Partitioning of Functional Models of Synchronous Digital Systems

Rajesh Gupta and Giovanni De Micheli
Center for Integrated Systems
Stanford University
Stanford, CA 94305

Abstract

We present a partitioning technique of functional models that is used in conjunction with high-level synthesis of digital synchronous circuits. The partitioning goal is to synthesize multi-chip systems from one behavioral description, that satisfy both chip area constraints and an overall latency timing constraint. There are three major advantages of using partitioning techniques at the functional abstraction level. First, scheduling techniques can be applied concurrently to partitioning. Therefore, partitioning under timing constraints, and in particular under latency constraints, can be performed. Second, the functional model captures large hardware systems with fewer objects (than at the logic netlist abstraction level), making the partitioning algorithm more efficient. Third, hardware sharing trade-offs can be considered. In this paper, hardware partitioning is formulated as a hypergraph partitioning problem. Algorithms for hardware partitioning are presented and experimental results are reported.

1 Introduction

We present a technique aiming at multiple-chip synthesis from a a single high-level model in a Hardware Description Language (HDL). The partitioning of hardware functions in a chip set is crucial in achieving an efficient implementation. While hardware partitioning is dictated by the chip area limitations, it affects the performance of the overall system. The purpose of this research is to investigate computer-aided partitioning techniques that allow efficient implementation of hardware in multiple chips.

Unlike previous approaches [1] [2], we present a partitioning technique performed at the functional abstraction level, where the digital hardware being designed is represented by a sequencing abstraction model capturing the operations to be performed and their dependencies. Such a functional model is a common abstraction in high-level synthesis. Most high-level synthesis system use a variation of this model as an intermediate form [3] [4], because it can be obtained by compiling a hardware description in a HDL and it forms a convenient data-structure for synthesis algorithms.

Our partitioning approach is motivated by the following reasons. We assume that the hardware being designed is synthesized from a high-level model in a HDL under a maximum timing constraint on the overall hardware latency. By using high-level synthesis techniques, the designer may try first to find a design configuration (i.e. binding and schedule) that satisfies the chip area and latency constraints. When such a structure cannot be found, then partitioning is used to overcome the area limitations while meeting the timing requirements. It is important to note that partitioning may introduce timing penalties, due to the inter-chip communication delays. For this reason, the designer will choose a design configuration that satisfies the latency constraint as a starting point for partitioning. Thus the search for a binding (that defines the hardware sharing) is done prior to partitioning, and it benefits the partitioning method in providing a starting point with an estimated area smaller than an unbound configuration.

This approach is important for hardware prototyping using programmable gate arrays, that have a limited capacity in terms of gate count. By using the same functional model in a HDL, both a multi-chip prototype and the final implementation can be synthesized automatically. Bounds on the latency of the prototype are important to insure that accurate performance measures can be derived from it.

A major advantage of applying partitioning techniques at the functional abstraction level is that scheduling techniques can be applied concurrently to partitioning. Therefore the overall latency of a partitioned structure can be readily evaluated, including the inter-chip communication delays. In this way, area-performance trade-offs can be exploited. Secondly, the functional model allows us to capture large hardware systems with fewer objects than at the logic netlist abstraction level. As a result the partitioning algorithms are more efficient for large scale designs. The major disadvantage is the possible inaccuracy of the area and delay models, that are estimated directly from the functional models and that do not include interconnect delays.

High-level partitioning techniques were pioneered by Dirkes and Thomas [5]. They considered a multistage clustering algorithm, that perfected the clustering techniques presented in [6]. The algorithm operates on the Value Trace [3] functional model, that is similar to our model. However, Dirkes' approach was not formulated as a constrained optimization problem and scheduling and latency computation was done separately in a later step [3]. Camposano and Brayton [7] studied high-level partitioning techniques by means of clustering, with the goal of improving the efficiency of logic synthesis. The major difference of our method over previous ones is that we bound the latency of the partitioned implementation.

We present in this paper partitioning techniques based on the Kernighan-Lin [9] and the Simulated Annealing [8] algorithms applied to sequencing hardware models described by hypergraphs. The algorithm has been implemented successfully in program *Vulcan*, that can be used in conjunction with programs *Hercules* and *Hebe* [10] to transform a hardware model in the *HardwareC* language into the logic netlists of the different partion blocks. We present then an example and some experimental results.

2 Problem modeling

We model hardware behavior as a hypergraph [11] having: i) *vertices* representing operations; ii) *edges*: ordered vertex pairs, representing dependencies; iii) *hyperedges*: vertex subsets, representing the sharing of hardware among operations. The edges are directed and do not form cycles. Vertices can model ***Boolean expression blocks*** that represent maximal sets of combinational logic equations, whose dependencies do not cross register boundaries. For these vertices, the *area-cost* is proportional to the corresponding number of literals. The *delay-cost* is a positive integer, corresponding to the (rounded-up) ratio of the maximum propagation delay in the block to the cycle-time. We assume the cycle-time to be a constant defined as part of the design specifications.

The overall area-cost of a hypergraph is the sum of the area-costs of its vertices, where all but one vertex incident to any hyperedge are discounted. The timing-cost of a hypergraph, also called *latency*, is the length of its longest weighted directed path, or *critical path*, where the weights are the delay-costs of the vertices.

The hardware model can be extended to include hierarchy, where vertices can be other hypergraphs (representing, for example, functional units that are characterized in terms or area and delay-costs in a bottom-up fashion), or represent the "body" of conditional and iterative constructs [10]. Timing constraints can be also considered as a part of the hardware specifications and their satisfaction be required by the partitioned implementation. In this paper, for the sake of conciseness, we consider only one maximum timing constraint $\overline{\lambda}$ on the overall latency. Therefore, we consider hardware specifications that exclude operations with data-dependent delays.

We consider partitioning under the following assumptions:

- Each block of the partition has an upper bound $\overline{A}$ on the area-cost.
- Each block of the partition has an upper bound $\overline{C}$ on the pinout-cost. The pinout-cost is the number of I/O pins excluding power/ground.
- The overall latency λ has an upper bound $\overline{\lambda}$.
- Synchronous single clock hardware implementation.
- Synchronous inter-block communication. An integer delay-cost is associated to each inter-block data transfer. Without loss of generality, we assume it to be one clock cycle.
- Shared hardware resources cannot be split among blocks of the partition.

We denote the (hierarchical) hypergraph by $\mathcal{H}$ and we state the general partitioning problem as follows:

Problem 1: *Partition a hypergraph $\mathcal{H}$ into a minimal number n of hypergraphs $\mathcal{H}_i$, $i = 1, 2, ..., n$ such that the area-cost of each block $A_i \leq \overline{A}$, the pinout-cost of each block $C_i \leq \overline{C}$ and the overall latency $\lambda \leq \overline{\lambda}$.*

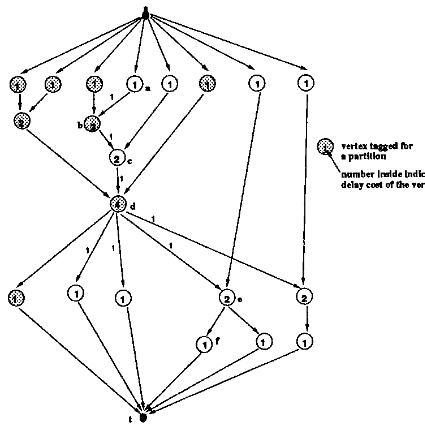
Generic network flow algorithms for hypergraph partitioning were investigated by Lawler [12]. Since in our problem we want to obtain *constrained* partitions, we have focussed on algorithms that support constraints on the sizes of the resulting partitions. We use a heuristic approach, to cope with the computational complexity of the problem. We approach the multi-way partitioning problem by performing successive bi-partitions. Let C be the communication cost (i.e. the number of wires) between the two blocks of the partition. The bi-partitioning problem can be stated as follows:

Problem 2: *Partition a hypergraph $\mathcal{H}$ into two hypergraphs $\mathcal{H}_i$, $i = 1, 2$ such that the area-cost of each block $A_i \leq \overline{A}$, the pinout-cost of each block $C_i \leq \overline{C}$, the overall latency $\lambda \leq \overline{\lambda}$ and the cost function $f = \alpha C + \beta(\lambda - \overline{\lambda})$ is minimal.*

A solution to Problem 2 is also a solution to Problem 1 for $n = 2$. If no feasible solution to Problem 2 exists, then a solution to Problem 1 may be found by relaxing the upper bound inequality on the size of the second block. Then, partitioning is applied iteratively to the second block of the partition until the area capacity constraint is met. The problem can be stated as follows:

Problem 3: *Partition a hypergraph $\mathcal{H}$ into two hypergraphs $\mathcal{H}_i$, $i = 1, 2$ such that the area-cost $A_1 \leq \overline{A}$, the pinout-cost $C_1 \leq \overline{C}$, the overall latency $\lambda \leq \overline{\lambda}$ and the cost function $f = \alpha C + \beta(\lambda - \overline{\lambda}) + \gamma(\overline{A} - A_1)$ is minimal.*

The last term in the cost function is a heuristic to achieve full utilization of the first block. In the sequel we describe a set of algorithms to solve Problem 2 and 3. Due to the similarity of the two problems, we will address the solution to Problem 2 in detail, and we will leave as comments the modifications needed to solve Problem 3.



vertex tagged for a partition
number inside indicates delay cost of the vertex
a
b
c
d
e
f
t

Figure 1: *Tagging and Latency Computation of SIF Graphs*

3 Partitioning Algorithms

The partitioning algorithms are based on two iterative improvement schemes. The former uses the *simulated annealing* algorithm [8] while the latter is related to the *Kernighan-Lin* algorithm [9]. We refer the reader to [9] [13] and [9] for the details of the algorithms. Here we concentrate on the moves and on the computation of the cost function.

We describe first partitioning of hardware models with no shared resources (i.e. no hyperedges) and no hierarchy. Therefore the hardware model is a directed graph. A partition is described by flagging each vertex with a tag that can take either one of two values. An example is shown in Figure 1. Iterative improvement in both algorithms is achieved by two kinds of moves: single-vertex displacement or swap of two vertices. In the former case the tag of a vertex changes, denoting its displacement from one block to the other of the partition. In the latter one, two vertices with different tag swap their tags.

We call *communication cost* the size of the data transferred between two blocks and its associated control. The pin-out C_i for the i^{th} block is the sum of the cost of I/O ports and communication cost. The variation in communication cost is computed by following the procedure outlined in [9]. The variation in area can be computed by adding (subtracting) the vertex area-cost for the block to (from) which the vertex has been displaced. In the case of swaps, the variations of the cost function can be computed as a sequence of two displacements.

The latency computation is more complex. We define the *latency of a partitioned graph* as the length of a modified critical path that includes edge weights, being 1 the weight of the edges that join vertices in different blocks and 0 otherwise. As an illustration, consider the example in Figure 1. The latency of the partitioned graph is determined by the longest path (s, a, b, c, d, e, f, t) to be 17 cycles of which 4 cycles are due to data transfer in the partitioned structure. Unlike area and communication costs, it is not possible to compute and update the latency *incrementally* for a generic move without traversing the subgraph(s) induced by the vertices that are successors of the moved vertex (vertices).

In the simulated annealing approach, we transform Problem 2 into an unconstrained optimization problem with penalty functions [13]. The cost function is a linear temperature-varying combination of the communication cost C, excess latency $(\lambda - \overline{\lambda})$, excess area $(A_i - \overline{A}; i = 1, 2)$ and excess pin-out $(C_i - \overline{C}; i = 1, 2)$. For problem 3, the excess area and pin-out are computed only for block $i = 1$ and the term $(\overline{A} - A_1)$ is added to the linear combination. For each move the algorithm computes the variation of the cost function. The latency computation requires a partial rescheduling, whose computational complexity is linear in the number of edges, whereas area and communication cost updates can be done in constant time.

We have also developed a faster partitioning algorithm, based on the Kernighan-Lin method. The original algorithm [9] applies to unconstrained optimization problems with $\beta = 0$. To cope with the latency computation and with the constraints we have considered the following two heuristics.

In the first one, the cost function is a linear combination of the communication cost, excess area and excess pin-out (i.e. excluding latency). Moves that would violate area and pinout constraints are discarded. The variation of the cost function can be easily computed for each move in constant time and it is recorded for each configuration. The configuration leading to the largest decrease of the cost function is then selected, as in the case of the original Kernighan-Lin algorithm[9]. If the decrease is non-positive, the algorithm terminates. Otherwise, latency is evaluated for this configuration. If latency does not satisfy the given bound, the configuration is discarded, another configuration is selected (leading to the largest decrease of the cost function), and the process is iterated.

A second heuristic strategy consists in computing an *approximation* of the latency at each move and incorporating it in the cost function. Moves such that the latency approximation violates its bound are discarded, as well as those that violate area and pinout constraints. Latency is affected by vertex displacement/swaps on the critical path, that may change after each move. Latency is approximated by assuming that the critical path does not change for the set of moves considered in the inner loop of the Kernighan-Lin algorithm. The local variation of the latency estimate can be easily computed for individual moves by checking the tags of the predecessor and successor vertices of the vertex being moved. The critical path is updated only when an exchange of vertices takes place, i.e. at the exit from the inner loop of the Kernighan-Lin algorithm. Note that an exact latency computation would require the computation of the changes to *all* paths in the graph. The approximation provides a lower bound on actual latency because it cannot be inferior than the delay of *a* path in the graph. However, the actual latency may be larger than the bound because of a change of the critical path. For this reasons, feasibility of a configuration must be checked by computing the exact latency before exchanging groups of vertices.

Let us comment now on the case in which hardware resource are shared. We assume that appropriate serialization of the shared tasks is represented by the graph edges [10]. Therefore the latency computation can still be done by graph traversal. Since shared resources cannot be split across partitions, the moves are now limited to vertices that are not incident to hyperedges and/or only to hyperedges, i.e. vertex subsets. In other words, objects of the moves are hardware resources that are either dedicated or shared. The algorithms described before be still be used with minor modifications, related to the move selection and cost function evaluation. In particular, the evaluation of communication cost function gain presented in [9] is extended to include movement of a group of vertices represented by the corresponding hyperedges.

Let us now consider hierarchical hypergraphs. According to our functional model [10], model call, looping and conditional constructs are represented by *complex* vertices and by using hierarchy. These vertices are not partitioned if a solution is found in terms of partitioning the hypergraph at the root of the hierarchy where complex vertices are considered indivisible. Otherwise, at descending levels of hierarchy the partitioning algorithm identifies the complex vertex with the largest associated area-cost. This complex vertex is considered for further partitioning as follows. The complex vertex is duplicated, so that one instance belongs to each block of the partition. The hypergraph called by the complex vertex is then considered for partitioning. An initial partition, satisfying resource sharing constraints, is then constructed and iteratively improved. This process is applied top-down in the hierarchy until an optimal overall partition is found. It is important to note that this formulation of hierarchical hypergraph partitioning preserves the ability of incrementally updating the communication cost as originally presented for flat graphs in [9]. An example related to a model call is shown in Figure 2. The calling vertex, C, is duplicated into two calling vertices, $C1$ and $C2$ each of which calls the partitioned model. Note that dependency edges $(a,c),(c,b),(d,c),(c,e)$ are accordingly modified in order to preserve the original sequencing dependencies. The same technique can be applied to complex vertices corresponding to looping constructs (repeated model call) and to those representing conditional constructs (selective model call), with the only extension that more than one hypergraph may be called by one conditional complex vertex.

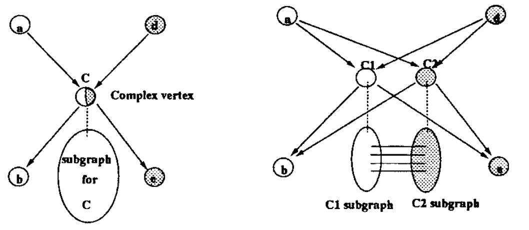


Figure 2: *Partitioning of Complex Condition-less Vertices*

Example	*Size*	λ	$\overline{\lambda}$	*Size1*	*Size2*	λ'	$\Delta Size$	*CPU*
FRISC	16018	24	27	6148	9990	26	+0.37%	778
6502	211094	34	38	97155	113939	38	+0.06%	1546
Elliptic	12542	20	21	6070	6532	21	+0.24%	11

Table 1: *Partitioning Results Under Latency Constraints*

4 Implementation and Results

The algorithms have been implemented in program *Vulcan*, that is written in the C programming language. The program reads a functional model generated by programs *Hercules* and *Hebe* from HardwareC description. *Vulcan* generates a partitioned model that can be transformed by *Hebe* into a synchronous logic circuit. In the present implementation, *Vulcan* solves Problem 2 and 3 with user supplied values for α β and γ. Multi-way partitions can be carried out under user control by solving Problem 3 repeatedly.

Program *Vulcan* was tested on a number of examples. Table 1 compares partitioning results. For all examples the values of parameters α and β were chosen to be 1. FRISC refers to a hardware description of a simple 16-bit microprocessor with 20 opcodes. 6502 refers to a commercial microcontroller design. All these designs where compiled from their respective *HardwareC* descriptions. The size and latency (λ) entries in columns 2 and 4 respectively refer to the original un-partitioned design. Column 4 refers to latency constraint,$\overline{\lambda}$ on the partitioned design. Size of the resulting partitions and latency (λ') of the partitioned model are reported in subsequent columns. These partitioning results are reported for Kernighan-Lin algorithm using complete cost function. $\Delta Size$ refers to area increase of the partitioned size. CPU run times are reported in seconds while running on DecStation 3100 with 16 MBytes of memory.

Table 2 shows the effect of resource sharing on hardware partitioning using an example of fifth order digital wave filter[14]. The original filter description was translated into *HardwareC* from an ISP model [15]. The filter description consists of 26 add operations and 8 multiply (by 2) operations. In addition, the design also contains 15 I/O operations. For the elliptic filter description used in Table 2 the multiply by a constant 2 operation in the filter description is replaced by combinational shift logic during behavioral synthesis phase by *Hercules*. As shown in the Table 2, the total size of the elliptic filter without any resource sharing is 6458. This size is estimated using the literal count of various blocks. Latency of the unpartitioned filter is 20 cycles. On partitioning the overall size increases to 6488 while latency increases to 17 cycles. Table 2 also compares running times and results for four different heuristics: simulated annealing (SA), Kernighan-Lin with cost function consisting of only area and pin-out costs (KL0), Kernighan-Lin with the complete cost function (KL1), and finally Kernighan-Lin where the latency is approximated as described in section 3 (KL2). For the case where only one adder is available to do all the add operations, the total area cost is 508 and latency of the unpartitioned graph is 30 cycles. Grouping together all 'add' vertices on a single hyperedges results in a partition in which all 'add' operations are performed in a single partition. Such a cut, however, increases the overall latency significantly. This demonstrates the effect of resource sharing on optimality of resulting partitions. Greater resource sharing leads to larger communication costs which increases the size of individual

Example	*Size*	λ	*Algo.*	*Size1*	*Size2*	λ'	*CPU*
Elliptic Filter	6458	16	SA	3244	3254	18	732
with no	6458	16	KL0	3001	3487	17	3
resource sharing	6458	16	KL1	3001	3487	17	331
	6458	16	KL2	3001	3487	17	10
Elliptic Filter	1222	17	SA	738	706	26	778
with four	1222	17	KL0	711	621	29	1
adders	1222	17	KL1	688	656	27	326
	1222	17	KL2	688	656	27	10
Elliptic Filter	746	20	SA	438	608	28	901
with two	746	20	KL0	453	423	31	2
adders	746	20	KL1	458	368	35	112
	746	20	KL2	403	453	31	2
Elliptic Filter	508	30	SA	368	340	35	805
with one	508	30	KL0	448	480	37	3
adder	508	30	KL1	343	263	35	38
	508	30	KL2	343	263	35	4

Table 2: *Effect of Resource Sharing on Partitioning*

$\overline{\lambda}$	$\overline{A}$	$\overline{C}$	*Size1*	*Size2*	λ'	*ΔSize*
21	none	none	6516	6506	21	+1.93%
22	none	none	6356	6346	22	+0.60%
23	none	none	5629	7073	22	+0.60%
22	7000	none	6356	6346	22	+0.60%
22	8000	none	6100	6562	22	+0.48%
22	9000	none	6070	6532	22	+0.24%
22	none	10	6100	6562	22	+0.48%
22	none	11	5363	7279	22	+0.39%
22	none	12	5825	6797	22	+0.31%

Table 3: *Elliptic Filter Partitioning Results Under Constraints*

partitions. For the elliptic filter we see that the effect of partitioning without any resource sharing is to increase area cost by about 0.5% and latency by 6%. However, the effect of partitioning the same filter with resource sharing increases the area cost by 9.0% and latency by 59% in case of four adders. Resource sharing with two adders increases area cost by 17.4% and latency by 55%. Finally, resource sharing with one adder increases area cost by 82% and latency by 23%.

Table 3 illustrates the effect of latency, area and pin-out constraints on partitioning results for the elliptic filter containing 26 add and 8 multiply resources. In contrast to the filter considered in Table 2, the multiply operations are now not restricted to be by 2 only. The multiply operations are instead modeled by calls to hardware blocks requiring two cycles per operation. Therefore, total size of the unpartitioned filter is 12542 considerably bigger than the filter description used in Table 2. The latency of the unpartitioned filter in this case is 20 cycles. In order to compare these results to results reported in [5], we have to impose the additional constraint of using 2 adders and 2 multipliers. In this case our algorithm yields a latency of 21 cycles, which is the same as in [5] when the I/O and inter-block communication delay is taken into account. No area comparison of the two approaches is meaningful due to different assumptions. It is observed that relaxing the latency limit from 21 to 22 reduced the impact on overall area from 1.93% to 0.60%. Relaxing the area limit from 7500 to 8000 reduced the increas on overall area from +0.60% to +0.48% for the same overall latency. Overall pin-out of the filter is 15. A pin-out constraint of 10 per block leads to a partitioned design which is 0.48% bigger. However, relaxing this constraint improves the overall size of the partitioned design as shown.

5 Summary

We have formulated the problem of partitioning hardware functional models under latency, area and pin-out constraints as a constrained hierarchical hypergraph partitioning problem. We have explored algorithms for generating partitions based on the simulated annealing and on the Kernighan-Lin algorithms. The latter algorithm is faster than the former and yields almost comparable results. It exploits two different heuristics to deal with constrained partitioning. The algorithms have been implemented in program *Vulcan* and tested successfully on benchmark examples. *Vulcan* can be used in conjunction with other high-level synthesis tools [10] to explore multi-chip implementations of a given functional model. Interesting trade-offs can be achieved by considering partitioning concurrently to resource sharing. The latter technique reduces area requirement at the cost of higher graph latency due to extra serialization introduced. Similarly partitioning reduces area requirement per partitioned block but adversely affects overall latency due to inter-block delays. Clearly, when design constraints can be satisfied by resource sharing alone, then partitioning is not required. However, partitioning techniques are required in the remaining cases, i.e. when chip area limitations can not be satisfied by resource sharing without violating latency constraints. Further research will be devoted to exploring further the relations between resource sharing and partitioning as well as perfecting the algorithms and considering more refined area/delay models.

6 Acknowledgments

David Ku implemented the programs *Hercules* and *Hebe* used for behavioral and structural synthesis respectively. This research was sponsored by NSF-ARPA, under grant No. MIP 8719546 and, by AT&T and DEC jointly with NSF, under a PYI Award program. We acknowledge also support from ARPA, under contract No. J-FBI-89-101.

References

[1] M. Beardslee, C. Kring, R. Murgai, H. Savoj, R. K. Brayton, A. R. Newton, "SLIP: A Software Environment for System Level Interactive Partitioning",*Proc. ICCAD'89* , Santa Clara, Nov 1989.

[2] W. E. Donath, "Logic Partitioning", in B. Preas, M. Lorenzetti (ed), *Physical Design Automation of VLSI Systems*, Chapter 3, Benjamin-Cummings Publishing Company, 1988.

[3] D. E. Thomas, E. D. Lagnese, R. A. Walker, J. A. Nestor, J. V. Rajan, R. L. Blackburn, "*The Systems Architect Workbench*", Kluwer Academic Press, 1989.

[4] M. C. McFarland, A. C. Parker, R. Camposano, "Tutorial on High-Level Synthesis", *Proc. 25th DAC*, 1988, pp. 330-336.

[5] E. Dirkes Lagnese, D. E. Thomas, "Architectural Partitioning for System Level Design", *Proc. 26th DAC*, pp. 62-67, June 1989.

[6] M. C. McFarland, S.J., "Computer-Aided Partitioning of Behavioral Hardware Descriptions", *Proc. 20th DAC*, pp. 472-478, 1983.

[7] R. Camposano, R. K. Brayton, "Partitioning Before Logic Synthesis", *Proc. 22nd DAC*, pp. 324-326, November 1987.

[8] S. Kirkpatrick, D. Gelatt and M. Vecchi, "Optimization by Simulated Annealing" *Science*, 220, N.4598, pp. 45-54, May 1983.

[9] B. W. Kernighan, S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *The Bell System Technical Journal*, 49(2) Feb 1970.

[10] G. De Micheli, D. Ku, F. Mailhot, T. Truong, "The Olympus Synthesis System", *IEEE Design and Test of Computers*, Oct 1990.

[11] C. Berge, "*Graphs and Hypergraphs*", North-Holland, 1973.

[12] E. L. Lawler, "Cutsets and Partitions of Hypergraphs", *Networks*, no. 3, pp. 275-285.

[13] P. J. M. van Laarhoven, E. H. L. Aarts, "Simulated Annealing: Theory and Applications", D. Reidel Publishing Company, 1987.

[14] P. DeWilde, E. Deprettere, R. Nouta, "Parallel and Pipelined VLSI Implementation of Signal Processing Algorithms", In S. Y. Kung, H. J. Whitehouse, and T. Kailath (ed), *VLSI and Modern Signal Processing*, pp. 257-264, Prentice-Hall, 1985.

[15] E. Dirkes Lagnese, *Private Communication.*