IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 9, pp. 1237-1248, September, 1999.

Evaluation and Optimization of Replication Algorithms for Logic Bipartitioning

Morgan Enos, Scott Hauck, Majid Sarrafzadeh

Department of Electrical and Computer Engineering

Northwestern University

Evanston, IL 60208

{morgan, hauck, majid}@ece.nwu.edu

(Revised October 21, 1998, April 1, 1999)

Abstract

Logic partitioning is an important area of VLSI CAD, and there have been numerous approaches proposed. Logic replication, the duplication of logic in order to minimize communication between partitions, can be an effective component of a complete partitioning solution. In this paper we seek a better understanding of the important issues in logic replication. By adding new optimizations to existing algorithms we are able to significantly improve the quality of these techniques, achieving up to 13.9% better results than the best existing replication techniques. When integrated into our already state-of-the-art (non-replication) partitioner, we improve overall cutsizes by 38.8%, while requiring the duplication of at most 7% of the logic.

1 Introduction

Logic partitioning is one of the critical issues in CAD for digital logic. Effective algorithms for partitioning circuits enable one to apply divide-and-conquer techniques to simplify most of the steps in the mapping process. For example, standard-cell designs can be broken up so that a placement tool need only consider a portion of the overall design at any one time, yielding higher-quality results in a shorter period of time. Also, large designs must be broken up into pieces small enough to fit into multiple devices. Traditionally, this problem was important for breaking up a complex system into several custom ASICs. Now, with the increasing use of FPGA-based emulators and prototyping systems, partitioning is becoming even more critical.

For all of these tasks, the goal is to minimize the communication between partitions while ensuring that each partition is no larger than the capacity of the target device. While it is possible to solve the case of unbounded partition sizes exactly [Cheng88], the case of balanced partition sizes is NP-complete [Garey79]. As a result, numerous heuristic algorithms have been proposed [Alpert95a].

There has been a huge amount of research on logic partitioning, and it is unclear how to use this information to create the best partitioning solution. We demonstrated in a previous paper [Hauck95b, Hauck97] that a careful integration of existing bipartitioning techniques, along with some new approaches and optimizations, can yield a bipartitioning algorithm significantly better than the current state-of-the-art. Our algorithm achieved a 8%

improvement over PROP [Dutt96], 16% improvement over FBB [Yang94], 22% improvement over Paraboli [Reiss94] and MELO [Alpert95b], 50% improvement over Fiduccia-Mattheyses [Fiduccia82], and a 58% improvement over EIG1 [Hagen92] [Alpert95b], some of the best current bipartitioning algorithms.

In this paper, we seek to understand the critical issues in logic replication, the selective duplication of logic to reduce the resulting cutset. We examine most of the current logic replication literature, integrating it into our already extremely efficient partitioning system. We also develop techniques to significantly improve the quality of their results, yielding much smaller cutsets. This will lead to a complete logic bipartitioning algorithm with replication which delivers results much better than is currently possible.

1.1 Optimized FM

Most of the algorithms for replication are either based upon the Fiduccia-Mattheyses (FM) algorithm [Fiduccia82] (Figure 1), or use it to generate an initial partition. FM is a widely used algorithm, and as such, a number of ideas have emerged in the literature which improve upon the basic design. By a judicious combination of these ideas, significant improvements in performance can be realized. This is the premise behind Strawman [Hauck95a, Hauck95b, Hauck97], for this partitioner actually consists of many techniques which achieve a synergistic reduction in cut size. As far as we know, this partitioner delivers the best standard bipartitioning results in the literature today. Because of these strong results we have chosen to use this system as a base for our replication work.

Although a complete description of the Strawman partitioner is beyond the scope of this paper, it will be important to understand three of its optimizations.

```
Create initial partitioning;
While cutsize is reduced {
   While valid moves exist {
      Use bucket data structures to find unlocked node in each partition that most
         improves/least degrades cutsize when moved to other partition;
      Move whichever of the two nodes most improves/least degrades cutsize while not
         exceeding partition size bounds;
      Lock moved node;
      Update nets connected to moved nodes, and nodes connected to these nets;
   } endwhile;
   Backtrack to the point with minimum cutsize in move series just completed;
   Unlock all nodes;
} endwhile;
```

Figure 1. The Fiduccia - Mattheyses Algorithm.

- **Random initial partition creation.** The partitioner randomly creates an initial partition, and multiple runs of the partitioner use different initial partitions. In this way, the algorithm can be run multiple times, and the best of these multiple runs will be retained.

- **Hierarchical clustering and iterative unclustering.** The algorithm clusters the circuit recursively, building a hierarchy of clusters. Pairs of logic nodes are grouped at the bottom level, then pairs of basic clusters form larger clusters, and so on. In this manner, a cluster tree is obtained. During partitioning the algorithm first partitions at the top level of the hierarchy, with relatively few, but very large clusters forming the circuit. Once the partitioner can find no improvement (as in any traditional KL_FM partitioner) , the highest level of clustering is removed, and the partitioning resumed. At the next level, the partitioner deals with more clusters, each of which is smaller than clusters at the previous level. This continues until all levels of the clustering hierarchy are removed. At each level, the number of clusters increases and partitioning is performed until no improvement can be found.

- **Higher-level gains.** We use higher-level gains as proposed by Krishnamurthy [Krishnamurthy94]. This is a tie-breaking mechanism for nodes with identical gains which adds some foresight to the algorithm. A net contributes an nth level gain of +1 to a node *S* on the net if there are n-1 other nodes on the net in the same partition as *S* which are unlocked, and zero which are locked. A net contributes a -1 n-th level gain to a node *S* on the net if there are n-1 unlocked nodes on the net in the partition opposite to *S*, and zero locked. The first level gain is identical to the standard gains of the FM algorithm. In this paper we use three gain levels.

1.2 Methodology

Replication for cut minimization seeks out cells that when replicated across the partition reduce the number of nets in the cut set. Replication reduces the number of cut nets in a partition by creating redundancy in signal pathways.

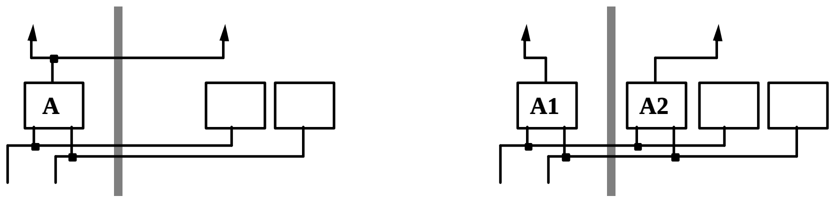

Figure 2. An example of node replication. Node **A** in the circuit at left is replicated into **A1** and **A2**, yielding a gain of 1.

As the above figure illustrates, the output net of a replicated cell need no longer span partitions since each partition has its own copy of the cell to generate the signal. However, to ensure that the correct signal is generated, every input net of the original cell must also input the replicated cell. Hence, the gain of cell replication is:

| cut output nets of cell | - | uncut, and *unreplicated* input nets of cell |

Equation 1. Gain of cell replication.

By *unreplicated* input net we mean to refer to input nets of the cell that have a replicated source. Since such nets can never be cut, they are excluded from the gain calculation.

To determine the effectiveness of the various replication algorithms, we run each algorithm thirty times on a SPARC5 workstation with each of the benchmarks in Table 1. Partitions are allowed to grow to at most 53.6% of the total circuit size (Strawman normally allows a partition to grow to 51% of total circuit size, and by allowing 5% of the logic to be replicated we reach an upper limit of .51*1.05 ≅ 0.536). When using Strawman to produce partitions for comparison against the various replication algorithms, we allow each partition to grow to 53.6% of the original circuit size.

Inputs to the circuit must be handled during replication. One could simply replicate them at will. However, in general a circuit will be given an input at only one place. Thus, to model this behavior we insist that if a circuit input is replicated, a net must be added to the cutset to carry the input between partitions. Details on how to extend each of the algorithms presented here to support this restriction can be found in [Enos96].

benchmark	# cells	# nets	# iopads	# pins
s5378	3225	3176	88	8241
s9234	6098	6076	45	15026
s13207	9445	9324	156	23442
s15850	11071	10984	105	27209
C6388	1956	1924	64	7046
biomed	6494	6442	77	24537
primary2	3035	3028	128	11294
struct	1952	1920	64	5535
s35932	19880	19560	359	55420
s38584	22451	22173	294	61309
s38417	25589	25483	138	64299

Table 1. Characteristics of benchmark circuits.

The 11 benchmarks used in this paper are given in Table 1. Although this is a subset of the circuits typically used in partitioning research, these were the only circuits available to us that contained information on what node generates a given net (the signal source), something that is necessary for replication. The other circuits in the benchmark suite are only available to us in a format that does not specify the net's source, and thus cannot be used for replication.

In the sections that follow we present most of the replication algorithms from the literature, as well as some novel extensions to these techniques. Throughout this paper, algorithms directly from the literature will be referred to as "Basic". Sections 2-4 present move-based replication algorithms, algorithms that integrate replication into the FM inner loop. In section two we detail Kring/Newton replication. This will also detail new techniques for integrating together Kring/Newton replication with higher-level gains, techniques for using Kring/Newton within a clustered partitioning, and a gradient approach for more intelligently applying replication. Section three will explore DFRG replication, and present new techniques to integrate it into other advanced partitioning optimizations. Section four finishes the move-based replication techniques by considering Functional Replication. Section five contains flow-based techniques which perform replication as a post-processing step. We detail the Hyper-MAMC algorithm, and demonstrate how new techniques for selecting flow sources, node choice methods for reducing the size of the replication set, and tradeoffs between maximum partition sizes for bipartitioning and replication can greatly improve this algorithm. We round out the flow based techniques by detailing the MC-Rep algorithm. Finally, we describe combinations of these replication algorithms which can be used to produce the best possible results.

2 Kring/Newton Replication

The algorithm proposed by Kring and Newton (hereafter referred to as KN) is a fairly straight-forward adaptation of FM to support cell replication [Kring91]. In the FM algorithm, a node can exist in one of two states dependent upon which partition it is currently located. Kring and Newton introduce a third state – replication, in which a node exists in both partitions simultaneously. Also, nodes now have two potential moves: unreplicated nodes can either be replicated or moved to the other partition, and replicated nodes can be unreplicated into either of the two partitions.

The Kring/Newton algorithm always requires that an unreplication move with a positive gain be taken before any normal moves, and disallows replication moves with a gain of less than 1. Our experiments indicate that allowing replication moves with a gain of 0 are useful, and our version of Kring/Newton allows such moves. The difference may be due to the fact that we use gate-level (single-output only) circuits, where several nodes may need to be replicated in order to improve the cutsize. In section 2.1 we detail the basic Kring/Newton algorithm, and then detail several new optimization in sections 2.2-2.4.

2.1 Basic Kring/Newton

The simplest way to integrate Kring/Newton replication into Strawman's hierarchical clustering approach is to replace Strawman's final FM run on the unclustered circuit with the Kring/Newton algorithm. This forms the basic KN algorithm. This is essentially the KN algorithm performed on an initial partition which has been optimized – a modification proposed by Kring and Newton which was found to be useful in partitioning larger benchmarks.

Benchmark	Strawman	Basic KN	KN w/higher-level gains	KN w/Clustering	KN Gradient
s5378	60	41	39	48	43
s9234	45	30	32	33	32
s13207	67	53	48	49	43
s15850	52	43	43	39	41
C6288	50	33	33	34	33
biomed	161	136	137	126	130
primary2	133	114	114	110	114
struct	33	33	33	33	33
s35932	46	39	39	28	23
s38584	50	35	35	46	32
s38417	55	51	49	47	49
Geom. Mean	60.8	48.2	47.7	47.6	44.7
Time	17.4	17.3	17.6	17.1	19.7
Replication	0.0%	4.2%	4.5%	6.2%	3.5%

Table 2. Comparison of different optimizations to Kring/Newton replication. Each algorithm is executed thirty times, and the best cutsize is kept. "Time" is the geometric mean total CPU minutes for all 30 partitionings for a given benchmark. "Replication" is the average percentage of the logic that is replicated to achieve the best cut for each benchmark.

2.2 Higher-Level Gains

In general, if we have two moves that have the same impact on the cutsize, but one is a normal move and one is a replication move, we should choose the normal move in order to limit the amount of replication. To do this, we set all higher-level gains of replication moves to -*max_fanin*, which forces replication moves to happen after normal moves with the same normal gain. For ordinary moves we retain the original definition of higher-level gains. For unreplication moves, we retain the original definition for all levels greater than one. This is true because once the unreplication move has been performed (the first level gain), both the node and the net revert to the original state for which the higher-level gains were formulated. As shown in Table 2, this achieves a slight improvement over KN with a normal gain function, and thus all future KN variants in this paper will be performed with higher-level gains.

2.3 Extension to Clustering

It was hoped that by earlier application of KN, better results might be achieved via increased opportunity for replication. So rather than perform KN style replication only after complete unclustering, KN replication was performed at each clustering level in place of the Strawman FM algorithm. To facilitate replication at each level of the clustering hierarchy, the total allowed circuit expansion was evenly distributed among all levels, and so if there were seven clustering levels, and we allowed up to seven percent circuit expansion via replication, then KN would be allowed to expand the circuit by one percent at each clustering level. Such expansion is cumulative, so at each clustering level the total amount of circuit expansion allowed is greater than the previous. Without this incremental circuit expansion replication in the first couple of clustering levels would inevitably expand the circuit maximally, and therefore severely limit replication in succeeding levels. This promotes early, and relatively ignorant replication over later replication, and tends to produce poorer results.

2.4 Gradient Method

As the FM algorithm progresses, the partitions become steadily better. Early in the algorithm there may be many high gain moves, and the partitioner radically changes the current partitioning. However, in later iterations the current partitioning is close to the final result, and the partitioner makes relatively minor alterations to the partitioning. When we add replication to this process, it can create a significant amount of duplication early in the partitioning process, limiting the partitioner’s ability to change the current partitioning. Thus, replicating too early can degrade the final results. To deal with this, in Gradient KN we only attempt further replication when the cut sizes between successive inner-loop iterations of the Strawman FM algorithm change by less than ten percent (hence the term Gradient). This procedure is applied at each clustering level in place of the Strawman FM algorithm. The circuit is expanded incrementally as in the previous section.

2.5 Kring/Newton Results

The results for the various optimizations to the Kring/Newton algorithm are given in Table 2. As can be seen, basic Kring/Newton produces results 20.7% better than the non-replicating algorithm. Higher-level gains improve this by an additional 0.8%, Kring/Newton hierarchical partitioning yields an additional 0.2%, and the Gradient approach yields an additional 4.8%. This produces an overall 26.5% improvement over the basic Strawman algorithm.

3 DFRG

Another move based approach is the Directed Fiduccia-Mattheyses on a Replication Graph, or DFRG [Liu95b]. DFRG uses a directed version of FM. In this algorithm one partition is designated the “source” partition, and a net is considered cut only if one of its sources is in the source partition, and one of its sinks is in the other partition.

Imagine that we take the graph being partitioned and duplicate it, and in the duplicate we reverse the direction of signal flow (net sources become sinks, net sinks become sources). We then connect each node to its duplicate with two infinite weight edges (edges that will never be cut in a reasonable partitioning), one directed from original to duplicate, another from duplicate to original (see Figure 3 left). This means that a node and its duplicate will always been in the same partition. Assuming a partitioning does not cut an infinite weight edge, it can be shown that the directed cutsize of this graph is the same as the undirected cutsize of the nets in the original graph. Thus, a directed partitioning of this graph is equivalent to an undirected partitioning of the original graph.

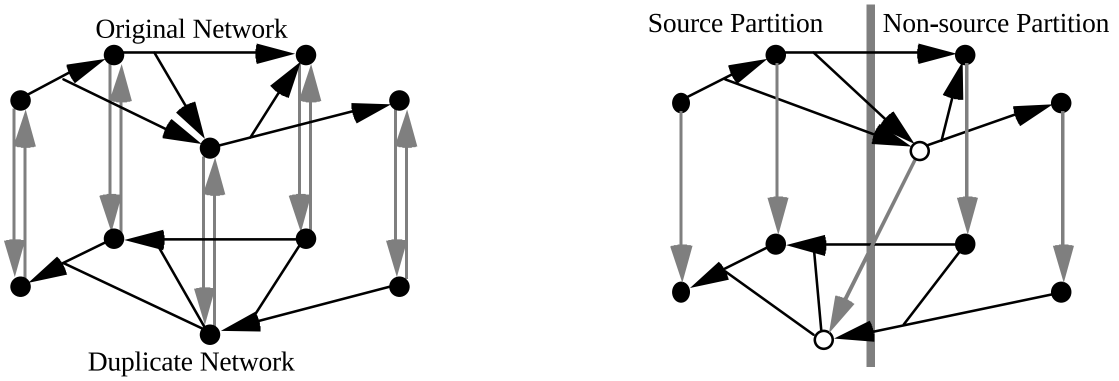

Figure 3. Hypothetical duplicated graph with infinite edges in both directions between original and duplicated nodes (left). Gray arrows are infinite weight edges. The replication graph has infinite edges only from original to duplicate (right). A duplicate node in the source partition with it’s original in the non-source partition is considered duplicated (such as the node shown in white).

In DFRG partitioning we use a the same graph, but remove the infinite weight edges from duplicate to original (the edge from original to duplicate remains). In this graph we can now have an original node in the non-source partition, yet its duplicate is in the source partition (see Figure 3 right). All such nodes will be replicated, and the directed cutsize of this graph accurately reflects this. Specifically, it can be shown that all input nets to a replicated node will be in the directed cutset – else, by definition, one of the inputs is also replicated -- and the outputs of a replicated node are never in the directed cutset – otherwise, there is an infinite weight edge from the source partition to the non-source partition and thus the cut cost is infinite. Thus, performing directed partitioning of this replication graph formulation allows partitioning and replication of the original graph to be performed simultaneously. A size constraint on the non-source partition can be maintained by limiting the number of original nodes in the non-source partition, while the constraint on the source partition is maintained by limiting the number of duplicated nodes in the source partition. In this way, the total amount of logic in each partition, including both normal and replicated nodes, can be exactly controlled.

For the basic DFRG algorithm we obtain an initial partition via Strawman and use DFRG as a post-processing step upon the gate-level nodes and sweep clusters. This is essentially DFRG as performed upon a good initial partition. In the next two subsections that follow we present two new techniques, proposed by us, for improving the DFRG results.

3.1 Higher-Level Gains

A net can be uncut either by moving all sources to the “non-source” partition, or by moving all sinks to the “source” partition. By moving a source, one does not lose any opportunities for uncutting a net via sink movements, and by moving a sink, one does not lose any opportunities for removing the net from the cut set via source movements. Since source and sink motion operates somewhat independently in determining the state of a net, we use the following higher-level gains formulation: a cut net contributes +1 to the nth level gain of a source node S if S is in the “source” partition, and there are n-1 other unlocked sources of the net, and no locked sources, in the “source” partition; a cut net contributes a -1 to the nth level gain of a source node S if S is in the “non-source” partition, and there are n-1 unlocked sources, and no locked sources, in the source partition; Higher-level gains for sink nodes are defined similarly. For ease of implementation, only cut nets contribute to the higher level gains of a node.

Benchmark	Strawman	Basic DFRG	DFRG w/Higher-Level Gains	DFRG w/Clustering
s5378	60	40	41	41
s9234	45	34	34	32
s13207	67	54	47	63
s15850	52	45	44	45
C6288	50	33	33	33
biomed	161	136	136	107
primary2	133	114	116	121
struct	33	33	33	33
s35932	46	39	39	24
s38584	50	37	35	29
s38417	55	48	48	45
Geom. Mean	60.8	48.9	48.2	44.9
Time	17.4	27.8	31.6	48.2
Replication	0.0%	3.5%	3.5	5.1%

Table 3. Comparison of different optimizations to DFRG replication. Each algorithm is executed thirty times, and the best cutsize is kept. “Time” is the geometric mean total CPU minutes for all 30 partitionings for a given benchmark. “Replication” is the average percentage of the logic that is replicated to achieve the best cut for each benchmark.

3.2 Extension to Clustering

Because DFRG more than doubles the size of the network before partition, it was thought that DFRG would benefit considerably if the problem size were reduced through clustering. We perform DFRG at each clustering level in place of the Strawman FM algorithm, and gradually expand the maximum circuit size as in the KN extension to clustering. The clustering is performed upon the original graph, and the replication graph is created from the clustered graph at each clustering level. We retain the use of higher-level gains as explained in the previous section.

3.3 DFRG Results

The results for the various optimizations to the DFRG algorithm are given in Table 3. As can be seen, basic DFRG produces results 19.6% better than the non-replicating algorithm. Higher-level gains improve this by an additional 1.1% and DFRG hierarchical partitioning yields an additional 5.5%. These techniques combined together produce an algorithm that yields results 26.2% better than the basic Strawman algorithm. Note that the DFRG algorithm, especially when applied at each level of the clustering algorithm, greatly increases the runtime of the algorithm. This is due to the overhead of creating the special graph required for this algorithm.

4 Functional Replication

Functional Replication is similar to KN, but is specialized for partitioning onto Xilinx-based (3000 series) devices [Kuznar94]. Rather than using gate-level nodes as the base element for partitioning, functional replication uses configurable logic blocks (CLBs), and therefore the circuit must be technology mapped prior to partitioning. Each CLB will take the place of multiple gates from the original gate-level netlist, and will produce one or two outputs.

Under normal partitioning a CLB must be moved as a single unit. However, the outputs of a two-output CLB may be required in different partitions. Keeping the CLB as a single unit would force one of the outputs into the cutset. Under Functional Replication such a CLB could be split into two CLBs, with each CLB generating a single output (see Figure 4). Also, since not all inputs will be needed to generate each output, the CLBs are connected to only those inputs needed to generate its output. Thus, unlike other replication techniques, Functional Replication does not necessarily force all of its inputs into the cutset. Note that the splitting of a two-output CLB is the only type of replication allowed under Functional Replication; Replication of one-output CLBs, as well as the complete duplication of a two-output CLB into both partitions, is not allowed. In order to perform Functional Replication, the basic FM algorithm is extended similarly to the KN algorithm. Two output CLBs can be in four states, with each output independently assigned to either partition, and when outputs are assigned to different partitions the CLB is replicated. In our implementation of Functional Replication we set a hard limit of seven percent CLB expansion. Note that the benchmark suite is reduced because for functional replication we need information on the functions computed in each node, and we were only able to get this information for these benchmarks.

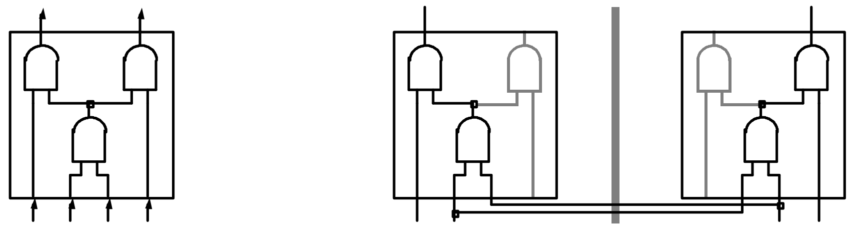

Figure 4. Functional Replication allows a two-output CLB (left) to be replicated into two CLBs, with each copy only generating a single output (right). Each copy is only connected to those inputs needed to generate the output of that CLB.

Benchmark	Best Strawman	KN w/Gradient	DFRG w/Clustering	Functional Replication
s5378	60	43	41	55
s9234	45	32	32	48
s13207	67	43	63	79
s15850	52	41	45	59
s35932	46	23	24	42
s38584	50	32	29	50
s38417	55	49	45	30
Geom. Mean	53.1	36.6	38.1	49.9
Replication	0.0%	3.5%	5.1%	2.9%

Table 4. Comparison of functional replication with other replication algorithms.

5 Flow Based Techniques

The final pair of algorithms are significantly different from the preceding in that they are flow based rather than move based. The flow based algorithms strive to capitalize on a flow network formulation which derives an optimal minimum-cut replication set (without size bounds) when provided with an initial partition. To develop this technique, we require the following two definitions [Cormen90] [Ford62]:

A *flow network*, G = (V,E), is a directed graph in which each edge, $(u,v) \in E$, has a non-negative capacity, and there exists two special nodes $S, T \in V$ designated as a source and sink respectively.

A *flow* in G is a function $f: V \times V \rightarrow \Re$ (where "$\Re$" is the set of reals) such that

(1) $f(u,v) \le c(u,v), \quad u,v \in V,$ where $c(u,v)$ is the capacity of the edge $(u,v) \in E$

(2) $f(u,v) = -f(v,u), \quad u,v \in V$

(3) $\sum_{v \in V} f(u,v) = 0, \; u \in V - \{S,T\}$

Also needed is the Ford-Fulkerson max-flow/min-cut theorem [Ford62]:

Theorem: *For any network the maximal flow value from S to T is the size of some minimum cut in G.*

5.1 The Optimal Algorithm for Unconstrained Replication Size [Hwang95, Yang95]

Given an initial partitioning of the vertices we select a source partition arbitrarily. We form a flow network by creating a node S, which will be the source of the network, and clustering with S nodes in the source partition which we would like to act as a source of flow in the network (see Figure 5 left and center). We also create a node T, which will be the sink of the network, and cluster with T all nodes which are in the non-source (sink) partition, and are attached to edges which have their source in the source partition. All edges which have their source in the sink partition are removed from the network, and all remaining edges are given a capacity of one.

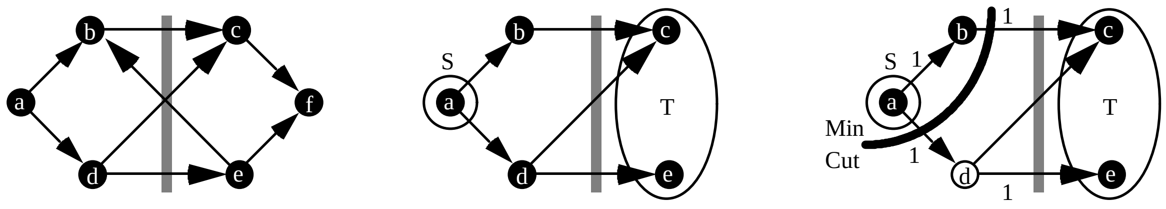

Figure 5. A directed graph (left) is turned into a flow graph (center). Once a maximum flow is found, which induces a minimum cut, nodes between the min cut and the original partitioning are replicated (right). The replicated node is shown in white.

We push as much flow from S to T as we can, and thus by the Ford-Fulkerson Theorem we produce a minimum cut. We find the minimum cut closest to T by a breadth-first search from T, and every node in partition X that is on the sink side of the minimum cut is designated for replication. By replicating these designated nodes we place all edges in the minimum cut into the cut set, and guarantee that any other edge which has its source in the source partition is not in the cut set. The original graph is then reestablished; we select the opposite partition as the source partition, and we repeat the above. By finding the replication subset which minimizes the cut edges which have their sources in both partitions we find the replication set which minimizes the number of cut edges for this partitioning.

The above algorithm has two limitations: it applies only to directed graphs (not hypergraphs), and it assumes unbounded partition size. By extending this algorithm to incorporate directed hyper-edges or size constraints we lose optimality, and therefore must resort to heuristics.

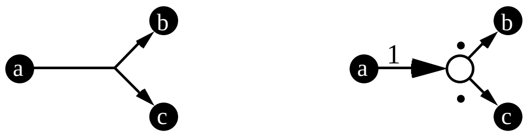

Figure 6. Reduction of hyper-edges to normal edges. White node is the hyper-node.

5.2 Hyper-MAMC (Min-Area Min-Cut) [Yang95]

In order to use the network flow formulation we must reduce hyperedges to edges. This is done as shown in Figure 6. For each hyperedge we create a hypernode with an edge of weight 1 from the hyperedge's source to the hypernode, and an edge of infinite weight from the hypernode to each of the sinks of the hyperedge. In this way, we restrict each hyperedge to unit flow, and ensure that the hyperedge can only be cut in one place - the edge from the source to the hypernode. This guarantees that each hyperedge contributes at most once to the minimum cut derived from the maximum flow.

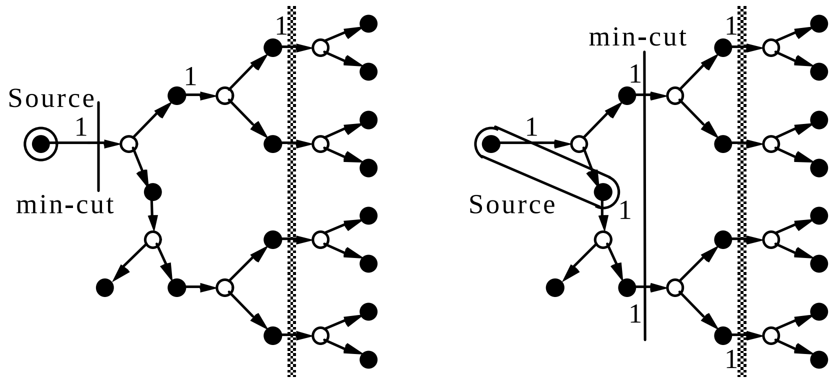

Figure 7. Given an initial flow network that replicates too much logic (left), incrementing the flow reduces the replication set and increases the cut size (right).

Hyper-MAMC proceeds identically to the optimal algorithm, pushing flow through the network and identifying nodes to be replicated. However, before actually replicating these nodes, it first checks if this amount of replication would exceed the partition size limitation. If so, we determine all hyperedges which have their source clustered with the network source, and which have positive flow. A sink of such a hyperedge is selected, provided it is in the same partition as the network source, and it is clustered with the network source. The hyper-MAMC algorithm is then rerun. By doing this the chosen sink cannot be replicated, and it can now act as a source of flow in the network. This can increase the flow, and reduce the size of the replication set (see Figure 7). This process of incrementing the flow is repeated until the size of the replication set falls within acceptable parameters. Once we have derived a sufficiently small set for replication, we dismantle the flow network, select the opposite partition, and repeat the above.

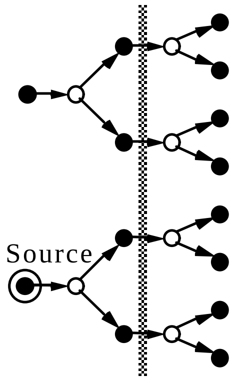

Figure 8. Nodes that cannot be reached from the source, such as those at the top of this figure, will always be replicated.

A peculiarity of this algorithm, which is particularly pronounced when working with small circuit expansions, is that we may exceed partition size limits, yet there may be no node available to cluster with the source for flow incrementing (Figure 8). This is caused by an excessive number of nodes in the active partition which are unreachable from the network source. However, this is a fairly rare event, and when it occurs we default to a directed-FM (see MC-rep below) on the modified network graph.

In the sections that follow we discuss several new techniques for improving the basic Hyper-MAMC algorithm. This will provide a significant quality improvement over the basic algorithm, and produce the highest quality results for any single replication approach.

In the next three subsection we highlight our contributions to the basic MAMC algorithm.

Benchmark	Hyper-MAMC IBUF	Hyper-MAMC RAND 2.0	Hyper-MAMC RAND 1.0	Hyper-MAMC RAND 0.5	Hyper-MAMC RAND 0.25	Hyper-MAMC RAND 0.125
s5378	46	44	42	41	39	40
s9234	31	32	29	29	28	29
s13207	45	46	48	47	46	45
s15850	40	39	39	38	37	37
C6288	34	34	33	33	33	33
biomed	70	92	79	75	73	83
primary2	115	120	112	109	110	110
struct	34	33	33	33	33	33
s35932	39	39	39	39	39	39
s38584	26	28	26	25	25	25
s38417	40	40	37	37	37	38
Geom. Mean	43.2	44.5	42.5	41.8	41.2	42.0
Time	24.7	21.7	22.5	23.4	24.3	21.9

Table 5. Effect of source choice on cut size for Hyper-MAMC. Initial partitions are generated by Strawman with a maximum partition size of 51% of the logic.

5.2.1 Node Choice for Source Clustering

When creating the flow network, we must choose a number of nodes to cluster together to create the network source. The most obvious candidates for source clustering are Input Buffers or IBUFs. These nodes tend to be liberally scattered about both partitions, and presumably every node in the graph is reachable through some IBUF. However, it appears that a random node selection tends to produce better results. This is likely due to the variability in node selection which it introduces, providing the algorithm with an opportunity to get "lucky" with its source choice when iterated multiple times.

When choosing nodes randomly to cluster with the source, we select the number of nodes as a function of initial directed cut size. The initial directed cut size is the number of hyperedges which have their source in the chosen partition, and one or more sinks in the opposite partition. This is simply the number of hyperedges which span both partitions in the newly created flow network. In the worst case, the minimum cut is the initial directed cut size (implying no replication occurred), and therefore the maximum flow can never be higher than this value. It seemed a reasonable convention to choose the number of sources based upon the maximum amount of flow we may have to push through the network. Table 5 details the various source selection strategies. The IBUF column uses all IBUFs in the source partition as flow sources. The other columns randomly select a number of sources equal to X * directed_cut_size, where X is the number listed in the column heading. For example, RAND 0.25 indicates that a random node selection was employed where a total of 0.25 * directed_cut_size nodes were selected. Randomly selecting a number of sources equal to one-quarter the directed cut size produces the best cuts. Therefore all future hyper-MAMC variants in this paper will incorporate this strategy of source selection.

5.2.2 Initial Partition Variation

In the move-based algorithms, size limits were expressed as maximum partition sizes, limiting both partition size imbalance and replication. Thus, with a limit of 53.6% of the logic, one partition could contain 53.6% of the logic and have no replication, or both partitions could contain 46.4% unreplicated logic and 7.2% replicated logic. This ability to trade size imbalance (which allows a standard bipartitioner to produce better results) with replication yields an added optimization opportunity. Hyper-MAMC is a post-processing algorithm, where the partitioning of the circuit, and thus the partition size imbalance, must be fixed before replication begins. In Table 6 we explore how the limits applied to partition size imbalance affect the final results. The results indicate that improving initial partition quality does indeed improve the cuts created by hyper-MAMC, but we must leave the algorithm some room to operate. All future hyper-MAMC variants in this paper will use an initial partition size limit of 52%/48% circuit size.

Benchmark	Strawman	Initial Partition size limit of 49%/51%	Initial Partition size limit of 48%/52%	Initial Partition size limit of 47%/53%
s5378	60	39	40	41
s9234	45	28	31	35
s13207	67	46	44	47
s15850	52	37	38	39
C6288	50	33	33	33
biomed	161	73	61	84
primary2	133	110	113	107
struct	33	33	33	33
s35932	46	39	23	39
s38584	50	25	25	25
s38417	55	37	36	38
Geom. Mean	60.8	41.2	39.1	43.1
Time	17.4	24.3	24.3	21.8

Table 6. Effect of initial partition size limit on Hyper-MAMC.

5.2.3 Node Choice for Incrementing Flow

When faced with an overly large replication set, the hyper-MAMC algorithm will attempt to increment the flow by choosing a node to cluster with the network source. We can attempt to find the best node choice by selecting each node, and determining which node produces the greatest decrease in the size of the replication set, or the smallest increase in maximum flow, or both. Unfortunately this exhaustive node search is generally too time consuming, and therefore the hyper-MAMC algorithm includes a threshold parameter for use with node choice [Yang94]. When incrementing the flow, and choosing from among a threshold number of nodes (or less), we determine the effects of incrementing the flow to each node, and increment the flow to that node which produced the best results (least increase in max-flow, with ties broken by choosing the largest decrease in total size of nodes replicated). If we have greater than a threshold number of nodes to choose from, we arbitrarily choose a single node for source clustering.

As shown in Table 7, small threshold values have little impact on partition quality. Apparently the number of occasions in which the node choice falls below the threshold value are very few on these benchmarks. Unfortunately, using a higher threshold value tends to slow the algorithm considerably. To avoid this performance penalty, we decided to select the initial partition (as produced by Strawman) which produced the best final cut size after performing hyper-MAMC with a size threshold of zero, and then run a high threshold hyper-MAMC on only this initial partition. This amortizes the time cost of a single high threshold hyper-MAMC over the previous thirty iterations of hyper-MAMC.

Benchmark	Strawman	Threshold			Amortized		Random	MC-Rep
		0	25	50	100	200	5	
s5378	60	40	40	40	40	40	39	44
s9234	45	31	31	31	28	28	28	37
s13207	67	44	44	44	44	44	42	61
s15850	52	38	38	38	38	38	38	50
C6288	50	33	33	33	33	33	33	33
biomed	161	61	61	61	58	58	58	127
primary2	133	113	113	113	108	108	106	121
struct	33	33	33	33	33	33	33	33
s35932	46	23	23	23	23	23	23	23
s38584	50	25	25	25	25	25	25	25
s38417	55	36	36	36	36	36	34	44
Geom. Mean	60.8	39.1	39.1	39.1	38.4	38.4	37.8	46.3
Time	17.4	21.9	37.6	26.3	28.1	32.0	59.7	21.7

Table 7. Effect of node choice for incrementing flow on partition quality. Threshold columns try all node choices when the number of choices is less than the threshold, and otherwise randomly pick a single node. Amortized columns run threshold 0, pick the best of 30 runs, and then uses the listed threshold on just that best partitioning. Random 5 always randomly tries 5 nodes each time the flow must be incremented. The MC-Rep column uses MC-Rep, while all the others use Hyper-MAMC.

An alternative to a threshold function is to instead always randomly choose a small number of nodes to test whenever we must increment the flow. In this way we always make a (somewhat) informed decision about what node to add to the source set. Although this greatly increases the processing time, we reduce this time by also amortizing the cost, only optimizing the best of the 30 runs as determined by threshold 0 runs. This techniques is used in the "Random 5" column of Table 7.

5.2.4 Hyper-MAMC Replication Results

The results for the various optimizations to the Hyper-MAMC algorithm are given in Table 5 - Table 7. As can be seen in Table 5, the choice of flow sources can make a significant difference, with randomly picking 1/4 as many sources as the cutsize of the initial partitioning producing a 4.6% improvement over just using the circuit's IBUFs. Balancing the amount of flexibility given to the partitioner verses the amount of replication allowed is also an issue, providing up to a 9.3% difference between 48%/52% and 47%/53% partitioning. Finally, always evaluating five random nodes to add to the source when incrementing the flow is important, providing a 3.3% improvement over just randomly adding nodes. However, this does almost triple the runtime, even though this technique is only applied to the most promising of the 30 partitioning attempts. Combining all of these approaches produces cutsizes 37.8% smaller than the unreplicated version, significantly better than any of the other techniques.

5.3 MC-Rep [Hwang95]

MC-Rep is a simpler version of Hyper-MAMC. Just as in Hyper-MAMC it uses the minimum cut in a flow network to form the replication set (we use Hyper-MAMC's hyperedge to edge transform, instead of MC-Rep's more complex transformation). However, unlike Hyper-MAMC, if the size of the replication set is too large we dismantle the flow network and re-establish the original hypergraph. All nodes which were not in the selected partition are permanently locked, and a directed FM is performed upon the hypergraph. In this directed FM hyperedges are cut only if the source of the hyperedge is in the selected partition (and is unreplicated), and at least one sink is in the opposite partition. We only allow replication and unreplication moves – no standard FM moves are allowed – and only by nodes which were originally in the selected partition (all other nodes are permanently locked). In this way, a replication set is derived which meets the size criterion. We then select the opposite partition, and repeat as above.

We apply the lessons learned from hyper-MAMC and randomly cluster a number of nodes equal to twenty-five percent of the directed cut size to create the network source. We also use Strawman to generate an initial partition with 48%/52% size limits. As shown in Table 7, MC-Rep performs significantly worse than Hyper-MAMC, which is able to stay closer to the basic network flow formulation for unbounded size replication.

6 Combinations

In this section we present several new combinations of replication techniques. These combinations apply several replication techniques together, producing better results than is possible with any of the individual algorithms alone.

DFRG, KN and Hyper-MAMC each approaches the replication problem from a different angle, and therefore there is every reason to believe that a combination of theses algorithms may produce better results than any single algorithm. The first combination tried was KN with DFRG. Since the DFRG performs quite well with clustering, at each clustering level of the Strawman algorithm the Strawman FM was replaced with KN followed by DFRG. This is essentially the DFRG as extended to clustering, but at each level the partition is optimized prior to DFRG processing.

Benchmark	Strawman	KN Gradient	DFRG w/ Clustering	Hyper-MAMC RAND 5	KN, DFRG	Hyper-MAMC, KN, DFRG
s5378	60	43	41	39	39	38
s9234	45	32	32	28	24	26
s13207	67	43	63	42	52	39
s15850	52	41	45	38	38	37
C6288	50	33	33	33	33	33
biomed	161	130	107	58	113	60
primary2	133	114	121	106	101	100
struct	33	33	33	33	33	33
s35932	46	23	24	23	23	23
s38584	50	32	29	25	31	25
s38417	55	49	45	34	44	35
Geom. Mean	60.8	44.7	44.9	37.8	42.0	37.2
Time	17.4	19.7	48.2	59.7	39.1	34.1
Replication	0.0%	3.5%	5.1%	4.6%	5.3%	5.1%

Table 8. Combinations of replication techniques.

The KN and DFRG combination, while better than either algorithm alone, could not match the performance of hyper-MAMC. The second combination was hyper-MAMC, KN, and DFRG. The standard hyper-MAMC algorithm (without random node choice for incremental flow since this significantly slows the algorithm) produces the original replication set, and then each of the KN and DFRG algorithms attempt to improve upon this base. The initial partition was generated by Strawman with 48%/52% partition size limits.

Results for these combined algorithms are shown in Table 8. Combining KN with DFRG does do better than either algorithm by itself (doing 6% better than KN and 6.5% better than DFRG alone), but does 11% worse than Hyper-MAMC. Combining all three together does do 1.6% better than Hyper-MAMC alone, and requires about half the time to complete. However, whether the small quality gains is worth the significant code complexity increases is unclear.

7 Circuit Expansion

Having found that hyper-MAMC produces the best cutsizes, we use our optimized hyper-MAMC algorithm with random node selection to determine the effects of increased circuit expansion on cut size. Each data point in the graph in Figure 9 was derived by choosing the best of ten runs of our optimized hyper-MAMC algorithm at the selected replication, and computing the geometric mean across all of the circuits in our benchmark suite. Thus, the righmost point allows up to 50% of the logic to be replicated, while the leftmost point allows no replication at all.

8 Conclusion

In this paper we have investigated logic replication, seeking to develop the best possible logic bipartitioning results. In the process we were able to reduce cutsizes by 38.8% over Strawman's already impressive results (though the 37.8% achieved by just our optimized version of Hyper-MAMC is probably more realistic than the 38.8% from combining Hyper-MAMC, KN, and DFRG because of code complexity). Also, these significant cutsize improvements are obtained with relatively little logic duplication, with most techniques replicating less than 5% of the logic.

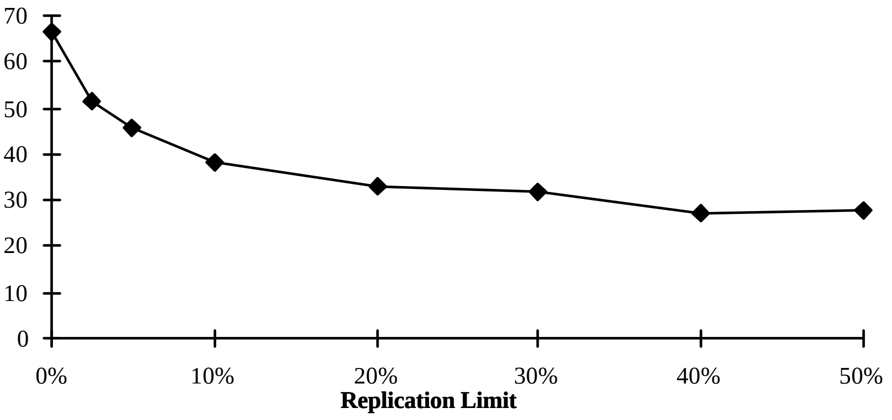

Figure 9. Graph of cutsize for various restrictions on allowed circuit expansion. Values are the geometric mean cutsize for the best of ten runs for each of the benchmarks

The replication techniques in the literature are not so much partitioning algorithms in their own right, but are optimizations to be added into an existing partitioning system. Thus, in order to perform a comparison between algorithms we have implemented most of the replication algorithms from the literature within our Strawman system. Thus, we can directly compare all of the algorithms on the same benchmarks, with the same size constraints, using the same set of bipartitioning engines. The only difference is the replication techniques. The comparison is shown in Table 9, which includes all of the replication algorithms discussed in this paper. Results are included both for comparing the optimized replication techniques against non-replicating Strawman, as well as the new optimized replication techniques proposed in this paper against the basic versions directly from the literature. As can be seen, our optimized algorithms are able to achieve a 12.5% to 13.5% improvement over the existing replication techniques even factoring out the benefits of our Strawman bipartitioner.

Achieving these high-quality results has required many innovative optimizations to the individual algorithms. To improve the Kring/Newton algorithm we extended the algorithm to take advantage of higher-level gains and recursive clustering. We also developed the Gradient method to delay replication until the latter stages of partitioning, avoiding early, ignorant replication moves. Combined together, this improves the results by 7.3%. To improve the DFRG algorithm we developed a directed graph partitioning formulation for higher-level gains and extended the algorithm to work under recursive clustering. This achieved a 8.2% improvement over the basic DFRG algorithm. Finally, we improved the Hyper-MAMC algorithm by carefully considering which nodes should be used as the sources for the flow network, by varying the amount of size imbalance allowed for bipartitioning verses replication, and by developing efficient methods for deciding which nodes to use to add to the set of flow sources, and thus reduce the replication set. This provides a 12.5% improvement over the basic Hyper-MAMC algorithm, and generates the best results for any single replication technique.

Replication Algorithm	Cutsize	Improvement over Strawman	Improvement over Basic Algorithm	Time (total minutes for 30 runs)
Strawman	60.8	---	---	17.4
KN	48.2	20.7%	---	17.3
higher-level gains	47.7	21.5%	1.0%	17.6
clustering	47.6	21.7%	1.2%	17.1
gradient	44.7	26.5%	7.3%	19.7
DFRG	48.9	19.6%	---	27.8
higher-level gains	48.2	20.7%	1.4%	31.6
clustering	44.9	26.2%	8.2%	48.2
Hyper-MAMC	43.2	28.9%	---	24.7
flow source node choice	41.2	32.2%	4.6%	24.3
partition variation	39.1	35.7%	9.5%	24.3
flow incr. node choice	37.8	37.8%	12.5%	59.7
MC-Rep	46.3	23.8%	---	21.7
Combined Approaches				
KN, DFRG	42.0	30.9%	8.7%	39.1
Hyper-MAMC, KN, DFRG	37.2	38.8%	13.9%	34.1

Table 9. Summary of optimizations. Basic algorithms are shown in bold, followed by the new optimizations presented in this paper. "Improvement over Strawman" represents how much each replication technique improves over a non-replicated bipartitioning. "Improvement over Basic Algorithm" is how much better the optimized version does as compared to the unoptimized replication technique (i.e. the version taken directly from the literature). Combined algorithms are compared against the basic version of the best replication technique it contains. All algorithms are allowed to generate partitions containing at most 53.6% of the logic through both bipartitioning imbalance and replication. Functional Replication does not appear here because it can only operate on a subset of our the benchmark suite. It produces a 6% improvement over Strawman, and replicated 2.9% of the logic.

We also investigated combined approaches which use multiple replication techniques together. By integrating KN with DFRG we achieved a 12.9% - 14.1% improvement over the basic algorithms and a 6% - 6.5% improvement over the optimized versions. By combining Hyper-MAMC, KN, and DFRG together we achieved a 13.9% - 23.9% improvement over the basic algorithms, but only a 1.6% improvement over optimized Hyper-MAMC, and thus this combination may not be worth the large added code complexity.

As has been demonstrated in this paper, achieving the greatest possible partitioning quality requires the careful consideration not only of individual techniques, but also how these techniques fit together. Optimizations such as higher-level gains, multiple random initializations, and recursive clustering may be key to achieving the best standard bipartitioning results, but integrating replication techniques into such a system can cause significant added values. Also, subtle issues such as node choice for flow sources and incrementing flows can easily be overlooked, yet can provide significant quality improvements. Thus we believe that as well continue to add to the repertoire of replication techniques, we must also consider how these techniques can be added to what is already known to produce the best overall algorithm.

References

[Alpert95a] C. J. Alpert, A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey", *Integration: the VLSI Journal*, Vol. 19, No. 1-2, pp. 1-81, 1995.

[Alpert95b] C. J. Alpert, S.-Z. Yao, “Spectral Partitioning: The More Eigenvectors, The Better”, *Design Automation Conference*, pp. 195-200, 1995.

[Cheng88] C. K. Cheng, T. C. Hu, “Maximum Concurrent Flow and Minimum Ratio-cut”, *Technical Report CS88-141*, University of California, San Diego, December, 1988.

[Cormen90] T. H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms*, MIT Press/McGraw-Hill, Cambridge, MA, 1990.

[Dutt96] S. Dutt, W. Deng, “A Probability-Based Approach to VLSI Circuit Partitioning”, *Design Automation Conference*, pp. 100-105, 1996.

[Enos96] M. Enos, M.S., “Replication for Logic Bipartitioning”, Master’s Thesis, Northwestern University, Department of ECE, 1996.

[Fiduccia82] C. M. Fiduccia, R. M. Mattheyses, “A Linear-Time Heuristic for Improved Network Partitions”, *Design Automation Conference*, pp. 241-247, 1982.

[Ford62] L.R. Ford, D.R. Fulkerson, *Flows in Networks*, Princeton Univ. Press, Princeton, NJ, 1962.

[Garey79] M. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco, CA: Freeman, 1979.

[Hagen92] L. Hagen, A. B. Kahng, “New Spectral Methods for Ratio Cut Partitioning and Clustering”, *IEEE Transactions on Computer-Aided Design*, Vol. 11, No. 9, pp. 1074-1085, Sept. 1992.

[Hauck95a] S. A. Hauck, *Multi-FPGA Systems*, Ph.D. dissertation, University of Washington, Dept. of CSE., pp. 131-168, 1995.

[Hauck95b] S. Hauck, G. Borriello, “An Evaluation of Bipartitioning Techniques”, *Chapel Hill Conference on Advanced Research in VLSI*, pp. 383-402, March, 1995.

[Hauck97] S. Hauck, G. Borriello, “An Evaluation of Bipartitioning Techniques”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, No. 8, pp. 849-866, August 1997.

[Hwang95] L. J. Hwang, A. El Gamal, “Min-Cut Replication in Partitioned Networks”, *IEEE Transactions on Computer-Aided Design*, Vol. 14, No. 1, pp. 96-106, Jan 1995.

[Kring91] C. Kring, A. R. Newton, “A Cell-Replicating Approach to Mincut-Based Circuit Partitioning”, *International Conference on Computer-Aided Design*, pp. 2-5, 1991.

[Krishnamurthy94] B. Krishnamurthy, “An Improved Min-Cut Algorithm for Partitioning VLSI Networks”, *IEEE Transactions on Computers*, Vol. C-33, No. 5, pp. 438-446, May 1984.

[Kuznar94] R. Kuznar, F. Brglez, B. Zajc, “Multi-way Netlist Partitioning into Heterogeneous FPGAs and Minimization of Total Device Cost and Interconnect”, *Design Automation Conference*, pp. 238-243, 1994.

[Liu95a] L. Liu, M. Kuo, S. Huang, C. K. Cheng, “A Gradient Method on the Initial Partition of Fiduccia-Mattheyses Algorithm”, *International Conference on Computer-Aided Design*, pp. 229-234, 1995.

[Liu95b] L. Liu, M. Kuo, C. K. Cheng, T. C. Hu, “A Replication Cut for Two-Way Partitioning”, *IEEE Transactions on Computed-Aided Design*, Vol. 14, No. 5, May 1995, pp. 623-632.

[Reiss94] G. M. Riess, K. Doll, F. M. Johannes, “Partitioning Very Large Circuits Using Analytical Placement Techniques”, *Design Automation Conference*, pp. 646-651, 1994.

[Yang94] H. H. Yang, D. F. Wong, “Efficient Network Flow Based Min-Cut Balanced Partitioning”, *International Conference on Computer-Aided Design*, pp. 50-55, 1994.

[Yang95] H. H. Yang, D. F. Wong, “New Algorithms for Min-Cut Replication in Partitioned Circuits”, *International Conference on Computer-Aided Design*, pp. 216-222, 1995.