor which takes a single vertex instead of a VertexList
                Cluster(Vertex* v_, UINT number_ = 0U, UINT clbs_ = 0U, UINT inPins_ = 0U,
                        UINT outPins_ = 0U, UINT biPins_ = 0U, UINTList* neighbors_ = NULL);
                // Copy ctor
                inline Cluster(const Cluster& rvalue_);
                // Default dtor
                virtual ~Cluster();
```

```
        VOID addVertex(Vertex* v);
        VOID removeVertex(Vertex* v);
        VOID addVertexList(VertexList* list);

        VOID findCandidate(ClusterList* clustersLeft, POSITION pos, double avgCellSize);
        inline INT getCloseness() const;
        inline Cluster* getClosestTo() const;
        inline UINT getNumber() const;
        inline VertexList* getVertices() const;
        // Returns the number of interconnections between <this> and d.
        UINT isConnected(const Cluster* d) const;
        inline UINT getSize() const;
        // Returns what the size would be if v were added
        inline UINT getPotentialSize(const Vertex* v) const;
        inline BOOL contains(const Vertex* v) const;
        inline VOID setCloseness(INT closeness_);
        inline VOID setClosestTo(Cluster* closestTo_);
        inline UINT getInPinsUsed() const;
        inline UINT getOutPinsUsed() const;
        inline UINT getBiPinsUsed() const;
        // Returns TRUE iff <this> has room to add v_
        BOOL hasRoomFor(const Vertex* v_) const;
        // Returns TRUE iff <this> has room to add every vertex in c_
        BOOL hasRoomFor(const Cluster* c_) const;
        inline BOOL isValid() const;
        inline const UINTList* getNeighbors() const;

        inline Cluster& operator=(const Cluster& rvalue_);
        inline BOOL operator==(const Cluster& rvalue_);
        inline BOOL operator!=(const Cluster& rvalue_);

        VOID Serialize(CArchive& ar_);

        VOID printVertexList(ostream& os_ = cout, size_t indentLevel = 0) c onst;
        VOID printNeighborList(ostream& os_ = cout, size_t indentLevel = 0) const;
        VOID printOn(ostream& os_ = cout, size_t indentLevel = 0) const;

protected:
        BOOL valid;

private:
        INT closenessTo(const Cluster* d, UINT numConns, double avgCellSize) const;
        UINT countNets();
        inline UINT getNumNets() const;
        UINT getListSize(const VertexList* list) const;
        VOID deepCopy(const Cluster& rvalue_);
```

```
                UINT number;
                UINT clbs;
                UINT inPins;
                UINT inPinsUsed;
                UINT outPins;
                UINT outPinsUsed;
                UINT biPins;
                UINT biPinsUsed;
                UINTList* neighbors;

                BOOL dirty;
                VertexList* vertices;
                Cluster* closestTo;
                INT closeness;
                UINT numConnections;
                UINT numNets;
                UINT size;
};

// Copy ctor
inline
Cluster::Cluster(const Cluster& rvalue_) :
        neighbors(NULL), vertices(NULL), closestTo(NULL)
{
        deepCopy(rvalue_);
}

inline
BOOL
Cluster::contains(const Vertex* v) const
{
        return vertices->Find((Vertex*)v) != NULL;
}

inline
INT
Cluster::getCloseness() const
{
        return closeness;
}

inline
const UINTList*
Cluster::getNeighbors() const
{
        return neighbors;
```

```
}

inline
UINT
Cluster::getNumber() const
{
        return number;
}

inline
VOID
Cluster::setCloseness(INT closeness_)
{
        closeness = closeness_;
}

inline
VOID
Cluster::setClosestTo(Cluster* closestTo_)
{
        closestTo = closestTo_;
}

inline
Cluster*
Cluster::getClosestTo() const
{
        return closestTo;
}

inline
UINT
Cluster::getInPinsUsed() const
{
        return inPinsUsed;
}

inline
UINT
Cluster::getOutPinsUsed() const
{
        return outPinsUsed;
}

inline
UINT
Cluster::getBiPinsUsed() const
```

```
{
        return biPinsUsed;
}

inline
UINT
Cluster::getNumNets() const
{
        return numNets;
}

inline
UINT
Cluster::getSize() const
{
        return size;
}

// Returns what the size would be if v were added
inline
UINT
Cluster::getPotentialSize(const Vertex* v) const
{
        return size + v->getWeight();
}

inline
VertexList*
Cluster::getVertices() const
{
        return vertices;
}

inline
Cluster::isValid() const
{
        return valid;
}

inline
Cluster&
Cluster::operator=(const Cluster& rvalue_)
{
        deepCopy(rvalue_);

        return *this;
}
```

```
inline
BOOL
Cluster::operator==(const Cluster& rvalue_)
{
        return number == rvalue_.number;
}

inline
BOOL
Cluster::operator!=(const Cluster& rvalue_)
{
        return !operator==(rvalue_);
}

#endif CLUSTER_H
```

A.3.2 Cluster.cpp

```
/*********************************************************************************************
**                                                                                          **
** Cluster.cpp                                                                              **
**                                                                                          **
** This class represents a Cluster which is designed to contain Vertexes and to be used   **
** in a SpatialPartitioner as the semiatomic structures which represent one processing    **
** element.                                                                                 **
**                                                                                          **
*********************************************************************************************/
#include "Cluster.h"
#include "osindent.h"
#include "RTRDefs.h"

IMPLEMENT_SERIAL(Cluster, CObject, CLUSTER_ID)

// Default ctor
Cluster::Cluster(VertexList* vertices_, UINT number_, UINT clbs_, UINT inPins_, UINT outPins_,
                                UINT biPins_, UINTList* neighbors_) :
        number(number_), clbs(clbs_), inPins(inPins_), outPins(outPins_), biPins(biPins_),
        neighbors(neighbors_), vertices(vertices_), closestTo(NULL), closeness(INT_MIN),
        numConnections(0), numNets(0), size(0), dirty(TRUE), inPinsUsed(0), outPinsUsed(0),
        biPinsUsed(0), valid(TRUE)
{
        if (!vertices)
        {
                vertices = new VertexList;
        }

        if (!neighbors)
        {
                neighbors = new UINTList;
        }

        numNets = countNets();
        size = getListSize(vertices);
}

// ctor which takes a single vertex as opposed to a VertexList
Cluster::Cluster(Vertex* v, UINT number_, UINT clbs_, UINT inPins_, UINT outPins_, UINT biPins_,
                                UINTList* neighbors_) :
        number(number_), clbs(clbs_), inPins(inPins_), outPins(outPins_), biPins(biPins_),
        neighbors(neighbors_),      vertices(NULL),      closestTo(NULL),      closeness(INT_MIN),
numConnections(0),
```

```
	numNets(0),	size(0),	dirty(TRUE),	inPinsUsed(0),	outPinsUsed(0),	biPinsUsed(0),
valid(TRUE)
{
	if (!neighbors)
	{
		neighbors = new UINTList;
	}

	vertices = new VertexList;
	vertices->AddHead(v);

	numNets = countNets();
	size = v->getWeight();
}

Cluster::~Cluster()
{
	delete neighbors;
	delete vertices;
}

UINT
Cluster::countNets()
{
	UINT count = 0;

	if (vertices->GetCount())
	{
		Vertex* v;
		Vertex* tail = vertices->GetTail();
		POSITION pos = vertices->GetHeadPosition();

		do
		{
			v = vertices->GetNext(pos);

			EdgeList* list = v->getIncomingEdgeList();
			if (list->GetCount())
			{
				Edge* e;
				Edge* eTail = list->GetTail();
				POSITION ePos = list->GetHeadPosition();

				do
				{
					e = list->GetNext(ePos);
					if (vertices->Find(e->getSource()) == NULL)
```

```
                                {
                                        count++;
                                }
                        } while ((e != eTail) && (ePos != NULL));
                }

                list = v->getOutgoingEdgeList();
                if (list->GetCount())
                {
                        Edge* e;
                        Edge* eTail = list->GetTail( );
                        POSITION ePos = list->GetHeadPosition();

                        do
                        {
                                e = list->GetNext(ePos);
                                if (vertices->Find(e->getSink()) == NULL)
                                {
                                        count++;
                                }
                        } while ((e != eTail) && (ePos != NULL));
                }

        } while ((v != tail) && (pos != NULL));
    }

    return count;
}

VOID
Cluster::addVertex(Vertex* v)
{
    UINT inPinsAvailable = inPins - inPinsUsed;
    UINT inputsTaken = inPinsAvailable > v->getInputsRequired() ? v->getInputsRequired()
            : inPinsAvailable;
    UINT inPinsStillNeeded = v->getInputs Required() - inputsTaken;
    inPinsUsed += inputsTaken;

    UINT outPinsAvailable = outPins - outPinsUsed;
    UINT outputsTaken = outPinsAvailable > v->getOutputsRequired() ? v->getOutputsRequired()
            : outPinsAvailable;
    UINT outPinsStillNeeded = v->getOutputsRequired() - outputsTaken;
    outPinsUsed += outputsTaken;

    UINT biPinsTaken = inPinsStillNeeded + outPinsStillNeeded;
    biPinsUsed += biPinsTaken;
```

```
        vertices->AddTail(v);
        v->setAdded(TRUE);
        v->setClusterNumber(number);
        numNets = countNets();
        size += v->getWeight();
        dirty = TRUE;
}

VOID
Cluster::addVertexList(VertexList* list)
{
        if (list->GetCount())
        {
                Vertex* v;
                Vertex* tail = list->GetTail();
                POSITION pos = list->GetHeadPosition();

                do
                {
                        v = list->GetNext(pos);
                        addVertex(v);
                } while ((v != tail) && (pos != NULL));

                dirty = TRUE;
        }
}

VOID
Cluster::removeVertex(Vertex* v)
{
        INT diff = inPinsUsed - v->getInputsRequired();
        UINT pinsFreed = diff > 0 ? diff : inPinsUsed;

        if (pinsFreed > inPinsUsed)
        {
                inPinsUsed = 0;
                diff = pinsFreed - inPinsUsed;
                biPinsUsed -= diff;
        }
        else
        {
                inPinsUsed -= pinsFreed;
        }

        diff = outPinsUsed - v->getOutputsRequired();
        pinsFreed = diff > 0 ? diff : outPinsUsed;
```

```
        if (pinsFreed > outPinsUsed)
        {
                outPinsUsed = 0;
                diff = pinsFreed - outPinsUsed;
                outPinsUsed -= diff;
        }
        else
        {
                outPinsUsed -= pinsFreed;
        }

        vertices->RemoveAt(vertices->Find(v));
        v->setAdded(FALSE);
        v->setClusterNumber(0);
        numNets = countNets();
        size -= v->getWeight();
        dirty = TRUE;
}

// Returns the number of interconnections between <this> and d
UINT
Cluster::isConnected(const Cluster* d) const
{
        UINT retval = 0;

        if (vertices->GetCount())
        {
                Vertex* v;
                Vertex* tail = vertices->GetTail();
                POSITION pos = vertices->GetHeadPosition();
                VertexList* list = d->getVertices();

                do
                {
                        v = vertices->GetNext(pos);
                        Edge* e;
                        EdgeList* eList = v->getIncomingEdgeList();

                        if (eList->GetCount())
                        {
                                Edge* eTail = eList->GetTail();
                                POSITION ePos = eList->GetHeadPosition();

                                do
                                {
                                        e = eList->GetNext(ePos);
                                        if (d->contains(e->getSource()))
```

```
                                {
                                        retval++;
                                }
                        } while (e != eTail);
                }

                eList = v->getOutgoingEdgeList();
                if (eList->GetCount())
                {
                        Edge* eTail = eList->GetTail();
                        POSITION ePos = eList->GetHeadPosition();

                        do
                        {
                                e = eList->GetNext(ePos);
                                if (d->contains(e->getSink()))
                                {
                                        retval++;
                                }
                        } while (e != eTail);
                }

        } while ((v != tail) && (pos != NULL));
    }

    return retval;
}

UINT
Cluster::getListSize(const VertexList* list) const
{
    UINT listSize = 0;

    if (list->GetCount())
    {
        Vertex* v;
        Vertex* tail = vertices->GetTail();
        POSITION pos = vertices->GetHeadPosition();

        do
        {
                v = vertices->GetNext(pos);
                listSize += v->getWeight();
        } while ((v != tail) && (pos != NULL));
    }

    return listSize;
```

```
}

// Computes the closeness between <this> and d.
INT
Cluster::closenessTo(const Cluster* d, UINT numConns, double avgCellSize) const
{
	INT closenessValue = 0;

	if (d->getNeighbors()->Find(number) && avgCe llSize && numNets && d->getNumNets()
		&& (size <= d->getSize()) && hasRoomFor(d))
	{
		closenessValue = ALPHA * numConns / __min(numNets, d->getNumNets())
			- ((size + d->getSize()) / (INT)avgCellSize);
	}

	return closenessValue;
}

VOID
Cluster::findCandidate(ClusterList* clustersLeft, POSITION pos, double avgCellSize)
{
	if (dirty)
	{
		Cluster* d;
		Cluster* tail;

		do
		{
			d = clustersLeft->GetNext(pos);

			UINT numConns = isConnected(d);

			if (numConns > 0)
			{
				INT newCloseness = closen essTo(d, numConns, avgCellSize);

				if (newCloseness > closeness)
				{
					closeness = newCloseness;
					closestTo = d;
					closestTo->setCloseness(closeness);
					closestTo->setClosestTo(this);
				}
			}

		} while ((d != tail) && (pos != NULL));
```

```
                dirty = FALSE;
        }
}

VOID
Cluster::Serialize(CArchive& ar)
{
        CObject::Serialize(ar);

        vertices->Serialize(ar);
        neighbors->Serialize(ar);
        closestTo->Serialize(ar);

        if (ar.IsStoring())
        {
                ar << number;
                ar << clbs;
                ar << inPins;
                ar << inPinsUsed;
                ar << outPins;
                ar << outPinsUsed;
                ar << biPins;
                ar << biPinsUsed;
                ar << closeness;
                ar << dirty;
                ar << numNets;
                ar << size;
                ar << valid;
        }
        else
        {
                UINT theType;

                ar >> number;
                ar >> clbs;
                ar >> inPins;
                ar >> inPinsUsed;
                ar >> outPins;
                ar >> outPinsUsed;
                ar >> biPins;
                ar >> biPinsUsed;
                ar >> closeness;
                ar >> dirty;
                ar >> numNets;
                ar >> size;
                ar >> theType;
                ar >> valid;
```

```
	}
}

// Copies EVERYTHING
VOID
Cluster::deepCopy(const Cluster& rvalue_)
{
	if (this != &rvalue_)
	{
		number = rvalue_.number;
		clbs = rvalue_.clbs;
		inPins = rvalue_.inPins;
		inPinsUsed = rvalue_.inPinsUsed;
		outPins = rvalue_.outPins;
		outPinsUsed = rvalue_.outPinsUsed;
		biPins = rvalue_.biPins;
		biPinsUsed = rvalue_.biPinsUsed;
		closestTo = rvalue_.closestTo;
		closeness = rvalue_.closeness;
		numNets = rvalue_.numNets;
		dirty = rvalue_.dirty;
		size = rvalue_.size;

		delete vertices;
		vertices = new VertexList;
		vertices->AddHead((VertexList*)(rvalue_.vertices)); // have to  cast away const

		delete neighbors;
		neighbors = new UINTList;
		neighbors->AddHead((UINTList*)(rvalue_.neighbors)); // have to cast away const

		valid = rvalue_.valid;
	}
}

VOID
Cluster::printVertexList(ostream& os, size_t indentLevel) const
{
	if (vertices->GetCount())
	{
		Vertex* v;
		Vertex* tail = vertices->GetTail();
		POSITION pos = vertices->GetHeadPosition();
		UINT idx = 0;

		do
		{
```

```
                        v = vertices->GetNext(pos);
                        os << indent(indentLevel) << "vertex[" << idx++ << "] = ";
                        v->printOn(os, indentLevel);
                } while ((v != tail) && (pos != NULL));
        }
}

VOID
Cluster::printNeighborList(ostream& os, size_t indentLevel) const
{
        if (neighbors->GetCount())
        {
                UINT i;
                UINT tail = neighbors->GetTail();
                POSITION pos = neighbors->GetHeadPosition();
                UINT idx = 0;

                do
                {
                        i = neighbors->GetNext(pos);
                        os << indent(indentLevel) << "neighbor[" << idx++ << "] = " << i << endl;
                } while ((i != tail) && (pos != NULL));
        }
}

VOID
Cluster::printOn(ostream& os, size_t indentLevel) const
{
        static printing = FALSE;

        size_t indentFiveMore = indentLevel + 5;

        os << "Cluster {" << endl;

        os << indent(indentFiveMore) << "number = " << number << endl;
        os << indent(indentFiveMore) << "clbs = " << clbs << endl;
        os << indent(indentFiveMore) << "inPins = " << inPins << endl;
        os << indent(indentFiveMore) << "inPinsUsed = " << inPinsUsed << endl;
        os << indent(indentFiveMore) << "outPins = " << outPins << endl;
        os << indent(indentFiveMore) << "outPinsUsed = " << outPinsUsed << endl;
        os << indent(indentFiveMore) << "biPins = " << biPins << endl;
        os << indent(indentFiveMore) << "biPinsUsed = " << biPinsUsed << endl;
        printNeighborList(os, indentFiveMore);

        printVertexList(os, indentFiveMore);

        if (!printing)
```

```
	{
		printing = TRUE;

		os << indent(indentFiveMore) << "closestTo = ";
		if (closestTo)
		{
			closestTo->printOn(os, indentFiveMore);
		}
		else
		{
			os << "NULL" << endl;
		}

		os << indent(indentFiveMore) << "closeness = " << closeness << endl;

		printing = FALSE;
	}

	os << indent(indentFiveMore) << "numNets = " << numNets << endl;
	os << indent(indentFiveMore) << "size = " << size << endl;
	os << indent(indentFiveMore) << "dirty = ";
	if (dirty)
	{
		os << "TRUE";
	}
	else
	{
		os << "FALSE";
	}
	os << endl;
	os << indent(indentFiveMore) << "valid = ";
	if (valid)
	{
		os << "TRUE";
	}
	else
	{
		os << "FALSE";
	}
	os << endl;

	os << indent(indentLevel) << "}" << endl;
}

// Returns true iff <this> has room for v.
BOOL
Cluster::hasRoomFor(const Vertex* v) const
```

```
{
	BOOL retval = FALSE;

	if (size + v->getWeight() <= clbs)
	{
		if (inPins - inPinsUsed >= v->getInputsRequired())
		{
			if (outPins - outPinsUsed >= v->getOutputsRequired())
			{
				retval = TRUE;
			}
			else if (outPins - outPinsUsed + biPins - biPinsUsed >= v-
>getOutputsRequired())
			{
				retval = TRUE;
			}
		}
		else if (inPins - inPinsUsed + biPins - biPinsUsed >= v->getInputsRequired())
		{
			UINT biPinsTaken = v->getInputsRequired() - (inPins - inPinsUsed);

			if (outPins - outPinsUsed >= v->getOutputsRequired())
			{
				retval = TRUE;
			}
			else if (outPins - outPinsUsed + biPins - biPinsUsed - biPinsTaken
				>= v->getOutputsRequired())
			{
				retval = TRUE;
			}
		}
	}

	return retval;
}

// Returns true iff <this> has room for all vertices in c.
BOOL
Cluster::hasRoomFor(const Cluster* c) const
{
	BOOL retval = FALSE;

	if (size + c->getSize() < clbs)
	{
		if (inPins - inPinsUsed >= c->getInPinsUsed())
		{
			if (outPins - outPinsUsed >= c->getOutPinsUsed())
```

```
                        {
                                if (biPins - biPinsUsed >= c->getBiPinsUs ed())
                                {
                                        retval = TRUE;
                                }
                        }
                        else   if   (outPins   -   outPinsUsed   +   biPins   -   biPinsUsed   >=   c-
>getOutPinsUsed())
                        {
                                UINT biPinsTaken = c->getOutPinsUsed() - (outPins - outPinsUsed);

                                if (biPins - biPinsUsed - biPinsTaken >= c->getBiPinsUsed())
                                {
                                        retval = TRUE;
                                }
                        }
                }
                else if (inPins - inPinsUsed + biPins - biPinsUsed >= c->getInPinsUsed())
                {
                        UINT biPinsTaken = c->getInPinsUsed() - (inPins - inPinsUsed);

                        if (outPins - outPinsUsed >= c->getOutPinsUsed())
                        {
                                if (biPins - biPinsUsed - biPinsTaken >= c->getBiPinsUsed())
                                {
                                        retval = TRUE;
                                }
                        }
                        else
                        {
                                biPinsTaken += c->getOutPinsUsed() - (outPins - outPinsUsed);

                                if (outPins - outPinsUsed + biPins - biPinsUsed - biPinsTaken >= c-
>getOutPinsUsed())
                                {
                                        if (biPins - biPinsUsed - biPinsTaken >= c->getBiPinsUsed())
                                        {
                                                retval = TRUE;
                                        }
                                }
                        }
                }
        }

        return retval;
}
```

```
// these three templated functions are for use by MFC during various calls made by CList or other
// container classes
template <> void AFXAPI ConstructElements<Cluster>(Cluster* newClusters, int nCount)
{
    for (int i = 0; i < nCount; i++, newClusters++)
    {
        new(newClusters) Cluster;
    }
}

template <> void AFXAPI DestructElements<Cluster>(Cluster* newClusters, int nCount)
{
    for (int i = 0; i < nCount; i++, newClusters++)
    {
        delete(newClusters);
    }
}

template <> void AFXAPI SerializeElements<Cluster> (CArchive& ar, Cluster* newClusters, int
nCount)
{
    for (int i = 0; i < nCount; i++, newClusters++)
    {
        newClusters->Serialize(ar);
    }
}
```

A.3.3 SpatialPartitioner.h

```
/*****************************************************************************************
**                                                                                      **
** SpatialPartitioner.h                                                                 **
**                                                                                      **
** This class represents a SpatialPartitioner whose job it is to determine the best way **
** to subdivide a Graph consiting of Vertices and Edges into a group of Clusters in the **
** shortest time possible.  It does so by simultaneously minimizing the number of PEs   **
** required (Clusters) and the number of interconnections between those PEs.  It finds  **
** the best solution possible in the time allowed and then saves the result.            **
**                                                                                      **
*****************************************************************************************/
#ifndef SPATIALPARTIONER_H
#define SPATIALPARTIONER_H

#include <afx.h>
#include <iostream.h>

#include "Cluster.h"
#include "Graph.h"

class SpatialPartitioner : public CObject
{
        DECLARE_SERIAL(SpatialPartitioner)

        public:
                // Default ctor
                SpatialPartitioner(const CString& filename_ = "", const Graph* g_ = NULL);
                // Copy ctor
                inline SpatialPartitioner(const SpatialPartitioner& rvalue_);
                // Default dtor
                virtual ~SpatialPartitioner();

                BOOL run();

                inline BOOL isValid() const;
                VOID Serialize(CArchive& ar_);

                inline SpatialPartitioner& operator=(const SpatialPartitioner& rvalue_);
                inline BOOL operator==(const SpatialPartitioner& rvalue_);
                inline BOOL operator!=(const SpatialPartitioner& rvalue_);

                VOID printClusterList(ostream& os = cout, size_t indentLevel = 0) const;
                VOID printTopology(ostream& os = cout, size_t indentLevel = 0) const;
                VOID printNonemptyPartitions(ostream& os = cout, size_t indentLevel = 0) const;
```

```
        VOID printOn(ostream& os_ = cout, size_t indentLevel = 0) const;
        UINT countInterconnections(const ClusterList* = NULL) const;
        inline UINT countPEs() const;

    protected:
        BOOL valid;

    private:
        static const CString outfile;

        UINT countNonemptyClusters(const ClusterList& list) const;
        // Saves the result to an output file
        VOID toFile() const;
        // Inserts a single vertex
        BOOL recurseInsertion(Vertex* inV, POSITION inPos);
        VOID initializeClusterList();
        BOOL insertVertices();
        // Returns TRUE iff it is legal to add v_ to toCluster_.
        BOOL legalToAdd(const Vertex* v_, const Cluster* toCluster_);
        // Calculates the gains associated with moving each vertex from fromCluster_ to
toCluster_
        VOID calculateGains(const Cluster* fromCluster, const Cluster* toCluster);
        VOID deepCopy(const SpatialPartitioner& rvalue_);
        VOID findCandidates();
        Cluster* findCluster(UINT clusterNumber_, ClusterList& clusterList_);
        Vertex* findVertexWithLargestGain(const Cluster* c);
        // the main partitioning method
        UINT gcep(ClusterList*);
        // called iterativel by gcep() to accomplish n-way partitioning
        VOID gcep_2way(Cluster* fromCluster, Cluster* toCluster);
        // calls gcep()
        BOOL hgcep();
        BOOL makeClusters(UINT numClusters);
        VOID mergeClosest();
        VOID outputResults();
        VOID parseTopologyFile();
        VOID refineClusters();
        VOID saveResult();
        VOID updateGain(Vertex* v, const Cluster* fromCluster, const Cluster* toCluster);

        ClusterList clusters;
        CString filename;
        const Graph* g;
        ClusterList* partition;
        ClusterList topology;
        UINT platformSize;
};
```

```
// Copy ctor
inline
SpatialPartitioner::SpatialPartitioner(const SpatialPartitioner& rvalue_) :
        g(NULL), partition(NULL)
{
        deepCopy(rvalue_);
}

inline
BOOL
SpatialPartitioner::isValid() const
{
        return valid;
}

inline
UINT
SpatialPartitioner::countPEs() const
{
        UINT retval = 0;

        if (partition)
        {
                retval = countNonemptyClusters(*partition);
        }

        return retval;
}

inline
SpatialPartitioner&
SpatialPartitioner::operator=(const SpatialPartitioner& rvalue_)
{
        deepCopy(rvalue_);

        return *this;
}

inline
BOOL
SpatialPartitioner::operator==(const SpatialPartitioner& rvalue_)
{
        return this == &rvalue_;
}

inline
```

```
BOOL
SpatialPartitioner::operator!=(const SpatialPartitioner& rvalue_)
{
        return !operator==(rvalue_);
}

#endif SPATIALPARTIONER_H
```

A.3.4 SpatialPartitioner.cpp

```
/*****************************************************************************************
**                                                                                      **
** SpatialPartitioner.h                                                                 **
**                                                                                      **
** This class represents a SpatialPartitioner whose job it is to determine the best way **
** to subdivide a Graph consiting of Vertices and Edges into a group of Clusters in the **
** shortest time possible.  It does so by simultaneously minimizing the number of PEs   **
** required (Clusters) and the number of interconnections between those PEs.  It finds  **
** the best solution possible in the time allowed and then saves the result.            **
**                                                                                      **
*****************************************************************************************/
#include <afxtempl.h>
#include <limits.h>

#include "osindent.h"
#include "RTRDefs.h"
#include "SpatialPartitioner.h"
#include "Vertex.h"

IMPLEMENT_SERIAL(SpatialPartitioner, CObject, SPATIALPARTITIONER_ID)

const CString SpatialPartitioner::outfile = "clusters.file";

// Default ctor
SpatialPartitioner::SpatialPartitioner(const CString& filename_, const Graph* g_) :
        g(g_), filename(filename_), partition(NULL), platformSize(0U), valid(FALSE)
{
        parseTopologyFile();
}

// Default dtor
SpatialPartitioner::~SpatialPartitioner()
{
        delete partition;
}

// Topology file looks like:
// PE #:# CLBs:# of Input Only Pins:# of Output Only Pins:# of Bidirectional Pins:neighbor 1,2,3;
VOID
SpatialPartitioner::parseTopologyFile()
{
        CFile file(filename, CFile::modeRead);
        size_t bufSize = 1024;
        CHAR* buffer = new CHAR[bufSize];
```

```
	UINT bytesRead = file.Read(buffer, bufSize);
	enum { GETTING_PE_NUM = 0, GETTING_CLBS = 1, GETTING_IN_PINS = 2, GETTING_OUT_PINS = 4,
		GETTING_BI_PINS = 5, GETTING_NEIGHBORS = 6 };
	UINT state = GETTING_PE_NUM;
	INT value;
	CString valString;

	if (bytesRead)
	{
		do
		{
			CString name;
			UINT peNumber;
			UINT clbs;
			UINT inPins;
			UINT outPins;
			UINT biPins;
			UINTList* neighbors = new UINTList;

			for (UINT x = 0; x < bytesRead; x++)
			{
				UINT strLen = 0;

				while (!(buffer[x] == '0' || buffer[x] == '1' || buffer[x] == '2'
|| buffer[x] == '3'
					|| buffer[x] == '4' || buffer[x] == '5' || buffer[x] ==
'6'|| buffer[x] == '7'
					|| buffer[x] == '8' || buffer[x] == '9'))
				{
					x++;
				}

				while (buffer[x] == '0' || buffer[x] == '1' || buffer[x] == '2' ||
buffer[x] == '3'
					|| buffer[x] == '4' || buffer[x] == '5' || buffer[x] ==
'6'|| buffer[x] == '7'
					|| buffer[x] == '8' || buffer[x] == '9')
				{
					x++;
					strLen++;
				}
				valString = CString(buffer + x - strLen, strLen);
				value = atoi(valString.GetBuffer(0));

				switch (state)
				{
					case GETTING_PE_NUM:
```

```
                                        name = valString;
                                        peNumber = value;
                                        state = GETTING_CLBS;
                                        break;

                                case GETTING_CLBS:
                                        clbs = value;
                                        state = GETTING_IN_PINS;
                                        break;

                                case GETTING_IN_PINS:
                                        inPins = value;
                                        state = GETTING_OUT_PINS;
                                        break;

                                case GETTING_OUT_PINS:
                                        outPins = value;
                                        state = GETTING_BI_PINS;
                                        break;

                                case GETTING_BI_PINS:
                                        biPins = value;
                                        state = GETTING_NEIGHBORS;
                                        break;

                                case GETTING_NEIGHBORS:
                                        neighbors->AddTail(value);
                                        if (buffer[x] == ';')
                                        {
                                                topology.AddTail(new Cluster(new Vertex(name,
"Topology", 0),
                                                        peNumber,   clbs,   inPins,   outPins,
biPins, neighbors));
                                                neighbors = new UINTList;
                                                state = GETTING_PE_NUM;
                                                platformSize += clbs;
                                        }
                                        break;
                        }
                }
        } while ((bytesRead = file.Read(buffer, bufSize)) == bufSize);
    }

    delete [] buffer;
}

VOID
```

```
SpatialPartitioner::printClusterList(ostream& os, size_t indentLevel) const
{
	if (clusters.GetCount())
	{
		Cluster* c;
		Cluster* tail = clusters.GetTail();
		POSITION pos = clusters.GetHeadPosition();
		UINT idx = 0;

		do
		{
			c = clusters.GetNext(pos);
			os << indent(indentLevel) << "cluster[" << idx++ << "] = ";
			c->printOn(os, indentLevel);
		} while ((c != tail) && (pos != NULL));
	}
}

VOID
SpatialPartitioner::printTopology(ostream& os, size_t indentLevel) const
{
	if (topology.GetCount())
	{
		Cluster* c;
		Cluster* tail = topology.GetTail();
		POSITION pos = topology.GetHeadPosition();
		UINT idx = 0;

		do
		{
			c = topology.GetNext(pos);
			os << indent(indentLevel) << "topology[" << idx++ << "] = ";
			c->printOn(os, indentLevel);
		} while ((c != tail) && (pos != NULL));
	}
}

VOID
SpatialPartitioner::printNonemptyPartitions(ostream& os, size_t indentLevel) const
{
	if (partition && partition->GetCount())
	{
		Cluster* c;
		Cluster* tail = partition->GetTail();
		POSITION pos = partition->GetHeadPosition();
		UINT idx = 0;
```

```
                do
                {
                        c = partition->GetNext(pos);
                        if (c->getSize())
                        {
                                os << indent(indentLevel) << "partition[" << idx++ << "] = ";
                                c->printOn(os, indentLevel);
                        }
                } while ((c != tail) && (pos != NULL));
        }
}

VOID
SpatialPartitioner::findCandidates()
{
        if (clusters.GetCount())
        {
                Cluster* c;
                Cluster* tail = clusters.GetTail();
                POSITION pos = clusters.GetHeadPosition();

                do
                {
                        c = clusters.GetNext(pos);
                        if (pos && c->getSize())
                        {
                                c->findCandidate(&clusters, pos, g->getAverageVertexWeight());
                        }
                } while ((c != tail) && (pos != NULL));
        }
}

VOID
SpatialPartitioner::mergeClosest()
{
        if (clusters.GetCount())
        {
                Cluster* c;
                Cluster* tail = clusters.GetTail();
                POSITION pos = clusters.GetHeadPosition();
                Cluster* toMerge = NULL;
                INT maxCloseness = 0;

                do
                {
                        c = clusters.GetNext(pos);
                        INT newCloseness = c->getCloseness();
```

```
                        if (newCloseness > maxCloseness)
                        {
                                maxCloseness = newCloseness;
                                toMerge = c;
                        }
                } while ((c != tail) && (pos != NULL));

                if (toMerge)
                {
                        Cluster* removing = toMerge->getClosestTo();

                        const UINTList* oldNeighbors = removing->getNeighbors();
                        const UINTList* newNeighbors = toMerge->getNeighbors();
                        unsigned legal = 1U;

                        if (oldNeighbors->GetCount())
                        {
                                UINT n;
                                UINT nTail = oldNeighbors->GetTail();
                                POSITION nPos = oldNeighbors->GetHeadPosition();

                                do
                                {
                                        n = oldNeighbors->GetNext(nPos);
                                        Cluster* check = findCluster(n, clusters);

                                        if (removing->isConnected(check))
                                        {
                                                legal = (newNeighbors->Find(n) != NULL);
                                        }
                                } while (legal && (n != nTail) && (nPos != NULL));
                        }

                        if (legal)
                        {
                                toMerge->addVertexList(removing->getVertices());
                                removing->getVertices()->RemoveAll();
                        }
                }
        }
}

VOID
SpatialPartitioner::refineClusters()
{
        BOOL improved = TRUE;
```

```
	while (improved)
	{
		improved = FALSE;

		if (clusters.GetCount())
		{
			Cluster* c;
			Cluster* tail = clusters.GetTail();
			POSITION pos = clusters.GetHeadPosition();

			while ((c = clusters.GetNext(pos)) != tail)
			{
				VertexList* list = c->getVertices();

				if (list->GetCount())
				{
					Vertex* v;
					Vertex* vTail = list->GetTail();
					POSITION vPos = list->GetHeadPosition();

					do
					{
						v = list->GetNext(vPos);
						Cluster* cl;
						POSITION cPos = pos;

						do
						{
							Cluster* toCluster;
							INT maxGain = 0;

							cl = clusters.GetNext(cPos);
							if (legalToAdd(v, cl))
							{
								if	(c->getNeighbors()->Find(cl->getNumber()))

								{
									if (c->isConnected(cl) && (c->getSize() >= cl->getSize()))

									{
										INT	sizeGain	=	c->getSize() - cl->getPotentialSize(v);

										INT oldNets = 0;
										INT newNets = 0;
```

```
EdgeList*  eList  =  v-
>getIncomingEdgeList();

if (eList->GetCount())
{
        Edge* e;
        Edge*  eTail  =
eList->GetTail();
        POSITION ePos =
eList->GetHeadPosition();

        do
        {
                e     =
eList->GetNext(ePos);
                Vertex*
source = e->getSource();

                if  (cl-
>contains(source))
                {
        oldNets += e->getWeight();
                }
                else  if
(c->contains(source))
                {
        newNets += e->getWeight();
                }
        }  while  (e  !=
eTail);
}

eList       =       v-
>getOutgoingEdgeList();

if (eList->GetCount())
{
        Edge* e;
        Edge*  eTail  =
eList->GetTail();
        POSITION ePos =
eList->GetHeadPosition();

        do
        {
```

```
                                                                        e     =
eList->GetNext(ePos);
                                                                        Vertex*
sink = e->getSink();

                                                                        if (cl-
>contains(sink))
                                                                        {

        oldNets += e->getWeight();
                                                                        }
                                                                        else if
(c->contains(sink))
                                                                        {

        newNets += e->getWeight();
                                                                        }
                                                                    } while (e !=
eTail);
                                                                }

                                                                INT netGain = oldNets
- newNets;
        //                                                      INT gain = sizeGain +
netGain;
                                                                INT gain = netGain;

                                                                if (gain > maxGain)
                                                                {
                                                                        toCluster = cl;
                                                                        maxGain = gain;
                                                                }
                                                            }
                                                        }

                                                        if (maxGain > 0)
                                                        {
                                                                improved = TRUE;
                                                                c->removeVertex(v);
                                                                toCluster->addVertex(v);
                                                        }
                                                    }
                                                } while (cl != tail);
                                            } while (v != vTail);
                                        }
                                    }
                                }
```

```
	}
}

Cluster*
SpatialPartitioner::findCluster(UINT clusterNumber, ClusterList& clusterList)
{
	Cluster* retval = NULL;

	if (clusterList.GetCount())
	{
		Cluster* c;
		Cluster* tail = clusterList.GetTail();
		POSITION pos = clusterList.GetHeadPosition();

		do
		{
			c = clusterList.GetNext(pos);
			if (c->getNumber() == clusterNumber)
			{
				retval = c;
				break;
			}
		} while ((c != tail) && (pos != NULL));
	}

	return retval;
}

BOOL
SpatialPartitioner::legalToAdd(const Vertex* v, const Cluster* toCluster)
{
	BOOL retval = toCluster->hasRoomFor(v);

	if (retval)
	{
		EdgeList* list = v->getIncomingEdgeList();

		if (list->GetCount())
		{
			Edge* e;
			Edge* tail = list->GetTail();
			POSITION pos = list->GetHeadPosition();

			do
			{
				e = list->GetNext(pos);
				Vertex* src = e->getSource();
```

```
                        if (src->isAdded())
                        {
                                Cluster*  srcCluster  =  findCluster(src->getClusterNumber(),
clusters);

                                if (srcCluster)
                                {
                                        if     (!toCluster->getNeighbors()->Find(srcCluster-
>getNumber())
                                                &&  (toCluster->getNumber()  !=  srcCluster-
>getNumber()))
                                        {
                                                retval = FALSE;
                                        }
                                }
                        }
                } while ((e != tail) && (pos != NULL) && retval);
        }

        if (retval)
        {
                list = v->getOutgoingEdgeList();

                if (list->GetCount())
                {
                        Edge* e;
                        Edge* tail = list->GetTail();
                        POSITION pos = list->GetHeadPosition();

                        do
                        {
                                e = list->GetNext(pos);
                                Vertex* snk = e->getSink();

                                if (snk->isAdded())
                                {
                                        Cluster*      snkCluster      =      findCluster(snk-
>getClusterNumber(), clusters);

                                        if (snkCluster)
                                        {
                                                if              (!toCluster->getNeighbors()-
>Find(snkCluster->getNumber())
                                                        &&     (toCluster->getNumber()     !=
snkCluster->getNumber()))
                                                {
```

```
                                                        retval = FALSE;
                                                }
                                        }
                                }
                        } while ((e != tail) && (pos != NULL) && retval);
                }
        }

        return retval;
}

VOID
SpatialPartitioner::initializeClusterList()
{
        clusters.RemoveAll();

        for (INT clusterCount = 1; clusterCount <= topology.GetCount(); clusterCount++)
        {
                Cluster* toAdd;

                if (toAdd = findCluster(clusterCount, topology))
                {
                        Cluster* newCluster = new Cluster(*toAdd);

                        newCluster->removeVertex(newCluster->getVertices()->GetHead());
                        clusters.AddTail(newCluster);
                }
        }
}

BOOL
SpatialPartitioner::recurseInsertion(Vertex* inV, POSITION inPos)
{
        BOOL added = FALSE;
        INT clusterCount = 1;

        while (!added && (clusterCount <= clusters.GetCount()))
        {
                Cluster* recipientCluster = findCluster(clusterCount, clusters);
                BOOL fits = legalToAdd(inV, recipientCluster);

                if (fits)
                {
                        recipientCluster->addVertex(inV);

                        if (inPos)
```

```
                        {
                                Vertex* v = g->getVertices()->GetNext(inPos);

                                if (!(added = recurseInsertion(v, inPos)))
                                {
                                        // this if/else resets inPos to the previous value
                                        if (inV == g->getVertices()->GetHead())
                                        {
                                                inPos = g->getVertices()->GetHeadPosition();
                                                Vertex* dummy = g->getVertices()->GetNext(inPos);
                                        }
                                        else
                                        {
                                                if (inPos)
                                                {
                                                        Vertex*     dummy     =     g->getVertices()-
>GetPrev(inPos);
                                                }
                                        }

                                        recipientCluster->removeVertex(inV);
                                        fits = FALSE;
                                }
                        }
                        else
                        {
                                return TRUE;
                        }
                }

                if (!fits)
                {
                        const UINTList* neighbors = recipientCluster->getNeighbors();

                        if (neighbors->GetCount())
                        {
                                UINT n;
                                UINT tail = neighbors->GetTail();
                                POSITION nPos = neighbors->GetHeadPosition();

                                do
                                {
                                        n = neighbors->GetNext(nPos);
                                        if (n > recipientCluster->getNumber())
                                        {
                                                Cluster* newRecipient = findCluster(n, clusters);
```

```
                                        if (legalToAdd(inV, newRecipient))
                                        {
                                                newRecipient->addVertex(inV);

                                                if (inPos)
                                                {
                                                        Vertex*    v    =    g->getVertices()-
>GetNext(inPos);

                                                        if  (!(added  =  recurseInsertion(v,
inPos)))
                                                        {
                                                                // this if/else resets inPos
to the previous value
                                                                if (inV == g->getVertices()-
>GetHead())
                                                                {
                                                                        inPos       =       g-
>getVertices()->GetHeadPosition();
                                                                        Vertex*  dummy  =  g-
>getVertices()->GetNext(inPos);
                                                                }
                                                                else
                                                                {
                                                                        if (inPos)
                                                                        {
                                                                                Vertex* dummy =
g->getVertices()->GetPrev(inPos);
                                                                        }
                                                                }

                                                                newRecipient-
>removeVertex(inV);
                                                        }
                                                }
                                                else
                                                {
                                                        return TRUE;
                                                }
                                        }
                                }
                        } while ((n != tail) && (nPos != NULL) && !added);
                }

                if (!added)
                {
                        clusterCount++;
```

```
                        }
                }
        }

        if (!added)
        {
                return FALSE;
        }

        return TRUE;
}

BOOL
SpatialPartitioner::insertVertices()
{
        VertexList* vertices = g->getVertices();

        if (vertices->GetCount())
        {
                POSITION pos = vertices->GetHeadPosition();
                Vertex* v = vertices->GetNext(pos);

                return recurseInsertion(v, pos);
        }

        return TRUE;
}

BOOL
SpatialPartitioner::makeClusters(UINT numClusters)
{
        BOOL retval = FALSE;

        initializeClusterList();

        if (insertVertices())
        {
                const UINT maxAttempts = 5;
                UINT attempts = 0;

                while   ((countNonemptyClusters(clusters)   >   numClusters)   &&   (attempts   <=
maxAttempts))
                {
                        findCandidates();
                        mergeClosest();
                        attempts++;
                }
```

```
                findCandidates();
                refineClusters();

                retval = TRUE;
        }

        return retval;
}

UINT
SpatialPartitioner::countNonemptyClusters(const ClusterList& list) const
{
        UINT retval = 0;

        if (list.GetCount())
        {
                Cluster* c;
                Cluster* tail = list.GetTail();
                POSITION pos = list.GetHeadPosition();

                do
                {
                        c = list.GetNext(pos);
                        if (c->getVertices()->GetCount())
                        {
                                retval++;
                        }
                } while ((c != tail) && (pos != NULL));
        }

        return retval;
}

UINT
SpatialPartitioner::countInterconnections(const ClusterList* list) const
{
        if (!list)
        {
                list = partition;
        }

        UINT count = 0;

        if (list->GetCount())
        {
                Cluster* c;
```

```
Cluster* tail = list->GetTail();
POSITION pos = list->GetHeadPosition();

do
{
	c = list->GetNext(pos);

	VertexList* vertices = c->getVertices();

	if (vertices->GetCount())
	{
		Vertex* v;
		Vertex* vTail = vertices->GetTail();
		POSITION vPos = vertices->GetHeadPosition();

		do
		{
			v = vertices->GetNext(vPos);

			EdgeList* edges = v->getIncomingEdgeList();

			if (edges->GetCount())
			{
				Edge* e;
				Edge* eTail = edges->GetTail();
				POSITION ePos = edges->GetHeadPosition();

				do
				{
					e = edges->GetNext(ePos);
					if (! c->contains(e->getSource()))
					{
						count++;
					}

				} while ((e != eTail) && (ePos != NULL));
			}

			edges = v->getOutgoingEdgeList();

			if (edges->GetCount())
			{
				Edge* e;
				Edge* eTail = edges->GetTail();
				POSITION ePos = edges->GetHeadPosition();

				do
```

```
                                        {
                                                e = edges->GetNext(ePos);
                                                if (! c->contains(e->getSink()))
                                                {
                                                        count++;
                                                }

                                        } while ((e != eTail) && (ePos != NULL));
                                }

                        } while ((v != vTail) && (vPos != NULL));
                }

        } while ((c != tail) && (pos != NULL));
    }

    return count / 2;
}

VOID
SpatialPartitioner::updateGain(Vertex* v, const Cluster* fromCluster, const Cluster* toCluster)
{
    EdgeList* list = v->getIncomingEdgeList();
    UINT oldNets = 0U;
    UINT newNets = 0U;

    if (legalToAdd(v, toCluster))
    {
        if (list->GetCount())
        {
            Edge* e;
            Edge* tail = list->GetTail();
            POSITION pos = list->GetHeadPosition();

            do
            {
                e = list->GetNext(pos);
                if (toCluster->contains(e->getSource()))
                {
                    oldNets++;
                }
                if (fromCluster->contains(e->getSink()))
                {
                    newNets++;
                }
            } while (e != tail);
        }
```

```
                list = v->getOutgoingEdgeList();

                if (list->GetCount())
                {
                        Edge* e;
                        Edge* tail = list->GetTail();
                        POSITION pos = list->GetHeadPosition();

                        do
                        {
                                e = list->GetNext(pos);
                                if (toCluster->contains(e->getSink()))
                                {
                                        oldNets++;
                                }
                                if (fromCluster->contains(e->getSource()))
                                {
                                        newNets++;
                                }
                        } while (e != tail);
                }
        }

        v->setGain(oldNets - newNets);
}

VOID
SpatialPartitioner::calculateGains(const Cluster* fromCluster, const Cluster* toCluster)
{
        VertexList* vertices = fromCluster->getVertices();

        if (vertices->GetCount())
        {
                Vertex* v;
                Vertex* tail = vertices->GetTail();
                POSITION pos = vertices->GetHeadPosition();

                do
                {
                        v = vertices->GetNext(pos);
                        updateGain(v, fromCluster, toCluster);
                } while (v != tail);
        }
}

Vertex*
```

```
SpatialPartitioner::findVertexWithLargestGain(const Cluster* c)
{
	Vertex* retval = NULL;
	INT maxGain = INT_MIN;
	VertexList* list = c->getVertices();

	if (list->GetCount())
	{
		Vertex* v;
		Vertex* tail;
		POSITION pos = list->GetHeadPosition();

		do
		{
			v = list->GetNext(pos);
			INT gain = v->getGain();

			if (gain > maxGain)
			{
				maxGain = gain;
				retval = v;
			}
		} while (v != tail);
	}

	if (maxGain <= 0)
	{
		retval = NULL;
	}

	return retval;
}

VOID
SpatialPartitioner::gcep_2way(Cluster* fromCluster, Cluster* toCluster)
{
	if (fromCluster && toCluster && fromCluster->getSize())
	{
		BOOL done = FALSE;
		INT midSize = (fromCluster->getSize() + toCluster->getSize()) / 2;
		double givenBal = midSize / fromCluster->getSize();
		double bal = (double)2 * givenBal;
		INT minSize = midSize - (INT)((double)midSize * bal);
		double reductionRatio = 0.9;

		while (!done)
		{
```

```
                        BOOL noMoreGain = FALSE;

                        calculateGains(fromCluster, toCluster);

                        while (!noMoreGain)
                        {
                                Vertex* toMove = findVertexWithLargestGain(fromCluster);

                                if (toMove)
                                {
                                        if  ((INT)fromCluster->getSize()  -  toMove->getWeight()  >=
minSize)
                                        {
                                                fromCluster->removeVertex(toMove);
                                                toCluster->addVertex(toMove);
                                                calculateGains(fromCluster, toCluster);
                                        }
                                        else
                                        {
                                                noMoreGain = TRUE;
                                        }
                                }
                                else
                                {
                                        noMoreGain = TRUE;
                                }
                        }

                        if (bal <= givenBal)
                        {
                                done = TRUE;
                        }
                        else
                        {
                                bal = __max(bal * reductionRatio, givenBal);
                        }
                }
        }
}

// saves the results to outFile
VOID
SpatialPartitioner::toFile() const
{
        CFile file(outfile, CFile::modeCreate | CFile::modeWrite);

        if (partition && partition->GetCount())
```

```
	{
		Cluster* c;
		Cluster* tail = partition->GetTail();
		POSITION pos = partition->GetHeadPosition();
		UINT idx = 0;

		do
		{
			c = partition->GetNext(pos);
			if (c->getSize())
			{
				UINT num = c->getNumber();
				char buf[20];

				sprintf(buf, "%u", num);
				CString line = CString(buf) + ":";

				VertexList* list = c->getVertices();

				if (list->GetCount())
				{
					Vertex* tail = list->GetTail();
					POSITION vPos = list->GetHeadPosition();
					Vertex* v = list->GetNext(vPos);

					line += v->getName();

					while ((v != tail) && (vPos != NULL))
					{
						v = list->GetNext(vPos);
						line += "," + v->getName();
					}
				}

				line += ";\n";
				file.Write(line, line.GetLength());
			}
		} while ((c != tail) && (pos != NULL));
	}
}

UINT
SpatialPartitioner::gcep(ClusterList* originalList)
{
	if (originalList->GetCount())
	{
		Cluster* c;
```

```
                Cluster* tail = originalList->GetTail();
                POSITION pos = originalList->GetHeadPosition();

                do
                {
                        c = originalList->GetNext(pos);
                        gcep_2way(c, c->getClosestTo());
                } while (c != tail);
        }

        return countInterconnections(originalList);
}

VOID
SpatialPartitioner::outputResults()
{
        toFile();
}

BOOL
SpatialPartitioner::hgcep()
{
        BOOL retval = FALSE;

        if (g && (g->getAverageVertexWeight() * g->getOrder() <= platformSize))
        {
                const UINT loopCount = 5U;
                const UINT largeCutoff = 8U;

                UINT numCells = g->getOrder();
                UINT mincut = INT_MAX;
                UINT cut = INT_MAX;

                for (UINT i = 0; i < loopCount; i++)
                {
                        if (makeClusters(numCells < i + largeCutoff ? numCells : numCells / (i +
largeCutoff)))
                        {
                                ClusterList* working = new ClusterList;
                                working->AddHead(&clusters);

                                if ((cut = gcep(working)) < mincut)
                                {
                                        mincut = mincut;
                                        delete partition;
                                        partition = working;
                                        retval = TRUE;
```

```
                                }
                                else
                                {
                                        delete working;
                                }
                        }
                }

                gcep(partition);
        }

        return retval;
}

BOOL
SpatialPartitioner::run()
{
        if (valid = hgcep())
        {
                outputResults();
        }

        return valid;
}

// Copies EVERYTHING
VOID
SpatialPartitioner::deepCopy(const SpatialPartitioner& rvalue_)
{
        if (this != &rvalue_)
        {
                filename = rvalue_.filename;
                g = rvalue_.g;
                valid = rvalue_.valid;

                delete partition;
                partition = new ClusterList;
                partition->AddHead((Cluster*)rvalue_.partition);

                clusters.RemoveAll();
                clusters.AddHead((Cluster*)&(rvalue_.clusters)); // have to cast away const

                topology.RemoveAll();
                topology.AddHead((Cluster*)&(rvalue_.topology)); // have to cast away const
        }
}
```

```
VOID
SpatialPartitioner::Serialize(CArchive& ar_)
{
	CObject::Serialize(ar_);

	((Graph*)g)->Serialize(ar_); // have to cast away const
	clusters.Serialize(ar_);
	topology.Serialize(ar_);
	partition->Serialize(ar_);

	if (ar_.IsStoring())
	{
		ar_ << filename;
		ar_ << valid;
	}
	else
	{
		ar_ >> filename;
		ar_ >> valid;
	}

}

VOID
SpatialPartitioner::printOn(ostream& os, size_t indentLevel) const
{
	UINT indentFiveMore = indentLevel + 5;

	os << "SpatialPartitioner {" << endl;
	os << indent(indentFiveMore) << "g = ";
	g->printOn(os, indentFiveMore);
	printTopology(os, indentFiveMore);
	printClusterList(os, indentFiveMore);
	printNonemptyPartitions(os, indentFiveMore);
	os << indent(indentFiveMore) << "valid = ";
	if (valid)
	{
		os << "TRUE";
	}
	else
	{
		os << "FALSE";
	}
	os << endl;

	os << indent(indentLevel) << "}" << endl;
}
```

```
// these three templated functions are for use by MFC during various calls made by CList or other
// container classes
template     <>     void     AFXAPI     ConstructElements<SpatialPartitioner>(SpatialPartitioner*
newSpatialPartitioners, int nCount)
{
    for (int i = 0; i < nCount; i++, newSpatialPartitioners++)
    {
        new(newSpatialPartitioners) SpatialPartitioner;
    }
}

template     <>     void     AFXAPI     DestructElements<SpatialPartitioner>(SpatialPartitioner*
newSpatialPartitioners, int nCount)
{
    for (int i = 0; i < nCount; i++, newSpatialPartitioners++)
    {
        delete(newSpatialPartitioners);
    }
}

template <> void AFXAPI SerializeElements<SpatialPartitioner> (CArchive& ar, SpatialPartitioner*
newSpatialPartitioners, int nCount)
{
    for (int i = 0; i < nCount; i++, newSpatialPartitioners++)
    {
        newSpatialPartitioners->Serialize(ar);
    }
}
```

A.4 SPMain.cpp

```
/*********************************************************************************************
**                                                                                          **
** SPMain.cpp                                                                               **
**                                                                                          **
** This file is the main() function which sets up the Graph and calls the                   **
** SpatialPartitioner.  It also shows timing results, the number of PEs required and the   **
**
** number of interconnections in the resulting solution.  This information can easily be  **
** removed once the SpatialPartitioner is beyond the laboratory state.                      **
**                                                                                          **
*********************************************************************************************/
#include <iostream.h>
#include <time.h>
#include <sys/timeb.h>

#include "Graph.h"
#include "Edge.h"
#include "SpatialPartitioner.h"
#include "Vertex.h"

void
main()
{
        Graph* g = new Graph();

        g->fromFile("graph.file");

        SpatialPartitioner sp("platform.top", g);

        struct _timeb timebuffer;

        _ftime( &timebuffer );
        unsigned long start = timebuffer.time;
        unsigned long startms = timebuffer.millitm;

        BOOL success = sp.run();

        _ftime( &timebuffer );

        cout   <<   ((unsigned   long)timebuffer.time   -   start)   *   1000UL   +   (unsigned
long)timebuffer.millitm - startms;
        cout << " milliseconds" << endl;
        cout << sp.countInterconnections() << " interconnections" << endl;
        cout << sp.countPEs() << " PEs" << endl;
```

```
        if (!success)
        {
                cout << "FAILED" << endl;
                exit(EXIT_FAILURE);
        }
}
```

Appendix B Files Associated With MCNC Circuit c1355nr

B.1 `platform.top` file associated with circuit c1355nr.

```
1:425:35:35:40:2,3;
3:100:20:5:20:1,5,6;
6:100:20:5:20:3,10;
10:425:35:35:40:6,13;

2:425:35:35:40:1,4,5;
5:100:0:0:0:2,3,8,9;
9:100:0:0:0:5,6,12,13;
13:425:35:35:40:9,10,15;

4:425:35:35:40:2,7,8;
8:100:0:0:0:4,5,11,12;
12:100:0:0:0:8,9,14,15;
15:425:35:35:40:12,13,16;

7:425:35:35:40:4,11;
11:100:0:0:0:7,8,14;
14:100:0:0:0:11,12,16;
16:425:35:35:40:14,15;
```

B.2 `graph.file` associated with circuit c1355nr.

```
"C1355NR"
{
     gat0:and:2:2:0,
     gat1:and:2:2:0,
     gat2:and:2:2:0,
     gat3:and:2:2:0,
     gat4:and:2:2:0,
     gat5:and:2:2:0,
     gat6:and:2:2:0,
     gat7:and:2:2:0,
     gat8:nand:2:2:0,
     gat9:nand:2:2:0,
     gat10:nand:2:2:0,
     gat11:nand:2:2:0,
     gat12:nand:2:2:0,
     gat13:nand:2:2:0,
     gat14:nand:2:2:0,
     gat15:nand:2:2:0,
     gat16:nand:2:2:0,
     gat17:nand:2:2:0,
     gat18:nand:2:2:0,
     gat19:nand:2:2:0,
     gat20:nand:2:2:0,
     gat21:nand:2:2:0,
     gat22:nand:2:2:0,
     gat23:nand:2:2:0,
     gat24:nand:2:2:0,
     gat25:nand:2:2:0,
     gat26:nand:2:2:0,
     gat27:nand:2:2:0,
```

```
gat28:nand:2:2:0,
gat29:nand:2:2:0,
gat30:nand:2:2:0,
gat31:nand:2:2:0,
gat32:nand:2:2:0,
gat33:nand:2:2:0,
gat34:nand:2:2:0,
gat35:nand:2:2:0,
gat36:nand:2:2:0,
gat37:nand:2:2:0,
gat38:nand:2:2:0,
gat39:nand:2:2:0,
gat40:nand:2:1:0,
gat41:nand:2:1:0,
gat42:nand:2:1:0,
gat43:nand:2:1:0,
gat44:nand:2:1:0,
gat45:nand:2:1:0,
gat46:nand:2:1:0,
gat47:nand:2:1:0,
gat48:nand:2:1:0,
gat49:nand:2:1:0,
gat50:nand:2:1:0,
gat51:nand:2:1:0,
gat52:nand:2:1:0,
gat53:nand:2:1:0,
gat54:nand:2:1:0,
gat55:nand:2:1:0,
gat56:nand:2:1:0,
gat57:nand:2:1:0,
gat58:nand:2:1:0,
```

```
gat59:nand:2:1:0,
gat60:nand:2:1:0,
gat61:nand:2:1:0,
gat62:nand:2:1:0,
gat63:nand:2:1:0,
gat64:nand:2:1:0,
gat65:nand:2:1:0,
gat66:nand:2:1:0,
gat67:nand:2:1:0,
gat68:nand:2:1:0,
gat69:nand:2:1:0,
gat70:nand:2:1:0,
gat71:nand:2:1:0,
gat72:nand:2:1:0,
gat73:nand:2:1:0,
gat74:nand:2:1:0,
gat75:nand:2:1:0,
gat76:nand:2:1:0,
gat77:nand:2:1:0,
gat78:nand:2:1:0,
gat79:nand:2:1:0,
gat80:nand:2:1:0,
gat81:nand:2:1:0,
gat82:nand:2:1:0,
gat83:nand:2:1:0,
gat84:nand:2:1:0,
gat85:nand:2:1:0,
gat86:nand:2:1:0,
gat87:nand:2:1:0,
gat88:nand:2:1:0,
gat89:nand:2:1:0,
```

```
gat90:nand:2:1:0,
gat91:nand:2:1:0,
gat92:nand:2:1:0,
gat93:nand:2:1:0,
gat94:nand:2:1:0,
gat95:nand:2:1:0,
gat96:nand:2:1:0,
gat97:nand:2:1:0,
gat98:nand:2:1:0,
gat99:nand:2:1:0,
gat100:nand:2:1:0,
gat101:nand:2:1:0,
gat102:nand:2:1:0,
gat103:nand:2:1:0,
gat104:nand:2:0:0,
gat105:nand:2:0:0,
gat106:nand:2:0:0,
gat107:nand:2:0:0,
gat108:nand:2:0:0,
gat109:nand:2:0:0,
gat110:nand:2:0:0,
gat111:nand:2:0:0,
gat112:nand:2:0:0,
gat113:nand:2:0:0,
gat114:nand:2:0:0,
gat115:nand:2:0:0,
gat116:nand:2:0:0,
gat117:nand:2:0:0,
gat118:nand:2:0:0,
gat119:nand:2:0:0,
gat120:nand:2:0:0,
```

```
gat121:nand:2:0:0,
gat122:nand:2:0:0,
gat123:nand:2:0:0,
gat124:nand:2:0:0,
gat125:nand:2:0:0,
gat126:nand:2:0:0,
gat127:nand:2:0:0,
gat128:nand:2:0:0,
gat129:nand:2:0:0,
gat130:nand:2:0:0,
gat131:nand:2:0:0,
gat132:nand:2:0:0,
gat133:nand:2:0:0,
gat134:nand:2:0:0,
gat135:nand:2:0:0,
gat136:nand:2:0:0,
gat137:nand:2:0:0,
gat138:nand:2:0:0,
gat139:nand:2:0:0,
gat140:nand:2:0:0,
gat141:nand:2:0:0,
gat142:nand:2:0:0,
gat143:nand:2:0:0,
gat144:nand:2:0:0,
gat145:nand:2:0:0,
gat146:nand:2:0:0,
gat147:nand:2:0:0,
gat148:nand:2:0:0,
gat149:nand:2:0:0,
gat150:nand:2:0:0,
gat151:nand:2:0:0,
```

```
gat152:nand:2:0:0,
gat153:nand:2:0:0,
gat154:nand:2:0:0,
gat155:nand:2:0:0,
gat156:nand:2:0:0,
gat157:nand:2:0:0,
gat158:nand:2:0:0,
gat159:nand:2:0:0,
gat160:nand:2:0:0,
gat161:nand:2:0:0,
gat162:nand:2:0:0,
gat163:nand:2:0:0,
gat164:nand:2:0:0,
gat165:nand:2:0:0,
gat166:nand:2:0:0,
gat167:nand:2:0:0,
gat168:nand:2:0:0,
gat169:nand:2:0:0,
gat170:nand:2:0:0,
gat171:nand:2:0:0,
gat172:nand:2:0:0,
gat173:nand:2:0:0,
gat174:nand:2:0:0,
gat175:nand:2:0:0,
gat176:nand:2:0:0,
gat177:nand:2:0:0,
gat178:nand:2:0:0,
gat179:nand:2:0:0,
gat180:nand:2:0:0,
gat181:nand:2:0:0,
gat182:nand:2:0:0,
```

```
gat183:nand:2:0:0,
gat184:nand:2:0:0,
gat185:nand:2:0:0,
gat186:nand:2:0:0,
gat187:nand:2:0:0,
gat188:nand:2:0:0,
gat189:nand:2:0:0,
gat190:nand:2:0:0,
gat191:nand:2:0:0,
gat192:nand:2:0:0,
gat193:nand:2:0:0,
gat194:nand:2:0:0,
gat195:nand:2:0:0,
gat196:nand:2:0:0,
gat197:nand:2:0:0,
gat198:nand:2:0:0,
gat199:nand:2:0:0,
gat200:nand:2:0:0,
gat201:nand:2:0:0,
gat202:nand:2:0:0,
gat203:nand:2:0:0,
gat204:nand:2:0:0,
gat205:nand:2:0:0,
gat206:nand:2:0:0,
gat207:nand:2:0:0,
gat208:nand:2:0:0,
gat209:nand:2:0:0,
gat210:nand:2:0:0,
gat211:nand:2:0:0,
gat212:nand:2:0:0,
gat213:nand:2:0:0,
```

```
gat214:nand:2:0:0,
gat215:nand:2:0:0,
gat216:nand:2:0:0,
gat217:nand:2:0:0,
gat218:nand:2:0:0,
gat219:nand:2:0:0,
gat220:nand:2:0:0,
gat221:nand:2:0:0,
gat222:nand:2:0:0,
gat223:nand:2:0:0,
gat224:nand:2:0:0,
gat225:nand:2:0:0,
gat226:nand:2:0:0,
gat227:nand:2:0:0,
gat228:nand:2:0:0,
gat229:nand:2:0:0,
gat230:nand:2:0:0,
gat231:nand:2:0:0,
gat232:nand:2:0:0,
gat233:nand:2:0:0,
gat234:nand:2:0:0,
gat235:nand:2:0:0,
gat236:nand:2:0:0,
gat237:nand:2:0:0,
gat238:nand:2:0:0,
gat239:nand:2:0:0,
gat240:nand:2:0:0,
gat241:nand:2:0:0,
gat242:nand:2:0:0,
gat243:nand:2:0:0,
gat244:nand:2:0:0,
```

```
gat245:nand:2:0:0,
gat246:nand:2:0:0,
gat247:nand:2:0:0,
gat248:nand:2:0:0,
gat249:nand:2:0:0,
gat250:nand:2:0:0,
gat251:nand:2:0:0,
gat252:nand:2:0:0,
gat253:nand:2:0:0,
gat254:nand:2:0:0,
gat255:nand:2:0:0,
gat256:nand:2:0:0,
gat257:nand:2:0:0,
gat258:nand:2:0:0,
gat259:nand:2:0:0,
gat260:nand:2:0:0,
gat261:nand:2:0:0,
gat262:nand:2:0:0,
gat263:nand:2:0:0,
gat264:nand:2:0:0,
gat265:nand:2:0:0,
gat266:nand:2:0:0,
gat267:nand:2:0:0,
gat268:nand:2:0:0,
gat269:nand:2:0:0,
gat270:nand:2:0:0,
gat271:nand:2:0:0,
gat272:nand:2:0:0,
gat273:nand:2:0:0,
gat274:nand:2:0:0,
gat275:nand:2:0:0,
```

```
gat276:nand:2:0:0,
gat277:nand:2:0:0,
gat278:nand:2:0:0,
gat279:nand:2:0:0,
gat280:nand:2:0:0,
gat281:nand:2:0:0,
gat282:nand:2:0:0,
gat283:nand:2:0:0,
gat284:nand:2:0:0,
gat285:nand:2:0:0,
gat286:nand:2:0:0,
gat287:nand:2:0:0,
gat288:nand:2:0:0,
gat289:nand:2:0:0,
gat290:nand:2:0:0,
gat291:nand:2:0:0,
gat292:nand:2:0:0,
gat293:nand:2:0:0,
gat294:nand:2:0:0,
gat295:nand:2:0:0,
gat296:not:1:0:0,
gat297:not:1:0:0,
gat298:not:1:0:0,
gat299:not:1:0:0,
gat300:not:1:0:0,
gat301:not:1:0:0,
gat302:not:1:0:0,
gat303:not:1:0:0,
gat304:not:1:0:0,
gat305:not:1:0:0,
gat306:not:1:0:0,
```

```
gat307:not:1:0:0,
gat308:not:1:0:0,
gat309:not:1:0:0,
gat310:not:1:0:0,
gat311:not:1:0:0,
gat312:not:1:0:0,
gat313:not:1:0:0,
gat314:not:1:0:0,
gat315:not:1:0:0,
gat316:not:1:0:0,
gat317:not:1:0:0,
gat318:not:1:0:0,
gat319:not:1:0:0,
gat320:not:1:0:0,
gat321:not:1:0:0,
gat322:not:1:0:0,
gat323:not:1:0:0,
gat324:not:1:0:0,
gat325:not:1:0:0,
gat326:not:1:0:0,
gat327:not:1:0:0,
gat328:not:1:0:0,
gat329:not:1:0:0,
gat330:not:1:0:0,
gat331:not:1:0:0,
gat332:not:1:0:0,
gat333:not:1:0:0,
gat334:not:1:0:0,
gat335:not:1:0:0,
gat336:and:3:0:0,
gat337:and:3:0:0,
```

```
gat338:and:3:0:0,
gat339:and:3:0:0,
gat340:and:3:0:0,
gat341:and:3:0:0,
gat342:and:3:0:0,
gat343:and:3:0:0,
gat344:or:4:0:0,
gat345:or:4:0:0,
gat346:and:5:0:0,
gat347:and:5:0:0,
gat348:and:5:0:0,
gat349:and:5:0:0,
gat350:and:5:0:0,
gat351:and:5:0:0,
gat352:and:5:0:0,
gat353:and:5:0:0,
gat354:and:2:0:0,
gat355:and:2:0:0,
gat356:and:2:0:0,
gat357:and:2:0:0,
gat358:and:2:0:0,
gat359:and:2:0:0,
gat360:and:2:0:0,
gat361:and:2:0:0,
gat362:and:2:0:0,
gat363:and:2:0:0,
gat364:and:2:0:0,
gat365:and:2:0:0,
gat366:and:2:0:0,
gat367:and:2:0:0,
gat368:and:2:0:0,
```

```
gat369:and:2:0:0,
gat370:and:2:0:0,
gat371:and:2:0:0,
gat372:and:2:0:0,
gat373:and:2:0:0,
gat374:and:2:0:0,
gat375:and:2:0:0,
gat376:and:2:0:0,
gat377:and:2:0:0,
gat378:and:2:0:0,
gat379:and:2:0:0,
gat380:and:2:0:0,
gat381:and:2:0:0,
gat382:and:2:0:0,
gat383:and:2:0:0,
gat384:and:2:0:0,
gat385:and:2:0:0,
gat386:nand:2:1:0,
gat387:nand:2:1:0,
gat388:nand:2:1:0,
gat389:nand:2:1:0,
gat390:nand:2:1:0,
gat391:nand:2:1:0,
gat392:nand:2:1:0,
gat393:nand:2:1:0,
gat394:nand:2:1:0,
gat395:nand:2:1:0,
gat396:nand:2:1:0,
gat397:nand:2:1:0,
gat398:nand:2:1:0,
gat399:nand:2:1:0,
```

```
gat400:nand:2:1:0,
gat401:nand:2:1:0,
gat402:nand:2:1:0,
gat403:nand:2:1:0,
gat404:nand:2:1:0,
gat405:nand:2:1:0,
gat406:nand:2:1:0,
gat407:nand:2:1:0,
gat408:nand:2:1:0,
gat409:nand:2:1:0,
gat410:nand:2:1:0,
gat411:nand:2:1:0,
gat412:nand:2:1:0,
gat413:nand:2:1:0,
gat414:nand:2:1:0,
gat415:nand:2:1:0,
gat416:nand:2:1:0,
gat417:nand:2:1:0,
gat418:nand:2:1:0,
gat419:nand:2:0:0,
gat420:nand:2:1:0,
gat421:nand:2:0:0,
gat422:nand:2:1:0,
gat423:nand:2:0:0,
gat424:nand:2:1:0,
gat425:nand:2:0:0,
gat426:nand:2:1:0,
gat427:nand:2:0:0,
gat428:nand:2:1:0,
gat429:nand:2:0:0,
gat430:nand:2:1:0,
```

```
gat431:nand:2:0:0,
gat432:nand:2:1:0,
gat433:nand:2:0:0,
gat434:nand:2:1:0,
gat435:nand:2:0:0,
gat436:nand:2:1:0,
gat437:nand:2:0:0,
gat438:nand:2:1:0,
gat439:nand:2:0:0,
gat440:nand:2:1:0,
gat441:nand:2:0:0,
gat442:nand:2:1:0,
gat443:nand:2:0:0,
gat444:nand:2:1:0,
gat445:nand:2:0:0,
gat446:nand:2:1:0,
gat447:nand:2:0:0,
gat448:nand:2:1:0,
gat449:nand:2:0:0,
gat450:nand:2:1:0,
gat451:nand:2:0:0,
gat452:nand:2:1:0,
gat453:nand:2:0:0,
gat454:nand:2:1:0,
gat455:nand:2:0:0,
gat456:nand:2:1:0,
gat457:nand:2:0:0,
gat458:nand:2:1:0,
gat459:nand:2:0:0,
gat460:nand:2:1:0,
gat461:nand:2:0:0,
```

```
gat462:nand:2:1:0,
gat463:nand:2:0:0,
gat464:nand:2:1:0,
gat465:nand:2:0:0,
gat466:nand:2:1:0,
gat467:nand:2:0:0,
gat468:nand:2:1:0,
gat469:nand:2:0:0,
gat470:nand:2:1:0,
gat471:nand:2:0:0,
gat472:nand:2:1:0,
gat473:nand:2:0:0,
gat474:nand:2:1:0,
gat475:nand:2:0:0,
gat476:nand:2:1:0,
gat477:nand:2:0:0,
gat478:nand:2:1:0,
gat479:nand:2:0:0,
gat480:nand:2:1:0,
gat481:nand:2:0:0,
gat482:nand:2:0:0,
gat483:nand:2:0:0,
gat484:nand:2:0:0,
gat485:nand:2:0:0,
gat486:nand:2:0:0,
gat487:nand:2:0:0,
gat488:nand:2:0:0,
gat489:nand:2:0:0,
gat490:nand:2:0:0,
gat491:nand:2:0:0,
gat492:nand:2:0:0,
```

```
gat493:nand:2:0:0,
gat494:nand:2:0:0,
gat495:nand:2:0:0,
gat496:nand:2:0:0,
gat497:nand:2:0:0,
gat498:nand:2:0:0,
gat499:nand:2:0:0,
gat500:nand:2:0:0,
gat501:nand:2:0:0,
gat502:nand:2:0:0,
gat503:nand:2:0:0,
gat504:nand:2:0:0,
gat505:nand:2:0:0,
gat506:nand:2:0:0,
gat507:nand:2:0:0,
gat508:nand:2:0:0,
gat509:nand:2:0:0,
gat510:nand:2:0:0,
gat511:nand:2:0:0,
gat512:nand:2:0:0,
gat513:nand:2:0:0,
gat514:buff:1:0:1,
gat515:buff:1:0:1,
gat516:buff:1:0:1,
gat517:buff:1:0:1,
gat518:buff:1:0:1,
gat519:buff:1:0:1,
gat520:buff:1:0:1,
gat521:buff:1:0:1,
gat522:buff:1:0:1,
gat523:buff:1:0:1,
```

```
gat524:buff:1:0:1,
gat525:buff:1:0:1,
gat526:buff:1:0:1,
gat527:buff:1:0:1,
gat528:buff:1:0:1,
gat529:buff:1:0:1,
gat530:buff:1:0:1,
gat531:buff:1:0:1,
gat532:buff:1:0:1,
gat533:buff:1:0:1,
gat534:buff:1:0:1,
gat535:buff:1:0:1,
gat536:buff:1:0:1,
gat537:buff:1:0:1,
gat538:buff:1:0:1,
gat539:buff:1:0:1,
gat540:buff:1:0:1,
gat541:buff:1:0:1,
gat542:buff:1:0:1,
gat543:buff:1:0:1,
gat544:buff:1:0:1,
gat545:buff:1:0:1;

net0:gat0:0:gat232:0:Edge:1,
net1:gat1:0:gat233:0:Edge:1,
net2:gat2:0:gat234:0:Edge:1,
net3:gat3:0:gat235:0:Edge:1,
net4:gat4:0:gat236:0:Edge:1,
net5:gat5:0:gat237:0:Edge:1,
net6:gat6:0:gat238:0:Edge:1,
net7:gat7:0:gat239:0:Edge:1,
```

```
net8:gat8:0:gat40:0:Edge:1,
net9:gat9:0:gat42:0:Edge:1,
net10:gat10:0:gat44:0:Edge:1,
net11:gat11:0:gat46:0:Edge:1,
net12:gat12:0:gat48:0:Edge:1,
net13:gat13:0:gat50:0:Edge:1,
net14:gat14:0:gat52:0:Edge:1,
net15:gat15:0:gat54:0:Edge:1,
net16:gat16:0:gat56:0:Edge:1,
net17:gat17:0:gat58:0:Edge:1,
net18:gat18:0:gat60:0:Edge:1,
net19:gat19:0:gat62:0:Edge:1,
net20:gat20:0:gat64:0:Edge:1,
net21:gat21:0:gat66:0:Edge:1,
net22:gat22:0:gat68:0:Edge:1,
net23:gat23:0:gat70:0:Edge:1,
net24:gat24:0:gat72:0:Edge:1,
net25:gat25:0:gat74:0:Edge:1,
net26:gat26:0:gat76:0:Edge:1,
net27:gat27:0:gat78:0:Edge:1,
net28:gat28:0:gat80:0:Edge:1,
net29:gat29:0:gat82:0:Edge:1,
net30:gat30:0:gat84:0:Edge:1,
net31:gat31:0:gat86:0:Edge:1,
net32:gat32:0:gat88:0:Edge:1,
net33:gat33:0:gat90:0:Edge:1,
net34:gat34:0:gat92:0:Edge:1,
net35:gat35:0:gat94:0:Edge:1,
net36:gat36:0:gat96:0:Edge:1,
net37:gat37:0:gat98:0:Edge:1,
net38:gat38:0:gat100:0:Edge:1,
```

```
net39:gat39:0:gat102:0:Edge:1,
net40:gat40:0:gat104:0:Edge:1,
net41:gat41:0:gat104:0:Edge:1,
net42:gat42:0:gat105:0:Edge:1,
net43:gat43:0:gat105:0:Edge:1,
net44:gat44:0:gat106:0:Edge:1,
net45:gat45:0:gat106:0:Edge:1,
net46:gat46:0:gat107:0:Edge:1,
net47:gat47:0:gat107:0:Edge:1,
net48:gat48:0:gat108:0:Edge:1,
net49:gat49:0:gat108:0:Edge:1,
net50:gat50:0:gat109:0:Edge:1,
net51:gat51:0:gat109:0:Edge:1,
net52:gat52:0:gat110:0:Edge:1,
net53:gat53:0:gat110:0:Edge:1,
net54:gat54:0:gat111:0:Edge:1,
net55:gat55:0:gat111:0:Edge:1,
net56:gat56:0:gat112:0:Edge:1,
net57:gat57:0:gat112:0:Edge:1,
net58:gat58:0:gat113:0:Edge:1,
net59:gat59:0:gat113:0:Edge:1,
net60:gat60:0:gat114:0:Edge:1,
net61:gat61:0:gat114:0:Edge:1,
net62:gat62:0:gat115:0:Edge:1,
net63:gat63:0:gat115:0:Edge:1,
net64:gat64:0:gat116:0:Edge:1,
net65:gat65:0:gat116:0:Edge:1,
net66:gat66:0:gat117:0:Edge:1,
net67:gat67:0:gat117:0:Edge:1,
net68:gat68:0:gat118:0:Edge:1,
net69:gat69:0:gat118:0:Edge:1,
```

```
net70:gat70:0:gat119:0:Edge:1,
net71:gat71:0:gat119:0:Edge:1,
net72:gat72:0:gat120:0:Edge:1,
net73:gat73:0:gat120:0:Edge:1,
net74:gat74:0:gat121:0:Edge:1,
net75:gat75:0:gat121:0:Edge:1,
net76:gat76:0:gat122:0:Edge:1,
net77:gat77:0:gat122:0:Edge:1,
net78:gat78:0:gat123:0:Edge:1,
net79:gat79:0:gat123:0:Edge:1,
net80:gat80:0:gat124:0:Edge:1,
net81:gat81:0:gat124:0:Edge:1,
net82:gat82:0:gat125:0:Edge:1,
net83:gat83:0:gat125:0:Edge:1,
net84:gat84:0:gat126:0:Edge:1,
net85:gat85:0:gat126:0:Edge:1,
net86:gat86:0:gat127:0:Edge:1,
net87:gat87:0:gat127:0:Edge:1,
net88:gat88:0:gat128:0:Edge:1,
net89:gat89:0:gat128:0:Edge:1,
net90:gat90:0:gat129:0:Edge:1,
net91:gat91:0:gat129:0:Edge:1,
net92:gat92:0:gat130:0:Edge:1,
net93:gat93:0:gat130:0:Edge:1,
net94:gat94:0:gat131:0:Edge:1,
net95:gat95:0:gat131:0:Edge:1,
net96:gat96:0:gat132:0:Edge:1,
net97:gat97:0:gat132:0:Edge:1,
net98:gat98:0:gat133:0:Edge:1,
net99:gat99:0:gat133:0:Edge:1,
net100:gat100:0:gat134:0:Edge:1,
```

net101:gat101:0:gat134:0:Edge:1,
net102:gat102:0:gat135:0:Edge:1,
net103:gat103:0:gat135:0:Edge:1,
net104:gat104:0:gat136:0:Edge:1,
net105:gat105:0:gat136:0:Edge:1,
net106:gat106:0:gat137:0:Edge:1,
net107:gat107:0:gat137:0:Edge:1,
net108:gat108:0:gat138:0:Edge:1,
net109:gat109:0:gat138:0:Edge:1,
net110:gat110:0:gat139:0:Edge:1,
net111:gat111:0:gat139:0:Edge:1,
net112:gat112:0:gat140:0:Edge:1,
net113:gat113:0:gat140:0:Edge:1,
net114:gat114:0:gat141:0:Edge:1,
net115:gat115:0:gat141:0:Edge:1,
net116:gat116:0:gat142:0:Edge:1,
net117:gat117:0:gat142:0:Edge:1,
net118:gat118:0:gat143:0:Edge:1,
net119:gat119:0:gat143:0:Edge:1,
net120:gat120:0:gat144:0:Edge:1,
net121:gat121:0:gat144:0:Edge:1,
net122:gat122:0:gat145:0:Edge:1,
net123:gat123:0:gat145:0:Edge:1,
net124:gat124:0:gat146:0:Edge:1,
net125:gat125:0:gat146:0:Edge:1,
net126:gat126:0:gat147:0:Edge:1,
net127:gat127:0:gat147:0:Edge:1,
net128:gat128:0:gat148:0:Edge:1,
net129:gat129:0:gat148:0:Edge:1,
net130:gat130:0:gat149:0:Edge:1,
net131:gat131:0:gat149:0:Edge:1,

net132:gat132:0:gat150:0:Edge:1,
net133:gat133:0:gat150:0:Edge:1,
net134:gat134:0:gat151:0:Edge:1,
net135:gat135:0:gat151:0:Edge:1,
net136:gat136:0:gat152:0:Edge:1,
net137:gat137:0:gat154:0:Edge:1,
net138:gat138:0:gat156:0:Edge:1,
net139:gat139:0:gat158:0:Edge:1,
net140:gat140:0:gat160:0:Edge:1,
net141:gat141:0:gat162:0:Edge:1,
net142:gat142:0:gat164:0:Edge:1,
net143:gat143:0:gat166:0:Edge:1,
net144:gat144:0:gat168:0:Edge:1,
net145:gat145:0:gat170:0:Edge:1,
net146:gat146:0:gat172:0:Edge:1,
net147:gat147:0:gat174:0:Edge:1,
net148:gat148:0:gat176:0:Edge:1,
net149:gat149:0:gat178:0:Edge:1,
net150:gat150:0:gat180:0:Edge:1,
net151:gat151:0:gat182:0:Edge:1,
net152:gat152:0:gat184:0:Edge:1,
net153:gat153:0:gat184:0:Edge:1,
net154:gat154:0:gat185:0:Edge:1,
net155:gat155:0:gat185:0:Edge:1,
net156:gat156:0:gat186:0:Edge:1,
net157:gat157:0:gat186:0:Edge:1,
net158:gat158:0:gat187:0:Edge:1,
net159:gat159:0:gat187:0:Edge:1,
net160:gat160:0:gat188:0:Edge:1,
net161:gat161:0:gat188:0:Edge:1,
net162:gat162:0:gat189:0:Edge:1,

net163:gat163:0:gat189:0:Edge:1,
net164:gat164:0:gat190:0:Edge:1,
net165:gat165:0:gat190:0:Edge:1,
net166:gat166:0:gat191:0:Edge:1,
net167:gat167:0:gat191:0:Edge:1,
net168:gat168:0:gat192:0:Edge:1,
net169:gat169:0:gat192:0:Edge:1,
net170:gat170:0:gat193:0:Edge:1,
net171:gat171:0:gat193:0:Edge:1,
net172:gat172:0:gat194:0:Edge:1,
net173:gat173:0:gat194:0:Edge:1,
net174:gat174:0:gat195:0:Edge:1,
net175:gat175:0:gat195:0:Edge:1,
net176:gat176:0:gat196:0:Edge:1,
net177:gat177:0:gat196:0:Edge:1,
net178:gat178:0:gat197:0:Edge:1,
net179:gat179:0:gat197:0:Edge:1,
net180:gat180:0:gat198:0:Edge:1,
net181:gat181:0:gat198:0:Edge:1,
net182:gat182:0:gat199:0:Edge:1,
net183:gat183:0:gat199:0:Edge:1,
net184:gat184:0:gat200:0:Edge:1,
net185:gat185:0:gat200:0:Edge:1,
net186:gat186:0:gat201:0:Edge:1,
net187:gat187:0:gat201:0:Edge:1,
net188:gat188:0:gat204:0:Edge:1,
net189:gat189:0:gat204:0:Edge:1,
net190:gat190:0:gat205:0:Edge:1,
net191:gat191:0:gat205:0:Edge:1,
net192:gat192:0:gat264:0:Edge:1,
net193:gat193:0:gat265:0:Edge:1,

net194:gat194:0:gat266:0:Edge:1,
net195:gat195:0:gat267:0:Edge:1,
net196:gat196:0:gat268:0:Edge:1,
net197:gat197:0:gat269:0:Edge:1,
net198:gat198:0:gat270:0:Edge:1,
net199:gat199:0:gat271:0:Edge:1,
net200:gat200:0:gat208:0:Edge:1,
net201:gat201:0:gat210:0:Edge:1,
net202:gat202:0:gat212:0:Edge:1,
net203:gat203:0:gat214:0:Edge:1,
net204:gat204:0:gat216:0:Edge:1,
net205:gat205:0:gat218:0:Edge:1,
net206:gat206:0:gat220:0:Edge:1,
net207:gat207:0:gat222:0:Edge:1,
net208:gat208:0:gat224:0:Edge:1,
net209:gat209:0:gat224:0:Edge:1,
net210:gat210:0:gat225:0:Edge:1,
net211:gat211:0:gat225:0:Edge:1,
net212:gat212:0:gat226:0:Edge:1,
net213:gat213:0:gat226:0:Edge:1,
net214:gat214:0:gat227:0:Edge:1,
net215:gat215:0:gat227:0:Edge:1,
net216:gat216:0:gat228:0:Edge:1,
net217:gat217:0:gat228:0:Edge:1,
net218:gat218:0:gat229:0:Edge:1,
net219:gat219:0:gat229:0:Edge:1,
net220:gat220:0:gat230:0:Edge:1,
net221:gat221:0:gat230:0:Edge:1,
net222:gat222:0:gat231:0:Edge:1,
net223:gat223:0:gat231:0:Edge:1,
net224:gat224:0:gat236:0:Edge:1,

net225:gat225:0:gat237:0:Edge:1,
net226:gat226:0:gat238:0:Edge:1,
net227:gat227:0:gat239:0:Edge:1,
net228:gat228:0:gat232:0:Edge:1,
net229:gat229:0:gat233:0:Edge:1,
net230:gat230:0:gat234:0:Edge:1,
net231:gat231:0:gat235:0:Edge:1,
net232:gat232:0:gat240:0:Edge:1,
net233:gat233:0:gat242:0:Edge:1,
net234:gat234:0:gat244:0:Edge:1,
net235:gat235:0:gat246:0:Edge:1,
net236:gat236:0:gat248:0:Edge:1,
net237:gat237:0:gat250:0:Edge:1,
net238:gat238:0:gat252:0:Edge:1,
net239:gat239:0:gat254:0:Edge:1,
net240:gat240:0:gat256:0:Edge:1,
net241:gat241:0:gat256:0:Edge:1,
net242:gat242:0:gat257:0:Edge:1,
net243:gat243:0:gat257:0:Edge:1,
net244:gat244:0:gat258:0:Edge:1,
net245:gat245:0:gat258:0:Edge:1,
net246:gat246:0:gat259:0:Edge:1,
net247:gat247:0:gat259:0:Edge:1,
net248:gat248:0:gat260:0:Edge:1,
net249:gat249:0:gat260:0:Edge:1,
net250:gat250:0:gat261:0:Edge:1,
net251:gat251:0:gat261:0:Edge:1,
net252:gat252:0:gat262:0:Edge:1,
net253:gat253:0:gat262:0:Edge:1,
net254:gat254:0:gat263:0:Edge:1,
net255:gat255:0:gat263:0:Edge:1,

net256:gat256:0:gat264:0:Edge:1,
net257:gat257:0:gat265:0:Edge:1,
net258:gat258:0:gat266:0:Edge:1,
net259:gat259:0:gat267:0:Edge:1,
net260:gat260:0:gat268:0:Edge:1,
net261:gat261:0:gat269:0:Edge:1,
net262:gat262:0:gat270:0:Edge:1,
net263:gat263:0:gat271:0:Edge:1,
net264:gat264:0:gat272:0:Edge:1,
net265:gat265:0:gat274:0:Edge:1,
net266:gat266:0:gat276:0:Edge:1,
net267:gat267:0:gat278:0:Edge:1,
net268:gat268:0:gat280:0:Edge:1,
net269:gat269:0:gat282:0:Edge:1,
net270:gat270:0:gat284:0:Edge:1,
net271:gat271:0:gat286:0:Edge:1,
net272:gat272:0:gat288:0:Edge:1,
net273:gat273:0:gat288:0:Edge:1,
net274:gat274:0:gat289:0:Edge:1,
net275:gat275:0:gat289:0:Edge:1,
net276:gat276:0:gat290:0:Edge:1,
net277:gat277:0:gat290:0:Edge:1,
net278:gat278:0:gat291:0:Edge:1,
net279:gat279:0:gat291:0:Edge:1,
net280:gat280:0:gat295:0:Edge:1,
net281:gat281:0:gat295:0:Edge:1,
net282:gat282:0:gat292:0:Edge:1,
net283:gat283:0:gat292:0:Edge:1,
net284:gat284:0:gat294:0:Edge:1,
net285:gat285:0:gat294:0:Edge:1,
net286:gat286:0:gat293:0:Edge:1,

net287:gat287:0:gat293:0:Edge:1,
net288:gat288:0:gat296:0:Edge:1,
net289:gat289:0:gat297:0:Edge:1,
net290:gat290:0:gat298:0:Edge:1,
net291:gat291:0:gat301:0:Edge:1,
net292:gat292:0:gat308:0:Edge:1,
net293:gat293:0:gat309:0:Edge:1,
net294:gat294:0:gat311:0:Edge:1,
net295:gat295:0:gat312:0:Edge:1,
net296:gat296:0:gat336:0:Edge:1,
net297:gat297:0:gat336:0:Edge:1,
net298:gat298:0:gat336:0:Edge:1,
net299:gat299:0:gat337:0:Edge:1,
net300:gat300:0:gat337:0:Edge:1,
net301:gat301:0:gat337:0:Edge:1,
net302:gat302:0:gat338:0:Edge:1,
net303:gat303:0:gat338:0:Edge:1,
net304:gat304:0:gat338:0:Edge:1,
net305:gat305:0:gat339:0:Edge:1,
net306:gat306:0:gat339:0:Edge:1,
net307:gat307:0:gat339:0:Edge:1,
net308:gat308:0:gat346:0:Edge:1,
net309:gat309:0:gat346:0:Edge:1,
net310:gat310:0:gat347:0:Edge:1,
net311:gat311:0:gat347:0:Edge:1,
net312:gat312:0:gat348:0:Edge:1,
net313:gat313:0:gat348:0:Edge:1,
net314:gat314:0:gat349:0:Edge:1,
net315:gat315:0:gat349:0:Edge:1,
net316:gat316:0:gat340:0:Edge:1,
net317:gat317:0:gat340:0:Edge:1,

```
net318:gat318:0:gat340:0:Edge:1,
net319:gat319:0:gat341:0:Edge:1,
net320:gat320:0:gat341:0:Edge:1,
net321:gat321:0:gat341:0:Edge:1,
net322:gat322:0:gat342:0:Edge:1,
net323:gat323:0:gat342:0:Edge:1,
net324:gat324:0:gat342:0:Edge:1,
net325:gat325:0:gat343:0:Edge:1,
net326:gat326:0:gat343:0:Edge:1,
net327:gat327:0:gat343:0:Edge:1,
net328:gat328:0:gat350:0:Edge:1,
net329:gat329:0:gat350:0:Edge:1,
net330:gat330:0:gat351:0:Edge:1,
net331:gat331:0:gat351:0:Edge:1,
net332:gat332:0:gat352:0:Edge:1,
net333:gat333:0:gat352:0:Edge:1,
net334:gat334:0:gat353:0:Edge:1,
net335:gat335:0:gat353:0:Edge:1,
net336:gat336:0:gat344:0:Edge:1,
net337:gat337:0:gat344:0:Edge:1,
net338:gat338:0:gat344:0:Edge:1,
net339:gat339:0:gat344:0:Edge:1,
net340:gat340:0:gat345:0:Edge:1,
net341:gat341:0:gat345:0:Edge:1,
net342:gat342:0:gat345:0:Edge:1,
net343:gat343:0:gat345:0:Edge:1,
net344:gat344:0:gat346:0:Edge:1,
net345:gat345:0:gat350:0:Edge:1,
net346:gat346:0:gat354:0:Edge:1,
net347:gat347:0:gat358:0:Edge:1,
net348:gat348:0:gat362:0:Edge:1,
```

```
net349:gat349:0:gat366:0:Edge:1,
net350:gat350:0:gat370:0:Edge:1,
net351:gat351:0:gat374:0:Edge:1,
net352:gat352:0:gat378:0:Edge:1,
net353:gat353:0:gat382:0:Edge:1,
net354:gat354:0:gat386:0:Edge:1,
net355:gat355:0:gat387:0:Edge:1,
net356:gat356:0:gat388:0:Edge:1,
net357:gat357:0:gat389:0:Edge:1,
net358:gat358:0:gat390:0:Edge:1,
net359:gat359:0:gat391:0:Edge:1,
net360:gat360:0:gat392:0:Edge:1,
net361:gat361:0:gat393:0:Edge:1,
net362:gat362:0:gat394:0:Edge:1,
net363:gat363:0:gat395:0:Edge:1,
net364:gat364:0:gat396:0:Edge:1,
net365:gat365:0:gat397:0:Edge:1,
net366:gat366:0:gat398:0:Edge:1,
net367:gat367:0:gat399:0:Edge:1,
net368:gat368:0:gat400:0:Edge:1,
net369:gat369:0:gat401:0:Edge:1,
net370:gat370:0:gat402:0:Edge:1,
net371:gat371:0:gat403:0:Edge:1,
net372:gat372:0:gat404:0:Edge:1,
net373:gat373:0:gat405:0:Edge:1,
net374:gat374:0:gat406:0:Edge:1,
net375:gat375:0:gat407:0:Edge:1,
net376:gat376:0:gat408:0:Edge:1,
net377:gat377:0:gat409:0:Edge:1,
net378:gat378:0:gat410:0:Edge:1,
net379:gat379:0:gat411:0:Edge:1,
```

```
net380:gat380:0:gat412:0:Edge:1,
net381:gat381:0:gat413:0:Edge:1,
net382:gat382:0:gat414:0:Edge:1,
net383:gat383:0:gat415:0:Edge:1,
net384:gat384:0:gat416:0:Edge:1,
net385:gat385:0:gat417:0:Edge:1,
net386:gat386:0:gat418:0:Edge:1,
net387:gat387:0:gat420:0:Edge:1,
net388:gat388:0:gat422:0:Edge:1,
net389:gat389:0:gat424:0:Edge:1,
net390:gat390:0:gat426:0:Edge:1,
net391:gat391:0:gat428:0:Edge:1,
net392:gat392:0:gat430:0:Edge:1,
net393:gat393:0:gat432:0:Edge:1,
net394:gat394:0:gat434:0:Edge:1,
net395:gat395:0:gat436:0:Edge:1,
net396:gat396:0:gat438:0:Edge:1,
net397:gat397:0:gat440:0:Edge:1,
net398:gat398:0:gat442:0:Edge:1,
net399:gat399:0:gat444:0:Edge:1,
net400:gat400:0:gat446:0:Edge:1,
net401:gat401:0:gat448:0:Edge:1,
net402:gat402:0:gat450:0:Edge:1,
net403:gat403:0:gat452:0:Edge:1,
net404:gat404:0:gat454:0:Edge:1,
net405:gat405:0:gat456:0:Edge:1,
net406:gat406:0:gat458:0:Edge:1,
net407:gat407:0:gat460:0:Edge:1,
net408:gat408:0:gat462:0:Edge:1,
net409:gat409:0:gat464:0:Edge:1,
net410:gat410:0:gat466:0:Edge:1,
```

net411:gat411:0:gat468:0:Edge:1,
net412:gat412:0:gat470:0:Edge:1,
net413:gat413:0:gat472:0:Edge:1,
net414:gat414:0:gat474:0:Edge:1,
net415:gat415:0:gat476:0:Edge:1,
net416:gat416:0:gat478:0:Edge:1,
net417:gat417:0:gat480:0:Edge:1,
net418:gat418:0:gat482:0:Edge:1,
net419:gat419:0:gat482:0:Edge:1,
net420:gat420:0:gat483:0:Edge:1,
net421:gat421:0:gat483:0:Edge:1,
net422:gat422:0:gat484:0:Edge:1,
net423:gat423:0:gat484:0:Edge:1,
net424:gat424:0:gat485:0:Edge:1,
net425:gat425:0:gat485:0:Edge:1,
net426:gat426:0:gat486:0:Edge:1,
net427:gat427:0:gat486:0:Edge:1,
net428:gat428:0:gat487:0:Edge:1,
net429:gat429:0:gat487:0:Edge:1,
net430:gat430:0:gat488:0:Edge:1,
net431:gat431:0:gat488:0:Edge:1,
net432:gat432:0:gat489:0:Edge:1,
net433:gat433:0:gat489:0:Edge:1,
net434:gat434:0:gat490:0:Edge:1,
net435:gat435:0:gat490:0:Edge:1,
net436:gat436:0:gat491:0:Edge:1,
net437:gat437:0:gat491:0:Edge:1,
net438:gat438:0:gat492:0:Edge:1,
net439:gat439:0:gat492:0:Edge:1,
net440:gat440:0:gat493:0:Edge:1,
net441:gat441:0:gat493:0:Edge:1,

```
net442:gat442:0:gat494:0:Edge:1,
net443:gat443:0:gat494:0:Edge:1,
net444:gat444:0:gat495:0:Edge:1,
net445:gat445:0:gat495:0:Edge:1,
net446:gat446:0:gat496:0:Edge:1,
net447:gat447:0:gat496:0:Edge:1,
net448:gat448:0:gat497:0:Edge:1,
net449:gat449:0:gat497:0:Edge:1,
net450:gat450:0:gat498:0:Edge:1,
net451:gat451:0:gat498:0:Edge:1,
net452:gat452:0:gat499:0:Edge:1,
net453:gat453:0:gat499:0:Edge:1,
net454:gat454:0:gat500:0:Edge:1,
net455:gat455:0:gat500:0:Edge:1,
net456:gat456:0:gat501:0:Edge:1,
net457:gat457:0:gat501:0:Edge:1,
net458:gat458:0:gat502:0:Edge:1,
net459:gat459:0:gat502:0:Edge:1,
net460:gat460:0:gat503:0:Edge:1,
net461:gat461:0:gat503:0:Edge:1,
net462:gat462:0:gat504:0:Edge:1,
net463:gat463:0:gat504:0:Edge:1,
net464:gat464:0:gat505:0:Edge:1,
net465:gat465:0:gat505:0:Edge:1,
net466:gat466:0:gat506:0:Edge:1,
net467:gat467:0:gat506:0:Edge:1,
net468:gat468:0:gat507:0:Edge:1,
net469:gat469:0:gat507:0:Edge:1,
net470:gat470:0:gat508:0:Edge:1,
net471:gat471:0:gat508:0:Edge:1,
net472:gat472:0:gat509:0:Edge:1,
```

net473:gat473:0:gat509:0:Edge:1,
net474:gat474:0:gat510:0:Edge:1,
net475:gat475:0:gat510:0:Edge:1,
net476:gat476:0:gat511:0:Edge:1,
net477:gat477:0:gat511:0:Edge:1,
net478:gat478:0:gat512:0:Edge:1,
net479:gat479:0:gat512:0:Edge:1,
net480:gat480:0:gat513:0:Edge:1,
net481:gat481:0:gat513:0:Edge:1,
net482:gat482:0:gat514:0:Edge:1,
net483:gat483:0:gat515:0:Edge:1,
net484:gat484:0:gat516:0:Edge:1,
net485:gat485:0:gat517:0:Edge:1,
net486:gat486:0:gat518:0:Edge:1,
net487:gat487:0:gat519:0:Edge:1,
net488:gat488:0:gat520:0:Edge:1,
net489:gat489:0:gat521:0:Edge:1,
net490:gat490:0:gat522:0:Edge:1,
net491:gat491:0:gat523:0:Edge:1,
net492:gat492:0:gat524:0:Edge:1,
net493:gat493:0:gat525:0:Edge:1,
net494:gat494:0:gat526:0:Edge:1,
net495:gat495:0:gat527:0:Edge:1,
net496:gat496:0:gat528:0:Edge:1,
net497:gat497:0:gat529:0:Edge:1,
net498:gat498:0:gat530:0:Edge:1,
net499:gat499:0:gat531:0:Edge:1,
net500:gat500:0:gat532:0:Edge:1,
net501:gat501:0:gat533:0:Edge:1,
net502:gat502:0:gat534:0:Edge:1,
net503:gat503:0:gat535:0:Edge:1,

```
    net504:gat504:0:gat536:0:Edge:1,
    net505:gat505:0:gat537:0:Edge:1,
    net506:gat506:0:gat538:0:Edge:1,
    net507:gat507:0:gat539:0:Edge:1,
    net508:gat508:0:gat540:0:Edge:1,
    net509:gat509:0:gat541:0:Edge:1,
    net510:gat510:0:gat542:0:Edge:1,
    net511:gat511:0:gat543:0:Edge:1,
    net512:gat512:0:gat544:0:Edge:1,
    net513:gat513:0:gat545:0:Edge:1;
}
```

B.3 `clusters.file` associated with circuit c1355nr.

```
1:gat0,gat1,gat2,gat3,gat4,gat5,gat6,gat7,gat8,gat27,gat28,gat29,
gat30,gat31,gat32,gat33,gat34,gat35,gat36,gat40,gat104,gat115,gat
116,gat117,gat118,gat119,gat120,gat121,gat122,gat123,gat124,gat12
5,gat126,gat127,gat128,gat129,gat130,gat131,gat132,gat133,gat134,
gat135,gat136,gat137,gat138,gat139,gat140,gat141,gat142,gat143,ga
t144,gat145,gat146,gat147,gat148,gat149,gat150,gat151,gat152,gat1
53,gat154,gat155,gat156,gat157,gat158,gat159,gat160,gat161,gat162
,gat163,gat164,gat165,gat166,gat167,gat168,gat169,gat170,gat171,g
at172,gat173,gat174,gat175,gat176,gat177,gat178,gat179,gat180,gat
181,gat182,gat183,gat184,gat185,gat186,gat187,gat188,gat189,gat19
0,gat191,gat192,gat193,gat194,gat195,gat196,gat197,gat198,gat199,
gat200,gat201,gat202,gat203,gat204,gat205,gat206,gat207,gat208,ga
t209,gat210,gat211,gat212,gat213,gat214,gat215,gat216,gat217,gat2
18,gat219,gat220,gat221,gat222,gat223,gat224,gat225,gat226,gat227
,gat228,gat229,gat230,gat231,gat232,gat233,gat234,gat235,gat236,g
at237,gat238,gat239,gat240,gat241,gat242,gat243,gat244,gat245,gat
246,gat247,gat248,gat249,gat250,gat251,gat252,gat253,gat254,gat25
5,gat256,gat257,gat258,gat259,gat260,gat261,gat262,gat263,gat264,
gat265,gat266,gat267,gat268,gat269,gat270,gat271,gat272,gat273,ga
t274,gat275,gat276,gat277,gat296;

2:gat37,gat38,gat39,gat41,gat42,gat43,gat44,gat45,gat46,gat47,gat
48,gat49,gat50,gat51,gat52,gat53,gat54,gat55,gat56,gat57,gat58,ga
t59,gat60,gat61,gat62,gat63,gat64,gat65,gat66,gat67,gat68,gat69,g
at70,gat71,gat72,gat73,gat74,gat75,gat76,gat77,gat78,gat79,gat80,
gat81,gat82,gat83,gat84,gat85,gat86,gat87,gat88,gat89,gat90,gat91
,gat92,gat93,gat94,gat95,gat96,gat97,gat98,gat99,gat100,gat101,ga
t102,gat103,gat278,gat279,gat280,gat281,gat282,gat283,gat284,gat2
85,gat286,gat287,gat288,gat289,gat290,gat291,gat292,gat293,gat294
,gat295,gat297,gat298,gat299,gat300,gat301,gat302,gat303,gat304,g
```

at305,gat306,gat307,gat308,gat309,gat310,gat311,gat312,gat313,gat314,gat315,gat316,gat317,gat318,gat319,gat320,gat321,gat322,gat323,gat324,gat325,gat326,gat327,gat328,gat329,gat330,gat331,gat332,gat333,gat334,gat335,gat336,gat337,gat338,gat339,gat340,gat341,gat342,gat343,gat344,gat345,gat346,gat347,gat348,gat349,gat350,gat351,gat352,gat353,gat354,gat355,gat356,gat357,gat358,gat359,gat362,gat366,gat370,gat374,gat378,gat382,gat386,gat387,gat388,gat389,gat390,gat391,gat9,gat10,gat11,gat12,gat13,gat14,gat15,gat16,gat17,gat18,gat19,gat20,gat21,gat22,gat23,gat24,gat25,gat105,gat106,gat107,gat108,gat109,gat110,gat111,gat112,gat113,gat114,gat26;

4:gat392,gat393,gat394,gat395,gat396,gat397,gat398,gat399,gat400,gat401,gat402,gat403,gat404,gat405,gat406,gat407,gat408,gat409,gat410,gat411,gat412,gat413,gat414,gat415,gat416,gat417,gat418,gat420,gat422,gat424,gat426,gat428,gat430,gat432,gat434,gat436,gat438,gat440,gat442,gat444,gat446,gat448,gat450,gat452,gat454,gat456,gat458,gat460,gat462,gat464,gat466,gat468,gat470,gat472,gat474,gat476,gat478,gat480,gat485,gat486,gat487,gat488,gat489,gat490,gat491,gat492,gat493,gat494,gat495,gat496,gat497,gat498,gat499,gat500,gat501,gat502,gat503,gat504,gat505,gat506,gat507,gat508,gat509,gat510,gat511,gat512,gat513,gat514,gat515,gat516,gat517,gat518,gat519,gat520,gat521,gat522,gat523,gat524,gat525,gat526,gat527,gat528,gat529,gat530,gat531,gat532,gat533,gat534,gat535,gat536,gat537,gat538,gat539,gat540,gat541,gat542,gat543,gat544,gat545,gat360,gat361,gat363,gat364,gat365,gat367,gat368,gat369,gat371,gat372,gat373,gat375,gat376,gat377,gat379,gat380,gat381,gat383,gat384,gat385,gat425,gat427,gat429,gat431,gat433,gat435,gat437,gat439,gat441,gat443,gat445,gat447,gat449,gat451,gat453,gat455,gat457,gat459,gat461,gat463,gat465,gat467,gat469,gat471,gat473,gat475,gat477,gat479,gat481,gat482,gat483,gat484,gat419,gat421,gat423;

Vita

David Moye was born in Radford, Virginia in 1971. Although born in Radford, he lived in Charleston, West Virginia until age 7 at which time he moved to St. Paul, Virginia. It was here, deep in the southwestern portion of Virginia that he lived through most of his impressionable years. At the age of 15, he again relocated; this time to Winter Garden, Florida. In this small suburb of Orlando is where he first met his now wife, Katie. When he completed high-school, he went immediately to Virginia Tech. Here he earned a Bachelor of Science in Computer Engineering and completed the Cooperative Education program. Once graduated, he took a job at Decision-Science Applications in Arlington, Virginia and was married shortly thereafter. Three years later, with the support of his wife, he re-enrolled at Virginia Tech where this work was completed as a part of his progress towards the Master's Degree in Computer Engineering. He currently works at IBM in RTP, NC in an ASIC development team and is planning to attend North Carolina State University in pursuit of his Doctor of Philosophy beginning in the Fall of 1999. Once that degree has been completed, he has plans to remain in academia as a professor and researcher.