Partitioning Under Timing and Area Constraints

Gregory Tumbush and Dinesh Bhatia
Design Automation Laboratory
Department of ECECS
University of Cincinnati
Cincinnati, OH 45221–0030

Abstract

Circuit partitioning is a very extensively studied problem. In this paper we formulate the problem as a nonlinear program (NLP). The NLP is solved for the objective of minimum cutset size under the constraints of timing. Our proposed methodology easily extends to multiple constraints that are very dominant in the design of large scale VLSI Systems. The NLP is solved using the commercial LP/NLP solver MINOS. We have done extensive testing using large scale RT level benchmarks and have shown that our methods can be used for exploring the design space for obtaining constraint satisfying system designs. We also provide extensions for solving system design problems where a choice between multiple technologies, packaging components, performance, cost, yield, and more can be the constraints for design related decisions.

1 Introduction

Ever changing complexity of VLSI systems requires support from CAE tools for automated decision making capability. Also, important design related decisions should be made early in the design process. This requires tools that have the capability to explore design choices, make tradeoffs between various constraints, and select/reject design options so as to obtain a very high quality constraint satisfying solution. Motivated with this task of automating the system design process we have conducted this research for system level partitioning problems.

In system level partitioning, a designer is presented with an application (design), a set of requirements, a set of options for realizing the design, and a set of constraints for implementing or physically realizing the overall design. In a typical design such parameters would include choice of packaging options, i.e., ICs from various technologies, their area and pin constraints, their costs, timing requirements on the overall design, yield, testability, and more. In the presence of such choices the designer must try to optimize the resources such that the final design implementation satisfies as many constraints as possible.

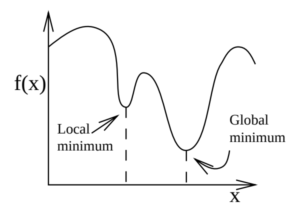

Figure 1. *Local and Global Optimum*

A large design can be implemented using one large ASIC chip, or a collection of devices packaged in a hierarchical manner. When the later is chosen, the design has to be partitioned into two or more segments and implemented using correct packages. Partitioning of a design in the presence of multiple constraints is an important and hard combinatorial problem. When the constraint set starts becoming large, it is very difficult to make correct design decisions. In this paper, we have modeled the problem of partitioning in the presence of multiple constraints as a non-linear programming problem and have presented effective solutions for partitioning designs in the presence of *area* and *timing* constraints.

1.1 Combinatorial Optimization

Generally a combinatorial optimization (CO) problem is an optimization problem of the form

$$\begin{aligned} &minimize \quad f(x) \\ &subject\ to \quad g_i(x) \leq 0 \;\; i = 1, \ldots, m. \end{aligned} \tag{1}$$

The function $f(x)$ is the objective function and the set of conditions $g_i(x) \leq 0$, $i = 1, \ldots, m$, are the constraints of the problem. Note that the number of constraints can be very large. Every vector x that satisfies the constraints is a *solution* to the problem. A solution that minimizes $f(x)$ over the set of all solutions is an *optimal solution*. A vector x' is a *local optimum* if and only if there exists a neighborhood $V(x')$ of x' such that x' is a *global optimum* of the problem. Figure 1 illustrates this concept with a function of a single variable[12].

The forms that $f(x)$ and $g_i(x)$ take determine the type of CO problem. If $f(x)$ is *linear*, the problem is a linear program(LP), while if $f(x)$ is *non-linear*, the problem is a

Proceedings of the 1997 International Conference on Computer Design (ICCD '97)
0-8186-8206-X/97 $10.00 © 1997 IEEE

non-linear program(NLP). $f(x)$ may have both linear and non-linear elements. The constraints $g_i(x)$ can also be linear, non-linear, or a combination of both. A special case of non-linear CO is when the terms of $f(x)$ are quadratic and the constraints are linear. This is called a *quadratic program*(QP). If $f(x)$ does not exist and only constraints are present the problem becomes a *constraint satisfiability problem* (CSP). Equations 2 and 3 show the forms of linear and quadratic programs respectively.

$$minimize \quad c^T x \quad subject\ to\ Ax \le b \tag{2}$$

$$minimize \quad \frac{1}{2} x^T D x + c^T x\ subject\ to\ Ax \le b \tag{3}$$

In equations 2 and 3, c^T is the transpose of the coefficients of the linear optimization function, x is the solution variables, A is the constraint matrix of the linear constraints, b is the right hand side of the linear constrains, D is the coefficient matrix of the quadratic optimization function, and x^T is the transpose of x.

Many fast methods exist to solve an LP[12] as do methods to solve a NLP if it is *convex*, that is, every local optimum is also a global optimum. However, there are no known methods to find a global optimum for a non-convex NLP problem. Only a local optimum is guaranteed to be found in this case. The graph bipartitioning problem when formulated as a CO problem is non-convex.

We solve our problem with the commercial LP/NLP solver MINOS 5.4. MINOS can solve large scale linear and non-linear programs and takes advantage of sparsity of matrices[13].

2 Partitioning Related Research

The partitioning problem with area constraints falls into the class of NP-complete problems [2]. Various heuristic approaches have been proposed.

Johannes[8] gives an overview of the partitioning problem. He divides partitioning algorithms into five categories: bipartitioning, k-way partitioning, performance driven partitioning, layout driven partitioning, and partitioning with replication. Furthermore, the solution techniques can be classified as constructive or iterative and deterministic or probabilistic.

The Kernighan and Lin (KL) algorithm [9] is a popular *iterative improvement* bipartitioning algorithm. Dutt[2] introduces a method called *Quick_Cut* that reduces the number of node pairs examined in the KL algorithm. Quick_cut searches on only d^2 node pairs to find the greatest swap gain where d is the max degree of any node in the graph. The complexity of Quick_cut is $O(e \cdot logn)$ where n is the number of nodes and e is the number of edges.

The Fiduccia and Mattheyses (FM) heuristic [5] will perform bipartitioning on nets by representing the circuit as a hypergraph. The runtime complexity of FM method is linear in the size of number of pins in the VLSI circuit. Iterative improvement techniques such as KL and FM perform well for small to medium size circuits but will produce increasingly poor results as the problem size rises. Clustering techniques attempt to reduce the problem size such that they can be efficiently solved by FM or KL.

Hagen and Kahng[6] introduce a new method for clustering circuits to reduce the number of nodes that require consideration called $RW-ST$. After clustering, FM is executed on the reduced circuit. The $RW-ST$ methodology computes a circuit clustering based on a random walk in the netlist graph. A cycle identified in the random walk should correspond to a natural cluster. After all cycles have been identified the $sameness$ of each pair of nodes u and v is determined. The $sameness$ measure determines for every node v how often node u occurs in a cycle originating at v. If a node pair has a $sameness > 0$ the nodes become a cluster. The authors of [6] introduce a clustering quality measure, DS (degree/separation), where $degree$ is the average number of nets incident to each module of the cluster and having at least two pins in the cluster and $separation$ is the average length of a shortest path between two nodes in the clustering, infinity if not connected.

Using the DS measure the RW-ST method is compared against the matching based compaction(MBC) method of[1]. It was found that the clustering produced by RW-ST had over 30% better DS qualities than those produced by MBC for large circuits. To further evaluate the quality of the clusters produced by RW-ST, FM was run on the resulting clustering graphs. MBC was found to produce very poor cutset results compared to that of RW-ST. The final experiment consisted of using the partition results obtained above as an initial partition and rerunning FM on the entire unclustered netlist. It was found that MBC produced cutset results on average 12% better than FM while RW-ST produced cutset results 17% better than FM. Interestingly, vastly superior initial partitions produced by RW-ST did not translate to a large improvement in the final partitions.

An algorithm by Huang et. al.[11] also strives to find a good initial partition for the FM algorithm. The partitioning problem is formulated as a QP problem with linear constraints. Obviously, the resulting assignments must be integer. A solution is obtained from the QP by removing all constraints and solving the QP using a gradient decent algorithm. The algorithm is called GFM, or GFM_r if cell replication is allowed.

Let $t^h = (t^h_{1,1}, \ldots, t^h_{1,n}, \ldots, t^h_{K,1}, \ldots, t^h_{K,n})$ be the listing of assignments where K is the number of partitions, n is the number of cells, and h is the present iteration number. It is unlikely that the values of t^h will be integer or meet the constraints. Therefore, solve $max \sum_{i \le b \le K} \sum_{1 \le i \le n} x_{b,i} t^h_{b,i}$ subject to the size constraints of each partition where $x_{b,i}$ is the assignment variable of cell i to partition b. If replication of cells is not allowed there is an additional size constraint of $\sum_{b=1}^{K} x_{b,i} =$

1 $\forall node i \in V$ where V is the set of nodes. After a solution for $x_{b,i}$ is found, FM is applied to generate a better solution. If another iteration of the algorithm is desired a new gradient is determined and the process repeats. GFM shows improvements of 15% over PARABOLI[14] in terms of cutset size.

Another method to improve upon iterative methods, called $CLIP$ (CLuster oriented Iterative improvement Partitioner)is introduced in[4]. CLIP alters the way in which cell gains are determined in the method, encouraging consideration of cells connected to recently moved cells. This promotes the movement of an entire densely-connected cluster into one partition.

A new technique for choosing the most productive cell to move is introduced in[3] and is called PROP (PRObalistic Partitioner). PROP is based on the assumption that a number of nodes will have similar gain values and a tie should be broken by considering the $potential$ gain associated with each node. That is, the decrease in cutset that is not immediately realized but has a good chance of occurring in future moves.

Partitioning using analytical placement techniques is proposed in a paper by F.M. Johannes, et al[14]. This technique, called *PARABOLI*, solves a one dimensional placement problem that has a linear objective function $min \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}|x_i - x_j|$, where n is the number of cells, a_{ij} is the sum of the edge weights connecting edge weights connecting cells i and j. After the placement solution is obtained the *Ratio Cut* (RC), is determined for every possible cut position between two cells. The RC is found by $RC = \frac{C_{LR}}{|L||R|}$ where C_{LR} is the cutset size of the two partitions and $|L|$ and $|R|$ are the set size of each partition. The minimum of RC is the *Minimum Ratio Cut* MRC and produces the optimal partition results.

A circuit modeled as a network flow problem can be partitioned using max-flow min-cut techniques. This method will find a partition, not necessarily balanced, in polynomial time. Repeatedly applying the max-flow technique will produce a balanced bi-partition but it may take as many iterations as number of nodes[7].

Solving the k-way partitioning problem by using integer programming is proposed by Kuh, et. al.[17]. A cost function p_{ij} representing the cost of assigning module j to partition i is minimized according to timing and capacity constraints. Solving the k-way partitioning problem where the target device is known is presented by Sawka and Thomas[16]. They use a set cover based approach (SCP) to achieve a multiway partition for look up table (LUT) based FPGA's.

3 Partitioning Under Timing and Area Constraints

Given a set of n netlist modules $V = \{v_0, v_2, \ldots, v_{n-1}\}$ we represent the circuit as a hypergraph $G = (V, E)$ with

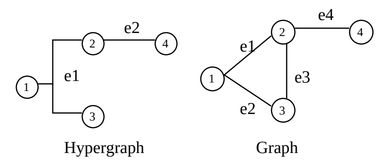

Figure 2. *Hypergraphs and Graphs*

n vertices and a set $E \subset 2^V$ of $hyperedges$ or $nets$. The number of hyperedges $|E| = m$. The vertices in a hyperedge $e \in E$ are called $terminals$ of e. $|e|$ denotes the cardinality of a hyperedge e. If all $e \in E$ have a cardinality of 2, i.e. $|e| = 2$, then G is a $graph$. Figure 2 shows a hypergraph and it's representation as a graph.

Given a set of n netlist modules the goal of partitioning is to assign each v_i $i = 0, \ldots, n-1$ to a specified number k of segments. If $k = 2$, the problem becomes that of graph bipartitioning. An edge e is cut if all the terminals of e are not within a single segment. The total number of cut edges is called the size of $cutset$. For Figure 2, if terminals 1 and 2 are in one segment and terminals 3 and 4 are in another the cutset is two for the hypergraph and three for the graph. Typically one chooses to minimize the size of cutset according to some pre-defined criteria. In this paper we perform hypergraph bipartitioning under timing and area constraints.

The input to the partitioner is a netlist and the area of each netlist component. The optimized function is an exact expression of the hypergraph cutset size. We optimize the cutset size according to timing and capacity, i.e. area constraints. The timing constraints are derived from the T critical timing paths.

The CO problem is solved as an assignment problem. We associate a variable $x_i, 0 \leq i \leq n-1$ for n components. It is predetermined for bipartitioning that if $x_i = 1$ then module i belongs to a partitioning segment and to the complimentary one if $x_i = 0$. A solution to the NLP problem can result in non-integer assignment to x_i which will not form a feasible partitioning solution. Thus, fractional assignment variables have to be rounded for generating a feasible partitioning solution. We employ 0-1 rounding for changing the fractional assignments to an integer form. This can be done simply by choosing a value, $median$, and if $x_i \geq median$ set x_i to 1, 0 otherwise. Other methods such as randomized rounding[15] can be employed to intelligently round the assignment variables. Given a fractional assignment variable, $x_i = p$, randomized rounding will round this variable to 1 with a probability p.

3.1 Partitioning: Problem and Solution

Consider a three cell net, 1, 2, and 3 , let $X = 1$ if the connection between cell 1 and cell 2 is cut, 0 otherwise and $Y = 1$ if the connection between cell 2 and cell 3 is cut, 0 otherwise. An exact expression for the hypergraph cutset size of this net requires a logical expression, $X + Y$. Using DeMorgan's Theorem this expression becomes

$\overline{\overline{X} \cdot \overline{Y}}$. Since $X \in \{0,1\}$ and $Y \in \{0,1\}$, this equation is *numerically* equal to:

$$1-(1-X)(1-Y) \tag{4}$$

Given the assignment of these three components, x_1, x_2, x_3, X and Y is expressed as:

$$\begin{aligned} X &= |x_1 - x_2| = x_1(1-x_2) + x_2(1-x_1) \\ Y &= |x_2 - x_3| = x_2(1-x_3) + x_3(1-x_2) \end{aligned} \tag{5}$$

Combining equation 4 and 5 results in the exact expression for hypercutset size between three components:

$$\begin{aligned} &1-(1-x_1(1-x_2)+x_2(1-x_1)) \cdot \\ &(1-x_2(1-x_3)+x_3(1-x_2)) = \\ &x_1+x_2+x_3-x_1x_2-x_1x_3-x_2x_3 \end{aligned} \tag{6}$$

In general, the hypercutset size of a circuit is :

$$\sum_{\forall r \in M} (\sum_{i=1}^{|Q_r|-1} (-1)^{i+1} C_i^{Q_r} - 2F \prod_{j=1}^{|Q_r|} x_j) \tag{7}$$

where Q_r is the set of assignment variables for all non I/O components on net r, F equals 1 if $|Q_r|$ is even, 0 otherwise, M is the set of nets, and $C_i^{Q_r}$ is the combinations of the set Q_r taken i at a time. As with any partitioning problem formulation, minimizing the *cutset* size is the most important objective for our formulation. Note that for net r with $|Q_r|$ non I/O components, an expression with $2^{|Q_r|}$ terms is required to fully express the hypergraph cutset size for bipartitioning. A typical VLSI circuit contains majority of nets that are small, i.e., two to four terminals. Hence, in our implementation, for very large nets, we drop out the terms in the above mentioned expression. However, we always account for an extra (possible) cut in our cutset size evaluation process.

3.2 Timing Constraints

In addition to minimizing the cutset, we also consider the timing constraints. In order to formulate timing constraints, we consider a set of *critical* paths. In practice such a constraint can be user defined. However, for our solution, we evaluate the first T longest paths in the given circuit. The T longest paths are found using Kundu's longest path algorithm [10]. This algorithm performs a levelized forward traversal of nodes with a merge sort of delay values, followed by a backward trace to identify T longest paths. All output cells are connected to a $pseudonode$ for this purpose. The delay values on each edge is dependent on three factors: fanout from the source cell, delay of the source cell, and type of the source and destination cell. The source and destination cell can be an input, output, or internal cell.

Let $delay_j$ be the delay of internal or I/O cell j, o_j the fanout to output cells, i_j the fanout to internal cells, β the delay due to driving an output cell, μ the delay due to driving an internal cell, and C the timing penalty for an edge leaving the chip. The delay from input to output is $delay_j + o_j\beta + i_j\mu$, from input to internal cell or internal cell to input is $delay_j + o_j\beta + i_j\mu + C$, and from internal cell to internal cell is $delay_j + o_i\beta + i_j\mu +$ 2C if the edge (i,j) is cut, 0 otherwise. Therefore the only variable in the critical path delay is the cutset of internal edges on the T critical paths.

Let x_{source} and x_{sink} be the assignment of internal source and sink cells. The timing penalty for an edge between the $source$ and $sink$ being cut is $2C(x_{source} + x_{sink} - 2x_{source}x_{sink})$. In general, the t'th, $1 \le t \le T$, timing constraint is

$$D_t + \sum_{\forall (i,j) \in E_t} 2C(x_i + x_j - 2x_ix_j) \le Time_t \tag{8}$$

where D_t is the delay on critical path t neglecting cut edges, E_t is the set of ordered pairs of edges traversed containing non I/O cells on critical path t, and $Time_t$ is the maximum delay allowed on critical path t. If $D_t \le Time_t < D_t + 2C$ no edge on critical path t can be cut while if $Time_t < D_t$ the timing constraint cannot be met.

3.3 Area Constraints

Let $a_i, i = 0, \ldots, n-1$ be the area of cell i. For bipartitioning, the area constraint on chip 1 and 2 is

$$\sum_{i=0}^{n-1} a_ix_i \le A_1 \ and \ \sum_{i=0}^{n-1} a_i(1-x_i) \le A_2 \tag{9}$$

where A_1 and A_2 are the capacity constraints on two partitioning segments, 1 and 2. Note that A_1 and A_2 are not necessarily the same.

3.4 Example

A short example is presented to illustrate these concepts. The structure of the example circuit is in Figure 3. We determine the cutset size, f, of this circuit from equation 7.

$$\begin{aligned} &f = \sum_{i=1}^{2} (-1)^{i+1} C_i^{\{x_0,x_1,x_2\}} + \\ &\sum_{i=1}^{2} (-1)^{i+1} C_i^{\{x_1,x_2,x_3\}} + \sum_{i=1}^{1} (-1)^{i+1} C_i^{\{x_2,x_4\}} - \\ &2x_2x_4 + \sum_{i=1}^{1} (-1)^{i+1} C_i^{\{x_3,x_5\}} - 2x_3x_5 + \\ &\sum_{i=1}^{1} (-1)^{i+1} C_i^{\{x_4,x_5\}} - 2x_4x_5 + \sum_{i=1}^{1} (-1)^{i+1} C_i^{\{x_4,x_5\}} \\ &-2x_4x_5 + \sum_{i=1}^{1} (-1)^{i+1} C_i^{\{x_5,x_6\}} - 2x_5x_6. \end{aligned} \tag{10}$$

Expanding equation 10 results in the optimization function in equation 11.

$$\begin{aligned} &min : x_0 + x_1 + x_2 - x_0x_1 - x_0x_2 - x_1x_2 + \\ &x_1 + x_2 + x_3 - x_1x_2 - x_1x_3 - x_2x_3 + x_2 + \\ &x_4 - 2x_2x_4 + x_3 + x_5 - 2x_3x_5 + x_4 + x_5 - \\ &2x_4x_5 + x_4 + x_5 - 2x_4x_5 + x_5 + x_6 - 2x_5x_6 \end{aligned} \tag{11}$$

Table 1. Example Circuit Delay Values

Src	Sink	Edge Delay
In2	0	$delay_{In2} + C + 1\mu$
In1	4	$delay_{In1} + C + 2\mu + \beta$
In1	5	$delay_{In1} + C + 2\mu + \beta$
In1	Out3	$delay_{In1} + 2\mu + \beta$
0	1	$delay_0 + 2\mu$+2C if edge (0,1) is cut
0	2	$delay_0 + 2\mu$+2C if edge (0,2) is cut
1	2	$delay_1 + 2\mu$+2C if edge (1,2) is cut
1	3	$delay_1 + 2\mu$+2C if edge (1,3) is cut
2	4	$delay_2 + \mu$+2C if edge (2,4) is cut
3	5	$delay_3 + \mu$+2C if edge (3,5) is cut
4	5	$delay_4 + \mu$+2C if edge (4,5) is cut
5	6	$delay_5 + \mu + \beta$+2C if edge (5,6) is cut
5	Out2	$delay_5 + \mu + \beta + C$
6	Out1	$delay_6 + \beta + C$
Out1	$pseudo$	$delay_{Out1}$
Out2	$pseudo$	$delay_{Out2}$
Out3	$pseudo$	$delay_{Out3}$

To determine the T critical paths, we find the delay on every edge. The resultant delay table appears in Table 1.

Let $delay_i = 1, i = 0, \ldots, 6$, the delay of I/O cells equal 5, C=2, β=5, and μ=0.5. Neglecting cutset size the two longest paths are $pseudo$ ⇒Out1⇒6⇒5⇒4⇒2⇒1⇒0⇒In2 with a delay of 28.0 and $pseudo$ ⇒Out1⇒6⇒5⇒3⇒1⇒0⇒In2 with a delay of 26.5. Utilizing equation 8 with $D_1 = 28.0$ and $D_2 = 26.5$ results in constraint 1, C1, and constraint 2, C2 where $Time_1$ and $Time_2$ is the timing constraint on critical path 1 and 2 respectively.

$$C1 : Time_1 \geq 28.0 \ + 2C(x_6 + x_5 - 2 * x_6 x_5) + 2C(x_5 + x_4 - 2 * x_5 x_4) + 2C(x_4 + x_2 - 2 * x_4 x_2) + 2C(x_2 + x_1 - 2 * x_2 x_1) + 2C(x_1 + x_0 - 2 * x_1 x_0) \quad (12)$$

$$C2 : Time_2 \geq 26.5 \ + 2C(x_6 + x_5 - 2 * x_6 x_5) + 2C(x_5 + x_3 - 2 * x_5 x_3) + 2C(x_3 + x_1 - 2 * x_3 x_1) + 2C(x_1 + x_0 - 2 * x_1 x_0)$$

Letting $a_i = 2, i = 0, \ldots, 6$, we utilize equation 9 to produce the area constraint for chip 1, C3, and chip 2, C4.

$$C3 : 2 \cdot (x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6) \leq A_1 \quad (13)$$

$$C4 : \quad 2(1 - x_0) + 2(1 - x_2) + 2(1 - x_3) + 2(1 - x_3) + 2(1 - x_4) + 2(1 - x_5) + 2(1 - x_6) \leq A_2$$

4 Experimental Results

All code is written in C++ and fortran and compiled using g++ and f77, respectively. MINOS is written in fortran. All benchmarks were tested on a Sparc 20 with 32 MB of RAM running at 60MHz.

We partition six RT level benchmarks generated from behavioral VHDL descriptions. representing the structure of

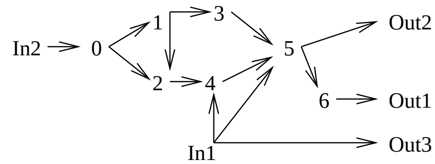

Figure 3. *Example Circuit*

six large circuits. We consider the ten most critical paths for all benchmarks whose characteristics are in Table 2.

The results of bi-partitioning with timing and area constraints are in Tables 3 - 8. Table 9 shows the results of bipartitioning with area constraints only. To compare the partition quality of our method, results using the Fiduccia-Mattheyses algorithm are also found in Table 9 where the benchmark is first tested with the tight area constraint and then the relaxed area constraint. The area constraint is $\frac{\sum_{i=1}^{n} a_i}{2} \cdot 1.05$ for tests one thru four and $\frac{\sum_{i=1}^{n} a_i}{2} \cdot 1.1$ for tests five thru eight. The columns headings in Table 2 - 9 are:

- Benchmark - The name of the benchmark circuit
- Total Area - Combined area of cells in the benchmark in square microns
- Number Cells - Total number of cells in the benchmark
- Number Nets - Total number of nets in the benchmark
- Area Const. - Area constraint considered by MINOS
- C - Number of cut edges allowed on each K longest paths
- Exit Condition - The exit condition reported by MINOS.
  - Optimal - All constraints were met.
  - Infeas. - The problem is infeasible.
  - No-Imp. - The current point cannot be improved upon.
- Run Time(sec) - The user + system CPU time in seconds required by MINOS to solve the problem.
- Cutsize - The cutset size of the hypergraph representation after rounding
- FM Cutsize - Cutset size as determined by the Fiduccia-Mattheyses algorithm in 1 run.
- Max Cut- The maximum number of cut edges on critical path $1, \ldots, 10$

Table 2. Benchmark Characteristics

Bench Mark	Total Area	Number Cells	Number Nets
TLC	2206942	33	93
decompress	2972054	35	164
compress	3267322	37	186
find	7858374	60	285
fifo	20628509	51	584
viper	25471959	81	792

Table 3. TLC Results

Test #	Area Const.	C	Exit Condition	Run Time	Cut Size
1	1158645	0	Optimal	0.7	12
2	1158645	1	Optimal	0.8	11
3	1158645	2	Optimal	0.8	13
4	1158645	3	Optimal	0.7	11
5	1213818	0	Optimal	0.7	12
6	1213818	1	Optimal	0.7	12
7	1213818	2	Optimal	0.7	13
8	1213818	3	Optimal	0.7	12

Table 4. Decompress Results

Test #	Area Const.	C	Exit Condition	Run Time	Cut Size
1	1560328	0	Optimal	0.8	10
2	1560328	1	Optimal	0.8	6
3	1560328	2	Optimal	0.9	6
4	1560328	3	Optimal	0.9	6
5	1634630	0	Optimal	0.8	6
6	1634630	1	Optimal	0.8	6
7	1634630	2	Optimal	0.7	6
8	1634630	3	Optimal	0.8	6

4.1 Analysis

Intuitively, one would think that a relaxed area or timing constraint would allow a more optimized cutset. In general this is true for the test cases presented, but there are exceptions. When dealing with non-linear optimization functions and constraints it is quite possible for the NLP tool to stop in a local minimum. Different constraints and optimization functions produce different search directions and therefore different local minima.

For tests one thru four, i.e. the most restrictive area constraint, restricting the critical path constraint to 0 cuts resulted in three out of the six benchmarks achieving non-optimal results. When the critical path constraint is relaxed to 1 cut per path four benchmarks achieved optimal results. Further relaxation of the critical path constraint resulted in one failure for critical path constraint equal to 2 and no failures for the critical path constraint equal to 3.

As expected, for tests five thru eight the total number of suboptimal results decreased. This is due to the relaxed area constraint used in these four tests. Restricting the critical path constraint to 0 cuts per path resulted in four out of the six benchmarks achieving non-optimal results. Further relaxing the critical path consraint allowed all benchmarks to produce an optimal partition.

Table 5. Compress Results

Test #	Area Const.	C	Exit Condition	Run Time	Cut Size
1	1715344	0	Optimal	0.8	11
2	1715344	1	Optimal	0.9	11
3	1715344	2	Optimal	0.9	11
4	1715344	3	Optimal	0.8	11
5	1797027	0	No-Imp.	0.7	13
6	1797027	1	Optimal	0.7	13
7	1797027	2	Optimal	0.7	13
8	1797027	3	Optimal	0.7	13

Table 6. Find Results

Test #	Area Const.	C	Exit Condition	Run Time	Cut Size
1	4125646	0	No-Imp.	0.9	60
2	4125646	1	No-Imp.	0.9	60
3	4125646	2	No-Imp.	0.9	60
4	4125646	3	Optimal	0.8	60
5	4322106	0	Infeas.	2.4	83
6	4322106	1	Optimal	0.9	62
7	4322106	2	Optimal	1.7	58
8	4322106	3	Optimal	0.8	56

Table 7. Fifo Results

Test #	Area Const.	C	Exit Condition	Run Time	Cut Size
1	10829967	0	Infeas.	1.3	74
2	10829967	1	Optimal	1.3	137
3	10829967	2	Optimal	1.0	136
4	10829967	3	Optimal	1.7	106
5	11345680	0	Infeas.	1.4	74
6	11345680	1	Optimal	5.2	105
7	11345680	2	Optimal	2.3	89
8	11345680	3	Optimal	1.4	74

Table 9 shows that our method may also be a viable alternative to heuristic partitioners such as the FM method. Table 10 illustrates the impact of timing constraints on the cutset size. Columns 3-6 give the increase in cutset size for the four critical path constraint tests over a non-timing constrained problem. The benchmark is first tested with the tight area constraint and then the relaxed area constraint. As can be seen from this table, considering timing caused from a 48.9% increase to a 57.1% decrease and on average a 0.2% decrease in cutset size for the benchmarks tested.

Table 8. Viper Results

Test #	Area Const.	C	Exit Condition	Run Time	Cut Size
1	13372778	0	Infeas.	4.6	189
2	13372778	1	Infeas.	3.5	205
3	13372778	2	Optimal	3.0	187
4	13372778	3	Optimal	3.4	174
5	14009577	0	Infeas.	5.0	183
6	14009577	1	Optimal	2.7	185
7	14009577	2	Optimal	6.7	171
8	14009577	3	Optimal	4.1	195

Proceedings of the 1997 International Conference on Computer Design (ICCD '97)
0-8186-8206-X/97 $10.00 © 1997 IEEE

Table 9. Optimizing Cutset Only

Bench Mark	Run Time	Cut Size	Max Cut	FM Cut Size
TLC	0.4	11	3	12
TLC	0.5	12	1	12
decompress	0.5	14	1	33
decompress	0.8	6	0	21
compress	0.7	13	1	
compress	0.8	14	2	12
find	0.8	55	4	49
find	0.8	54	4	39
fifo	0.9	92	3	120
fifo	1.0	91	3	73
viper	2.1	142	4	154
viper	2.3	166	9	139

Table 10. Impact on Cutset

Bench Mark	Increase in Cutset for Critical Path Constraint = . . .			
	0	1	2	3
TLC	0	-8.3%	8.3%	-8.3%
TLC	0	0	8.3%	0
decompress	-28.5%	-57.1%	-57.1%	-57.1%
decompress	0	0	0	0
compress	-15.4%	-15.4%	-15.4%	-15.4%
compress	N/A	7.1%	7.1%	7.1%
find	N/A	N/A	N/A	9.1%
find	N/A	14.8%	7.4%	3.7%
fifo	N/A	48.9%	47.8%	15.2%
fifo	N/A	15.4%	-2.2%	-18.7%
viper	N/A	N/A	31.7%	22.5%
viper	N/A	17.4%	3.0%	17.5%

5 Concluding Remarks

In this paper we have presented a methodology that can be used for effective partitioning of circuits by taking multiple constraints into account. In general, partitioning with multiple constraints is solved by lumping cost parameters such as area, timing, power, and more into one multi-variable function. This has a tendency of not producing designs that can meet the required constraints. We have presented test results for a variety of large real circuits when taking area and timing costs into consideration. In general we have observed that our methods are fairly compute intensive and partitioning at gate level networks is not a preferred recommendation. However, partitioning using our techniques at RT level of design may be very effective as the size of a circuit's netlist is fairly small.

Our on going work includes addressing the problem of *k-way* partitioning, hierarchical partitioning (when multiple constraints like area, pin, cost, timing are very important for designing VLSI Systems), and exploring methods for guiding NLP solver to obtain better constraint satisfying local minimas, perhaps close to global minimas.

References

[1] T. Bui. Improving the performance of the kernighan-lin and simulated annealing graph bisection algorithms. In *Proceedings of the 1989 IEEE Design Automation Conference*, pages 775–778, 1989.

[2] S. Dutt. New faster kernighan-lin-type graph-partitioning algorithms. In *Digest of Technical Papers - ACM/IEEE International Conference on Computer-Aided Design ICCAD-93*, pages 370–377, November 1993.

[3] S. Dutt and W. Deng. A probability-based approach to VLSI circuit partitioning. In *Proceedings of the 1996 IEEE/ACM Design Automation Conference*, pages 100–105, June 1996.

[4] S. Dutt and W. Deng. VLSI circuit partitioning by cluster-removal using iterative improvement techniques. In *Digest of Technical Papers - ACM/IEEE International Conference on Computer-Aided Design ICCAD-96*, November 1996.

[5] C. Fiduccia and R. Mattheyses. A linear time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181, 1982.

[6] L. Hagen and A. Kahng. A new approach to effective circuit clustering. In *Digest of Technical Papers - IEEE International Conference on Computer-Aided Design ICCAD-92*, 1992 November.

[7] T. C. Hu and K. Moerder. *Multiterminal Flows in a Hypergraph*. IEEE Press, 1985.

[8] F. M. Johannes. Partitioning of VLSI circuits and systems. In *Proceedings of the 33st ACM/IEEE Design Automation Conference*, pages 83–87, 1996.

[9] B. Kernighan and S. Lin. An efficient heuristic procedure to partition graphs. *Bell System Technical Journal*, 49(2), February 1970.

[10] S. Kundu. An incremental algorithm for identification of longest(shortest) paths. *INTEGRATION, the VLSI Journal*, 17:25–31, 1994.

[11] L. Liu, M. Kuo, S. Huang, and C. Cheng. A gradient method on the initial partition of fiduccia-mattheyses algorithm. In *IEEE/ACM International Conference on Computer Aided Design*, pages 229–234, 1995.

[12] M. Minoux. *Mathematical Programming Theory and Applications*. Wiley-Interscience, 1986.

[13] J. More' and S. Wright. *Optimization Software Guide*. SIAM, 1993.

[14] B. Riess, K. Doll, and F. Johannes. Partitioning very large circuits using analytical placement techniques. In *Proceedings of the 31st ACM/IEEE Design Automation Conference*, pages 646–651, 1994.

[15] R.Raghavan and C. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, pages 365–374, 1987.

[16] P. Sawkar and D. Thomas. Multi-way partitioning for minimum delay for look-up table based fpga's. In *Proceedings 32th ACM/IEEE Design Automation Conference*, June 1995.

[17] M. Shih, E. Kuh, and R. Tsay. Integer programming techniques for multiway system partitioning under timing and capacity constraints. In *Proceeedings of the 1993 European Design Automation Conference*, pages 294–298, 1993.

0-8186-8206-X/97 $10.00 © 1997 IEEE