Efficient Synthesis of Networks On Chip *

Alessandro Pinto Luca P. Carloni Alberto L. Sangiovanni-Vincentelli
EECS Department, University of California at Berkeley, Berkeley, CA 94720-1772

Abstract

We propose an efficient heuristic for the constraint-driven communication synthesis (CDCS) of on-chip communication networks. The complexity of the synthesis problems comes from the number of constraints that have to be considered. In this paper we propose to cluster constraints to reduce the number that needs to be considered by the optimization algorithm. Then a quadratic programming approach is used to solve the communication synthesis problem with the clustered constraints. We provide an analytical model that justifies our choice of the clustering cost function and we discuss a set of experiments showing the effectiveness of the overall approach with respect to the exact algorithm.

1. Introduction

The continued growth of the number of processors and IP cores that are integrated on a single die [1] together with the shift from "computation-bound design" to "communication-bound design" [6] lead many researchers to advocate new design methodologies to organize systematically communication architectures for System-on-Chip (SOC). In [9], we proposed *Constraint-Driven Communication Synthesis (CDCS)* as a formal method for deriving automatically the implementation of a communication network of a system from a high-level specification: the resulting network is a composition of basic elements that are instances taken from a library of pre-defined Intellectual Property (IP) communication components, such as wires, repeaters, and switches. In CDCS, the essential communication requirements that govern all the point-to-point communications among the system modules are captured as a set of *arc constraints* in a graph called *communication constraint graph*. Similarly, the communication features offered by each of the components available in the IP communication library are captured as a set of *feature resources* together with their cost figures. Then, every communication architecture that can be built from the available components while satisfying all constraints is implicitly encoded as an *implementation graph* matching the constraint graph. Finally, the optimum network is found by solving a constrained optimization problem with an exact algorithm. Unfortunately, even though some theoretical results enable the reduction of the size of the search space [9], the computational complexity of the exact algorithm undermines its scalability, and, ultimately, the applicability of the approach to the synthesis of fairly large on-chip networks.

*This research was supported in part by the GSRC and the SRC.

In this paper we present two efficient heuristics for CDCS that are based on a decomposition of the optimization problem into two steps: (1) we use quadratic programming to compute quickly the cost of satisfying a set of arc constraints with a shared communication medium, and (2) we propose two clustering algorithms to single out sets of constraints that should be considered together. After providing some background material on CDCS, we discuss how the exact algorithm of [9] suffers from scalability issues and we summarize the main ideas of our heuristic approach (Section 2). In Section 3, we start from the assumption that the cost function of an arc implementation is a concave function and we show how this enables the efficient computation of the cost of "implementing" n arc constraints with a shared communication medium. Then, we present two distinct heuristics for clustering of constraints: divisive clustering and agglomerative clustering (Section 4). Finally, we report on a set of experiments showing the effectiveness of our approach (Section 5).

Related Work. In [2], Benini and De Micheli present network on chip (NOC) as a new paradigm for SOC design based on an approach similar to the micro-network stack model [10]. They discuss the design problems and possible solutions for each level of the stack from the application level to the physical level through the topology and protocol levels. The standard solution of the topology selection problem is the use of a single bus, but this may turn out quite inefficient from a power consumption viewpoint. Instead, [2] suggests to use packet-switching architectures. They focus on providing some examples of known topologies and do not discuss the problem of selecting an optimum one. A methodology centered on the simulation of traces is proposed in [7]: the resulting communication architecture is an interconnection of well-characterized communication structures similar to buses. Finally, in [4] the interconnection structure between computation blocks is fixed (a grid) and predictable. Information are routed in the communication network by means of dedicated switches. Constraint-driven communication synthesis (CDCS) [9] follows an approach that is inherently different from the previous ones because it aims to derive a communication architecture as the union of heterogeneous subnetworks that together satisfy the original communication constraints given by the designer.

2. A Heuristic Approach for CDCS

A *communication constraint graph* $CG(V,A,p,b)$ is a directed bipartite connected graph where each vertex $v \in V$

Proceedings of the 21st International Conference on Computer Design (ICCD'03)
1063-6404/03 $ 17.00 © 2003 IEEE

is associated to a port of a computational module of the system and each constraint arc $a \in A$ represents a point-to-point communication channel between two modules. The set of vertices V is partitioned in a set of source ports V_s and a set of target ports V_t, each vertex $v \in V$ has unit degree, a pair $p(v) = (p_x(v), p_y(v))$ is associated to each vertex $v \in V$ to denote its position on the plane, and a weight $b(a)$ is associated to each arc $a \in A$ to denote the required channel bandwidth. Given an arc $a = (u,v)$, the distance between its ports is denoted as $d(a) = ||p(u) - p(v)||$.

A *communication library* $\mathcal{L} = L \cup N$ is a collection of communication links and communication nodes. Each node $n \in N$ has a cost $c(n)$. Each link $l \in L$ has a set of *link properties*: (1) the link *length* (or, *distance*) $d(l)$ corresponds to the length of the longest communication channel that can be realized by this link, (2) the link *bandwidth* $b(l)$ corresponds to the bandwidth of the fastest communication channel that can be realized by this link, and (3) the link *cost* $c(l)$ is defined with respect to the other library links based on an optimality criterion that varies with the type of application.

Given a constraint graph $C\mathcal{G}(V,A,p,b)$ and a communication library $\mathcal{L} = L \cup N$, an *implementation graph* $I\mathcal{G}(C\mathcal{G},\mathcal{L}) = G(V' \cup N', A')$ is a directed graph where each vertex corresponds to either a vertex of V or a communication node instance from $\mathcal{L}$, each arc is associated to a link instance from $\mathcal{L}$, and for each arc of $C\mathcal{G}$ there is a corresponding path in $I\mathcal{G}$. The *cost* of an implementation graph $I\mathcal{G}$ is defined as: $C(I\mathcal{G}) = \sum_{n' \in N'} c(n') + \sum_{a' \in A'} c(a')$. Generally, for a given library there are many possible implementation graphs that satisfy the requirements expressed by the constraint graph while having different costs. In particular, one implementation graph, the *optimum point-to-point implementation graph*, is guaranteed to exist and it is derived by implementing a single arc constraint independently from all the others present in the constraint graph. On the other hand, by analyzing the definition of implementation graph it is clear that some of its arc implementations may share paths (i.e. links and/or communication vertices). Figure 1 illustrated the case where 3 arc constraints share a path. This structure is called a 3-way merging. In general, we may have a k-way merging, with $2 \leq k \leq |A|$. A k-way merging is characterized by the presence of two communication nodes: a node s at the beginning of the shared path and a node t ad the end. This structure models a shared communication medium such as a bus. Usually, the cost of an implementation graph is smaller than the sum of the costs of its point-to-point arc implementations. Hence, it is natural to define the following constrained optimization problem that can be seen as a special case of 0-1 integer linear programming (ILP).

Problem 2.1 *Given a constraint graph $C\mathcal{G}$ and a communication library $\mathcal{L} = L \cup N$, minimize the cost $C(I\mathcal{G})$ over all implementation graphs $I'(C\mathcal{G},\mathcal{L})$.*

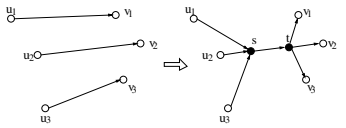


Figure 1. Example of 3-Way Merging.

The exact algorithm presented in [9] to find the solution of Problem 2.1 is divided in two steps: first a set of candidate k-way mergings are generated and added to the set of minimum-cost point-to-point implementations for all the given arc constraints, then an instance of the Unate Covering Problem (UCP) is solved. The algorithm is exact, but it is computationally expensive and it scales poorly with the cardinality of the set A of constraint arcs. The main contribution of this paper is the idea of solving Problem 2.1 with an heuristic approach that is centered around the efficient solution of a clustering problem [8]. In fact, we focus on finding an optimal partition of the set A of constraint arcs to minimize the overall cost of the implementation graph that is obtained by implementing separately each element of the partition with an optimum sub-network of communication library components. We propose two distinct algorithms for clustering of constraints: a divisive one and an agglomerative one. The former starts from a single cluster containing all constraints arcs and subsequently consider a series of smaller clusters that are derived by splitting the larger ones found at the previous step. Conversely, the latter starts from considering a set of singleton clusters, one for arc constraint, and subsequently attempts to merge smaller clusters to derive larger ones. In both cases, only a subset of all possible clustering configuration is considered, and the one with best implementation cost is chosen. Both algorithms need an estimation of the cluster cost which, generally, is represented by a function on a metric space (e.g., the maximum distance between members of the clusters). Instead, we use the actual implementation cost of the cluster. This would be either a point-to-point channel if the cluster contains only a single constraint or a k-way merging structure if it contains k arc constraints, with $k \geq 2$. For the latter case, this cost function may be quite complex. However, as discussed next, a reasonable assumption on the implementation cost of one constraint allows us to derive a closed-form expression for the cost of a k-way merging structure.

3. Finding the Structure of an K-Way Merging

In CDCS, the cost of an n-way merging is defined as the sum of the cost of all arcs plus the cost of all communication nodes in the implementation graph. This sum depends on the required bandwidths of all the constraint arcs to be merged and on the position of the corresponding source and destination ports. We make the following assumption on the cost of the implementation of a constraint arc: *Given an arc $a = (u,v)$ of a constraint graph $C\mathcal{G}$ with bandwidth $b(a) = b$ the cost of its point-to-point implemen-*

tation is $C(a) = f(b) \cdot ||p(u) - p(v)||_2^2$, where $f(b)$ is a concave function of b. It is reasonable to assume that the cost of a communication link depends on the distance to be covered, while the assumption of the concavity of function $f(b)$ is justified by the following consideration: the cost of covering a distance with a link supporting bandwidth $2 \cdot b$ should be at most twice the cost of covering the same distance with a link of bandwidth b. With the previous assumption we can show that the cost of a n-way merging can be computed analytically after deriving the detailed structure of an n-way merging through the solution of a quadratic programming problem. For each constraint arc a_i in the implementation graph $I\mathcal{G}$ we pick the link in the library with the smallest bandwidth greater then $b(a_i)$. Let c_i be the cost of this link. Also we pick the link in the library with the smallest bandwidth greater then $\sum_{i=1}^{n} b(a_i)$ and let c_0 be the cost of this link. Then, the minimization problem can be written as:

$$\min_{s,t} \sum_{i=1}^{n} c_i \cdot \left(||p(u) - p(s)||_2^2 + ||p(v) - p(t)||_2^2 \right) + c_0 \cdot ||p(s) - p(t)||_2^2$$

where s and t are the two communication nodes in th n-way merging. This problem can be decomposed into two independent sub-problems along the x, y coordinates of the plane. For the x coordinate, the minimization problem is:

$$\min_{s,t} \sum_{i=1}^{n} c_i \cdot \left((p_x(u) - p_x(s))^2 + (p_x(v) - p_x(t))^2 \right) + c_0 (p_x(s) - p_x(t))^2$$

This can be re-written as $\min_x x^T P x + x^T q + r$, where:

$$x = (p_x(u), p_x(v))^T, \qquad P = \begin{pmatrix} \sum_{i=0}^{n} c_i & -c_0 \\ -c_0 & \sum_{i=0}^{n} c_i \end{pmatrix},$$

$$q = -2 \begin{pmatrix} \sum_{i=1}^{n} c_i \cdot p_x(u) \\ \sum_{i=1}^{n} c_i \cdot p_x(v) \end{pmatrix}, \qquad r = \sum_{i=1}^{n} c_i \cdot (p_x(u)^2 + p_x(v)^2)$$

Note that P is responsible for all the quadratic terms while q for all the linear terms. The matrix P is positive semidefinite, the objective function is convex and the problem has only one solution [3]. The solution to the unconstrained minimization problem can be computed in closed-form by setting the gradient of the differentiable function equal to zero: $\nabla(x^T P x + x^T q + r) = 2Px + q = 0 \Rightarrow x^\star = -\frac{1}{2} P^{-1} q$. The inverse of the matrix always exists unless all costs are zeroes:

$$P^{-1} = \frac{1}{\left(\sum_{i=0}^{n} c_i\right)^2 - c_0^2} \begin{pmatrix} \sum_{i=0}^{n} c_i & c_0 \\ c_0 & \sum_{i=0}^{n} c_i \end{pmatrix}$$

The cost of the n-way merging is, then, $p^\star = -\frac{1}{2} q^T x^\star + r$. Note that P is a two-by-two matrix because the structure of an n-way merging has only 2 switches s, t. However, one could envision other structures with more than two communication nodes. While the sizes of P and q would change accordingly, the nature of the optimization problem would remain the same as long as the routing path of the data from the source to the target ports is decided a priori. If this is not the case, the optimization problem becomes an instance of geometric programming that can be still solved easily [3].

4. Hierarchical Clustering of Constraints

We use the cost of an n-way merging to estimate the cost of a cluster of constraints and we focus on clustering as the first step towards finding an optimal implementation graph. As clustering is known to be NP-complete, we propose two heuristic algorithms: *divisive clustering* and *agglomerative clustering*. Both algorithms explore efficiently a subset of all possible clustering by generating them using a hierarchical technique.

```
1: DivisiveClustering(CG)
2: SG ← deriveSimilarityGraph(CG)
3: SF ← computeSpanningForest(SG)
4: i ← 0
5: while ||SF|| > n do
6:    (e, sol[i], cost[i]) ← selectSplittingEdge(SF)
7:    remove e from SF
8:    i ← i + 1
9: end while
10: return sol[argmin_i cost[i]]
1: selectSplittingEdge(SF)
2: for all edges e ∈ SF do
3:    remove e from SF
4:    sol[e] ← Implement(SF)
5:    cost[e] ← Σ_{∀t∈SF} cost(t)/|t|
6:    add e to SF
7: end for
8: pick e that minimizes cost[e]
9: return (e, sol[e], cost[e])
```

Divisive Clustering. This algorithm is centered around the notion of similarity functions between pairs of constraints, which captures the advantage of implementing two constraints with a shared communication medium as opposed to realized them with a dedicated connection. The *similarity function* $\sigma(a_1, a_2)$ between two constraints a_1 and a_2 in $C\mathcal{G}$ is defined as $\sigma(a_1, a_2) = c(a_1, a_2) - c(a_1) - c(a_2)$, where $c(a_1, a_2)$ denotes the cost of implementing a_1, a_2 as a two-way merging, while $c(a_i)$ is the cost of implementing a_i as a point-to-point connection. From the constraint graph $C\mathcal{G}(V, A, p, b)$, we derive a directed complete similarity graph $S\mathcal{G}(W, E, \omega)$ as follows: (1) a vertex $w(a) \in W$ is associated to each constraint arc $a \in A$, (2) an edge $e = (w_1, w_2) \in E$ is drawn between any pair of vertices $w_1, w_2 \in W$, and (3) a weight $\omega(e)$ is attached to each edge $e = (w_1, w_2)$ such that $\omega(e) = \sigma(a_1, a_2)$, where $w_i = w(a_i)$. The divisive clustering algorithm is divided in three steps: First, the similarity graph $S\mathcal{G}$ is derived from the constraint graph $C\mathcal{G}$. This step requires $n(n-1)/2$ similarity function computations. Then, the minimum spanning forest $S\mathcal{F}$ is found for the similarity graph $S\mathcal{G}$. This step takes time $O(E \log V)$ and, since $S\mathcal{G}$ is a complete graph, returns a minimum spanning tree. This spanning tree captures the similarities among the vertices of $S\mathcal{G}$ (i.e. the constraint arcs of $C\mathcal{G}$), as "more similar" vertices end up being adjacent in the tree. The algorithm proceeds by removing one edge at the time from $S\mathcal{F}$, which becomes a spanning forest with two trees after the first removal, and, sees the "generation" of a new tree after each subsequent removal. The edge to be removed is found invoking the routine *selectSplittingEdge*, which returns also the cluster-

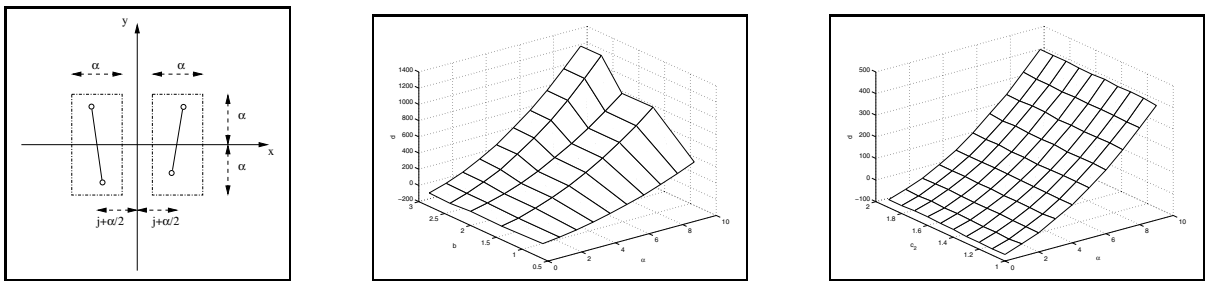


Figure 2. Experiment settings (l), σ vs. position and bandwidth (c), σ vs. position and cost (r)

ing encoded by the new forest together with its cost effectiveness (i.e. the cost of the corresponding implementation divided by the number of trees in the forest). After n invocations of routine *selectSplittingEdge*, $\mathcal{SF}$ becomes a forest without edges, while n distinct implementations have been considered and stored, together with their costs, in the array variables *sol* and *cost*. Finally, the clustering with minimum cost effectiveness is returned. Overall, assuming that the solution of the quadratic programming problem is obtained with a constant number of operations, the algorithm complexity can be derived as follows: the main while loop is executed n times and at the i-th iteration, the inner procedure has to remove/add $n-i-1$ edges. So the number of constant time operations is the sum of the first $n-1$ integers. The complexity is then $O(n^2)$.

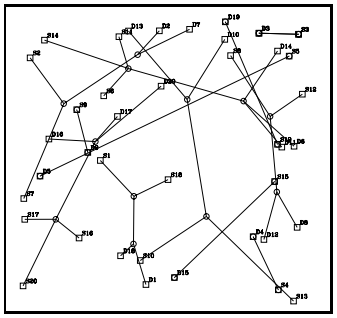
From its definition, similarity function σ depends on the constraints specification and on the communication library. However, by reporting on a couple of experiments, we show here that the dependency from the library is indeed quite weak. In Figure 2, two bounding boxes are depicted such that their position in the Euclidean plane depends on two parameters j and α. Let's assume that the bounding boxes define two distinct areas where each of two arc constraints may respectively reside and that the position of the source and the destination port of each constraint are chosen randomly and uniformly within each bounding box. For a given α, we sweep parameter j to increase the distance between the two constraints: for each position of the bounding boxes, we randomly generate 1000 pairs of constraints and compute the corresponding value of σ. Finally, we calculate the arithmetic mean. We repeated this procedure for two different experimental scenarios. First, we set the bandwidth of the two constraints to be the same (equal to b) and we repeat the above procedure for different values of b. In this experiment we consider a communication library characterized by the following pairs of bandwidth and cost per unit length: $\{(1,1),(2,1.8),(3,2.6),(4,3.4),(5,4.2),(6,5.0)\}$. It is easy to verify that *the library is concave*, meaning that the cost per unit length as a function of the bandwidth is a concave function. Figure 2 illustrates the dependency of the mean value of σ on both α and the required bandwidth b. In the second experiment, we keep the required bandwidth of the two constraints equal to 1 and we consider a library presenting two kind of links. The first link can support a bandwidth equal to 1 with a cost per unit length c_1. The second link can support a bandwidth equal to 2 with a cost per unit length $c_2 = c_1(1+\delta)$ where $0 \le \delta \le 1$. Notice that for each possible value of delta the library is still concave. Figure 2(right) shows that the similarity function doesn't depend sensibly on the parameter δ and, therefore, neither on the characteristics of the library. To understand this fact, let's reconsider the quadratic programming approach of Section 3. The cost of 2-way merging can be written as $c_x = x^T P x + x^T q + r = -\frac{1}{4} q^T P^{-1} q + r$, where r is independent from the cost c_2. Matrix P^{-1} is the only factor depending on c_2 and is written as follows:

$$P^{-1} = \frac{1}{c_1^2 + 2c_1c_2}\begin{pmatrix} c_2+c_1 & c_2 \\ c_2 & c_2+c_1 \end{pmatrix} = \frac{1}{c_1(3+2)\delta}\begin{pmatrix} 2+\delta & 1+\delta \\ 1+\delta & 2+\delta \end{pmatrix}$$

where the equality holds when $c_2 = c_1(1+\delta)$. The matrix eigenvalues are $\lambda_1 = 1/c_1$ and $\lambda_2 = -2/(c_1(3+2\delta))$. They determine the shape of the paraboloid over which the cost is computed. While λ_1 is independent from δ, λ_2 has a very smooth dependency. Hence, the value of the quadratic function computed at q is almost the same for all the values of δ (note: q and r don't depend on δ). In summary, the similarity function can be considered as *technology independent*, i.e. only dependent on the problem specification.

```
1: Agglomerative Clustering(CG)
2: K^0 ← ∅
3: for all constraints arcs a_i ∈ R do
4:     K^0 ← K^0 ∪ a_i
5: end for
6: l ← 0
7: while |K^l| > 1 do
8:     (i, j) ← argmin_{u,v∈[1,n−l]} σ(k_u, k_v)
9:     K^{l+1} ← K^l \ {k_i, k_j}
10:    K^{l+1} ← K^{l+1} ∪ {k_i, k_j}
11:    l ← l + 1
12: end while
13: return K^{opt} where opt = argmin_{t∈[0,n−1]} c(K^t)
```

Agglomerative Clustering. This is a greedy algorithm that, at each step, considers more complex clusters that are derived by composing the simpler clusters found at the previous step. Before, we defined the similarity function between two constraints. Now, we extend the concept to clus-



$\|A\|$	Exact Algorithm cost	Exact Algorithm time	Divisive cost	Divisive time	Agglomerative cost	Agglomerative time
15	4.74	210	15.99	160	5.77	230
20	6.82	1422	15.57	350	6.84	470
25	6,63	8882	18.34	410	8.18	961
30	–	t/o	20.38	911	7.12	1932

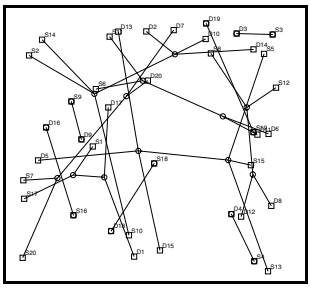


Figure 3. Exact algorithm (left), results table (center), agglomerative clustering (right)

ters of constraints. The intuition is the same: the similarity function measures the advantage of implementing two clusters with the same communication medium versus using two dedicated media: The *similarity function between two constraint clusters* k_i, k_j is $\sigma(k_i, k_j) = c(k_i \cup k_j) - c(k_i) - c(k_j)$. This function is used by the *agglomerative clustering* algorithm to derive an optimum clustering. The algorithm first builds a single cluster for each constraint in CG. The set of these clusters is denoted as $\mathbb{K}^0 = \{a_1, \ldots, a_n\}$. Then, at each step of the "while loop", the two *more similar* clusters are greedily selected and merged together. Thus, the algorithm considers n distinct clustering configurations before returning the optimal one from an implementation cost viewpoint. The complexity of this algorithm is $O(n^3)$.

5. Simulation and Result

We implemented the proposed clustering algorithms in a C++ package called SENC (Synthesis Engine for Networks-on-Chip). To derive the optimum n-way merging topology, SENC uses the NEWMAT matrix library [5]. The table of Figure 3 reports the experimental results obtained by running the two clustering algorithms and the exact optimization algorithm from [9]. We used a communication library with only three types of links having respectively bandwidth $100, 500, 1000$ and cost per unit length $2, 2.2, 2.4$. We randomly generated four constraints graphs with arc cardinality $|A|$ equal to $15, 20, 25, 30$. For each algorithm, we report the cost of the implementation graph and the *CPU time* in milliseconds (on a $750MHZ$, $256Mbyte$ Pentium III). The exponential nature of the exact algorithm is clear: for $|A| = 30$ it does not return the solution in the allotted time (10 minutes). As expected from the complexity analysis, divisive clustering is faster than agglomerative clustering. However, the latter returns better results that are quite close to the exact solution (when this is computed). Note that the cost of the implementation graph may be non-monotonic in the number of requirements because adding new constraints may lead to the generation of new mergings with constraints that previously were implemented as point-to-point dedicated links. The two diagrams of Figure 3 show the synthesis results for the exact algorithm and the agglomerative clustering one (case $|A| = 20$). For each constraint a_i, S_i is the source port and D_i the target port. Note the similarities between the results obtained with the two approaches. Also notice how *outliers* are singled out by the clustering algorithm (particularly constraints $3, 4, 9, 18$). Constraints $3, 4$, and 9 are so short and with low bandwidth that are better implemented as point-to-point channels. Arc constraint 18 is oriented orthogonally to all its neighbors, which are long arc constraints that get merged together.

Concluding Remarks. We divided the optimization problem of CDCS [9] into two steps: clustering of constraints and optimal synthesis of the clustered constraints. We cast the latter as a quadratic programming problem. For the former, we developed two heuristic algorithms, divisive clustering and agglomerative clustering, whose complexity is respectively $O(n^2)$ and $O(n^3)$, where n is the number of constraints. Experimental results show that the agglomerative algorithm is closer to the exact solutions than the divisive one, which, however, runs faster. Even though we focused on bus-based communication networks, the applicability of our approach is quite general because changing the cluster implementation does not change the nature of the optimization problem.

References

[1] A. Allan, D. Edenfeld, W. J. Jr., A.B.Kahng, M. Rodgers, and Y. Zorian. 2001 Technology Roadmap for Semiconductors. *IEEE Computer*, 35(1):42–53, Jan. 2002.

[2] L. Benini and G. De-Micheli. Networks on-chips: A new soc paradigm. *IEEE Computer*, January 2002.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. available at http://www.stanford.edu/ boyd/cvxbook.html, 2000.

[4] W. J. Dally and B. Towles. Route packets, not wires. In *Proc. of the Design Automation Conf.*, pages 684–689, 2001.

[5] R. Davies. *Newmat C++ Matrix Library*. available at http://www.robertnz.net/nm_intro.htm, 2000.

[6] R. Ho, K. Mai, and M. Horowitz. The Future of Wires. *Proc. of the IEEE*, 89(4):490–504, Apr. 2001.

[7] K. Lahiri, A. Raghunathan, and S. Dey. Efficient exploration of the soc communication architecture design space. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 424–430, 2000.

[8] B. Mirkin. *Mathematical Classification and Clustering*. Kluwer Academic Publishers, 1996.

[9] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. Constraint-Driven Communication Synthesis. In *Proc. of the Design Automation Conf.*, pages 783–788. IEEE, June 2002.

[10] J. Walrand and P. Varaija. *High Performance Communication Networks*. Morgan Kaufmann,San Francisco, 2000.