Generation of Interconnect Topologies for Communication Synthesis*

M. Gasteier† M. Münch‡ M. Glesner†

†Darmstadt University of Technology
Institute of Microelectronic Systems
Karlstraße 15
D-64283 Darmstadt, Germany

‡University of Kaiserslautern
Institute of Microelectronic Systems
Erwin-Schrödinger-Straße
D-67663 Kaiserslautern, Germany

Abstract

One of the key problems in hardware/software co-design is communication synthesis which determines the amount and type of interconnect between the hardware components of a digital system. To do so, communication synthesis derives a communication topology to determine which components are to be connected to a common communication channel in the final hardware implementation.

In this paper, we present a novel approach to cluster processes to share a communication channel. An iterative graph-based clustering algorithm is driven by a heterogeneous cost function which takes into account bit widths, the probability of access collisions on the channels, cost for arbitration logic as well as the availability of interface resources on the hardware components to trade-off cost against performance in a most optimum fashion. The key aspects of the approach are demonstrated on a small example.

1 Introduction

The rapidly growing computational power of microelectronic components like microcontrollers and processors together with increasing integration densities for ASICs enable hardware realization of highly computationally intensive applications as they occur for example in the area of real-time image-processing. However, as it turns out one of the main obstacles in obtaining full benefit of such high performance components is the problem of communication. Most of these computationally intensive applications have to deal with an immense amount of data. Not the processing component itself, but the quality of the interfaces and throughput of connections between the components is often the main bottleneck of the system [1, 2]. Designing appropriate communication connections becomes a design task of its own [3], where the designer has to analyze a wide variety of possible implementations ranging from dedicated point-to-point connections to a single global bus in order to select a solution most appropriate for the requirements of the system and additional constraints [4]. The final performance of the system can only be assessed by extensive simulation or, as for many mechatronic applications, after implementation of a prototype. Therefore, manual evaluation of all possible implementation alternatives is usually not practical. *Communication synthesis* tries to automatically determine cost efficient communication connections, where the notion of "cost" is usually expressed in terms of a weighted sum of the number of buses with their required bus width, costs for intermediate storage and area for peripheral devices like arbitration logic or interfaces.

The problems to be solved during communication synthesis can be grouped according to the level of abstraction they appear on: On *topological level* we cluster processes into sets such that data transfers executed between processes within the same set are suited to share a common communication channel. This requires a global view of the connection topology where all transfers executed within the system have to be considered. On this level we do abstract from most physical implementation details of the communication structure used to execute transfers between processes within the same cluster. We will therefore in the following use the term *logical channel* for such a connection. On *channel level* the implementation details have to be evaluated for each logical communication channel. This can be done without knowledge of the global topology, information about the transfers executed between the processes assigned to the current channel is sufficient. Tasks to be solved on this level include selection of a protocol to be used as well as assignment of priorities or, in case of buses without

*This work was supported by the DFG under grant Gl144/11-1 and Gl144/11-2

arbitration, scheduling of accesses [5, 6, 7].

An optimal solution to the problem of communication synthesis can only be found if the problems on both levels, global and local, are simultaneously considered since they depend on each other. However, in order to reduce the complexity of the problems to be dealt with, we decided to use a divide-et-impera approach which determines a global topology without considering local implementation details. We propose a graph-based approach for clustering of processes on topological level which relies on basic transfer information only, i.e. which processes communicate with each other and how much information is exchanged. An initial solution, where each transfer is assigned its own logical channel, is optimized iteratively by merging logical channels.

After the optimization process, the resulting logical channels can be converted into descriptions of physical buses by applying techniques as for example presented in [8].

There are only few publications related to communication synthesis on topological level. In [1], a greedy approach is used to map a multi-process description onto one or more buses. A new bus is successively added whenever the expected delay of an existing bus exceeds a given threshold. Thus, bus selection and assignment is executed locally while we propose a more global view to achieve buses with a more balanced load. In [9], four different implementation alternatives for communication based on global and local memory and buses are classified. Selection of the model to be used is up to the designer. The selected model is then implemented by an automatic refinement procedure.

2 Problem Description

We specify the system to be realized as a set of n communicating processes $p_1, p_2, \ldots, p_n$ [10]. Two processes p_i and p_j exchange data with a certain frequency which we characterize by a so-called *communication density* d_{ij}. The density d_{ij}, $0 \leq d_{ij} \leq 1$ is defined as the quotient of the number of clock cycles in which at least one transfer from process p_i to process p_j is performed and the number of clock cycles executed in total. A density d_{ij} of zero means that no data transfers occur from p_i to p_j, whereas a density of one denotes that transfers are executed in each clock cycle. Such values can for example be derived by co-simulation, as described in [11]. Please note that the density does not specify the temporal distribution of transfers but just the number of transfers executed over a certain time interval.

Each process p_i is executed on a hardware component which communicates via a port interface with other processes. If the process is implemented in software executed on a standard component, for example a microcontroller, the number and widths of ports available are fixed. While most microcontrollers have n ports of a single width m there are also some types which provide ports of different widths. The number and widths of logical channels this component is connected to must not exceed the available port resources, otherwise forcing merging of channels.

We assume the temporal distribution of transfers to be non-deterministic. This means that the delay in terms of clock cycles between two transfers cannot be predicted in advance. Consequently, we have to use buses with arbitration logic for implementation of a logical channel since we cannot exclude multiple accesses at the same time. Applications for which this assumption does not hold can also be realized using the approach presented here. However, more efficient results are usually achieved when applying techniques as presented in [7] where buses without arbitration implement the required connections. It is also possible to mix both approaches if the execution times of data transfers are predictable for only a subset of the processes. In this case the processes in the subset can be treated as a single process for which a bus without arbitration is used for internal communication. Communication with processes not in the subset is then handled by the approach presented in this paper.

The main problem for efficient clustering under the given assumptions is the heterogeneity of the target function. This function has to control the clustering process in a way that, first, all port violations are removed and afterwards a performance/area trade-off is performed. Reducing hardware costs is achieved by channel merging, the degree of which needs to be traded off against the probability of access conflicts on the channels which in turn reduce their communication throughput.

3 Graph-based Clustering

3.1 The Graph Model

In a first step we derive a directed, cyclic graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, w, p, b)$ which models the system to be realized. Each process is modeled by a vertex $v_i \in \mathcal{V}$. To each vertex v_i we assign a priority $p(v_i)$ which characterizes the number of port violations. Port violations occur if the number and/or width of logical channels assigned to a vertex exceed the port resources available on the corresponding hardware component. As long as $p(v_i) > 0$, logical channels connected to v_i have to be merged. For hardware components $p(v_i)$ can initially be set to zero. However, it is also possible to specify a port capability for hardware components in order to

optimize the interconnect cost of a component.

An edge $e_{ij} = (v_i, v_j) \in \mathcal{E}$ indicates that data is transferred from the process associated with vertex v_i to the process associated with vertex v_j, i.e. that $d_{ij} > 0$. To each edge e_{ij} we assign a weight $w(e_{ij})$ which denotes the degree of efficiency with which the transfers between v_i and v_j can share a common channel with transfers between other vertices. The density d_{ij} is a good indicator for the weight of an edge between two processes. Transfers which are executed frequently, i.e. which own a d_{ij} close to one would with a high probability block a bus for other transfers, while transfers with d_{ij} close to zero block a bus with a low probability only. We therefore set $w(e_{ij}) = d_{ij}$. In addition, each edge e_{ij} is attributed with the number $b(e_{ij})$ of bits required to execute the corresponding transfers. $b(e_{ij})$ can be derived from the maximum width of all data transferred from v_i to v_j.

Merging of transfers into logical channels is performed by assigning edges $e \in \mathcal{E}$ of graph $\mathcal{G}$ to clusters C_i. A cluster C_i is defined as a set of edges from $\mathcal{G}$, an associated weight $\omega(C_i)$ which describes the total communication density and a bit width $\beta(C_i)$ which describes the required bit width for the logical channel represented by this cluster. Since each edge $e \in \mathcal{E}$ is contained in exactly one cluster C_i, the set $\mathcal{C}$ of all clusters C_i describes a partitioning of $\mathcal{E}$:

$$\mathcal{C} = \{C_i \mid \bigcup_i C_i = \mathcal{E} \ \wedge \ \forall i, j : i \neq j : C_i \cap C_j = \emptyset\}.$$

Each cluster $C_i \in \mathcal{C}$ thus defines a logical channel through which all transfers of the associated edges will be executed and which will be realized later on by a separate bus.

3.2 The Optimization Strategy

The algorithm we have developed to solve the problem described above follows an iterative approach. The basic characteristics of this algorithm are:

- The number and bit widths of logical channels connected to a component will not exceed the component's port resources.
- The average probability of access conflicts for each logical channel is balanced and can be upper-bounded.
- Transfers with widely differing bit widths will not be assigned to the same channel in order to avoid blocking of wide channels by transfers requiring small bit widths only.
- The trade-off between performance and area can be controlled by a single parameter.

Before describing the complete algorithm, we will first present the basic concepts used. The initial solution we start with assigns each edge between a pair of processes (v_i, v_j) its own cluster or logical channel. The cluster weight is initialized with the edge weight. We then optimize this initial solution by iteratively merging clusters, thus reducing the number of logical channels. In order to achieve a feasible solution, we first resolve any port violations by merging clusters connected to processes which exhibit such violations. After resolution of all port violations, we continue to optimize the clustering by executing a performance/area trade-off as specified by a trade-off parameter in the cost function. The costs for a current clustering $\mathcal{C}$ are calculated as follows:

$$\begin{aligned} f_c(\mathcal{C}) = L \sum_{v_i \in \mathcal{V}} p(v_i) + \sum_{C_i \in \mathcal{C}} \beta(C_i) \\ + c_t \cdot \max_{\forall C_i \in \mathcal{C} : |C_i| > 1} (\omega(C_i)) + n_{arb} \cdot cost_{arb} \end{aligned} \tag{1}$$

At the beginning of the clustering process, resolution of port violations is forced by the first term. L is a weight factor which should be set to a very large value in order to introduce high costs in case of port violations. Thus actions which will reduce port conflicts are preferred as long as port conflicts occur. The second and third term are used for evaluation of area and performance. The function $\beta(C_i)$ returns the bit width required for the logical channel assigned to cluster C_i. Accumulating the bit widths for all clusters gives us an estimate of the required area. The total performance of communication is usually dominated by the slowest bus within the system. We therefore estimate the performance of a clustering by evaluation of the maximum cluster weight of all shared channels, i.e. clusters which contain more than one edge. n_{arb} is the number of arbitration units which have to be implemented when channels are shared. Each arbitration unit increases the costs by $cost_{arb}$.

The trade-off between area and performance can be controlled by the parameter c_t. Since the sum of all bit widths of the channels will usually be much larger than the maximum density on the channels, c_t should be set to a value larger than the expected accumulated bit width in order to assign performance a greater priority over area. Setting c_t to a lower value will generate smaller channels with a higher probability of access conflicts. Although the designer will often not be able to come up with an optimal setting for c_t the first time the algorithm is executed, iterative adaption of c_t allows adjusting the generated solution accordingly.

To improve the quality of the generated solution, the algorithm applies a *look-ahead* technique before merging two clusters in order to assess the potential for further optimization. Therefore in each iteration we not only evaluate the costs for all possible merges of two clusters, but also check if there exist other clusters whose associated vertices are all contained in the clusters proposed for merging. In this case, a merge with either of these clusters executed in the next iteration will with a high probability significantly reduce the costs, since additional arbitration logic will not be required for merging in these clusters. To prefer such merges over a merge which, from a local point of view, would lead to a slightly better improvement but without further optimization potential, we calculate the resulting costs under consideration of the average improvement achieved in two steps. An example which demonstrates the effectiveness of the applied look-ahead technique is presented in Section 4.

The algorithm will always find a solution provided that there are no transfers performed by a process which cannot be handled by the hardware component the process is assigned to due to port limitations.

3.3 The Algorithm

Algorithm 1 formally describes the clustering algorithm. As described in the preceding section, the algorithm starts with a solution of maximum cost which is iteratively optimized by merging clusters until no further reduction in cost can be obtained. An initial solution of maximum cost obviously requires a separate logical channel for each edge in $\mathcal{E}$; this solution is constructed in Line 1 of the algorithm.

To be able to reason about the set of hardware components linked to a logical channel, we define a mapping

$$\mathcal{P} : C \to \mathcal{V}$$
$$\mathcal{P} : C_i \mapsto \{v \mid v \perp C_i\},$$

which maps a cluster C_i onto the set of vertices v adjacent to any edge in C_i. The symbol $\perp$ denotes the adjacency relation. In the following, we will refer to this set of vertices $\mathcal{P}(C_i)$ as the "projection" of the cluster C_i.

The "severeness" of a port constraint violation of a vertex v is expressed by the function $p(v)$: the higher $p(v)$, the higher the difference of the number of logical channels to which v is connected and the number of ports on the associated node processor. A vertex v for which $p(v) = 0$ holds does not violate its port constraints. A more rigorous, mathematical definition of p would require taking into account various other parameters (such as the bit width of each port) and is therefore omitted in this paper for the sake of simplicity.

Algorithm 1 Clustering of transfers

```
1:  C = {{e_ij} | e_ij ∈ E};
2:  cost_curr ← ∞;
3:  repeat
4:    if ∃v_i ∈ V : p(v_i) > 0 then
5:      X ← {v_i ∈ V : p(v_i) max.};
6:    else
7:      X ← {};
8:    end if
9:    cost_impr ← ∞;
10:   for all {C_i, C_j} ∈ C^2 : X ⊆ P(C_i) ∩ P(C_j) do
11:     cost_merge ← f_c(C_[C_i,C_j]);
12:     cost_la ← min_{C_k ∈ C : P(C_k) ⊆ P(C_i) ∪ P(C_j)} {f_c(C_[[C_i,C_j],C_k]), ∞};
13:     cost_new ← min{(cost_merge + cost_la)/2, cost_merge};
14:     if cost_new < cost_impr then
15:       M_1 ← C_i;   M_2 ← C_j;
16:       cost_impr ← cost_new;
17:     end if
18:   end for
19:   Improved ← FALSE;
20:   if cost_impr < cost_curr then
21:     C ← C \ M_1;   C ← C \ M_2;
22:     C ← C ∪ merge(M_1, M_2);
23:     cost_curr ← f_c(C);
24:     Improved ← TRUE;
25:   end if
26: until Improved = FALSE;
```

The outer loop of the clustering algorithm (Line 3–26) checks whether there are still vertices whose port constraints are violated. If so, the vertex with the worst violation is selected in Line 5. If this vertex cannot uniquely be determined, we randomly select a vertex with maximum $p(v)$. The selected vertex is stored in the set $\mathcal{X}$; this set remains empty if the port constraints for all vertices are satisfied.

In the inner loop (Lines 10–18), we investigate each pair (C_i, C_j) of clusters and determine the cost of the new topology assuming a merge of these clusters. In doing so, we require $\mathcal{X}$ to be a subset of the intersection of the projections of clusters C_i and C_j (Line 10), thereby constraining potential merges to those which would resolve open port constraint violations. Since the empty set is a subset of any set by definition, the selection of cluster pairs is not affected if no more constraint violations are present.

Let $C_{(i,j)}$ denote the resulting cluster of merging

clusters C_i and C_j. Let $\mathcal{C}_{[C_i,C_j]}$ denote the set of clusters derived from $\mathcal{C}$ by merging clusters C_i and C_j. To compute the cost of the resulting overall configuration, the cost parameters of the merged cluster are derived from those of its parent clusters as follows:

$$\omega(C_{(i,j)}) = \omega(C_i) + \omega(C_j) \quad (2)$$

$$\beta(C_{(i,j)}) = \max\{\beta(C_i), \beta(C_j)\}. \quad (3)$$

Based on these, we now re-compute in Line 11 the cost of the clustering *after* a merge using the cost function described in Section 3.2 and assign this cost to $cost_{merge}$.

To alleviate the problem of getting caught in a local minimum, we then perform a one-step "look-ahead" as discussed in the preceding section. We consider all clusters C_k whose projections are subsets of the union of projections of the clusters currently being investigated. Such a cluster C_k has the property that the corresponding channel only accommodates transfers which could also be assigned to the channel of the merged cluster $C_{(i,j)}$. As described in the preceding section, merging C_k into the latter could therefore be beneficial for reducing the overall cost of the bus topology. The best cost achieved by a two-step merge is assigned to $cost_{la}$ in Line 12. In the following line, we then compute the "effective new cost" after merging the clusters currently under consideration. Note that a successive merge that seems promising is incorporated in the value $cost_{new}$. The purpose of the `if`-clause in Lines 14–17 is to store the best cluster pair encountered so far in $\mathcal{M}_1$ and $\mathcal{M}_2$.

After having investigated all potential clusters to be merged, the best clusters are selected and effectively merged in Lines 19–25. The algorithm then iterates until no further improvement can be obtained.

4 Example

In the following, we will walk through a small example to illustrate the main features of our algorithm. Fig. 1(a) shows the initial graph $\mathcal{G}$. Four processes modeled by vertices v_1 to v_4 exchange data. While three of them (v_1, v_2 and v_3) communicate with each other, the fourth process (v_4) is connected to a single process (v_3) only. The edge weights $w(e_{ij})$ are derived from the communication densities as indicated. We assume that a bit width of eight is used for all transfers to simplify the example. However, later on we will also explain how a different bit width affects the results we achieve. We also assume that v_3 is implemented on a microcontroller providing two eight bit ports, such that a port constraint violation occurs as long as more than two channels are connected to v_3.

To demonstrate the influence of the trade-off factor c_t on the cost function $f_c(\mathcal{C})$ we will calculate costs for $c_t = 10$, $c_t = 20$ and $c_t = 40$. The corresponding costs will be referred to as $f_{c,10}(\mathcal{C})$, $f_{c,20}(\mathcal{C})$ and $f_{c,40}(\mathcal{C})$, respectively.

The initial clustering derived from $\mathcal{G}$ is depicted in Fig. 1(b). The cluster weights $\omega(C_i)$ (denoted in italics) are set according to the edge weights $w(e_{ij})$. If we assume a port violation cost of 8000 ($p(v_i) = 8$, $L = 1000$), calculation of the initial costs using the cost function (Eqn. 1) returns a value of $f_c(\mathcal{C}) = 8040$ independent of c_t.

In the following we will describe the clustering process. The costs for a arbitration unit are assumed to be 10.

Iteration 1:

In the first iteration, the algorithm only compares merging of clusters which contain vertex v_3, since this vertex exhibits a port violation. Thus, the following candidates are evaluated: pair (C_3, C_4), pair (C_3, C_5) and pair (C_4, C_5). The resulting costs for a merge of the pair (C_3, C_4) are shown in column 1 of Table 1. The first column describes the costs after the first merge, as calculated in Line 11 of the algorithm.

The cost for $c_t = 40$ is, for example, computed as follows:

$$\begin{aligned} f_{c,40}(\mathcal{C}_{[C_3,C_4]}) &= 1000 \sum_{v_i \in \mathcal{V}} p(v_i) + \sum_{C_i \in \mathcal{C}_{[C_3,C_4]}} \beta(C_i) \\ &\quad + 40 \cdot \max_{\forall C_i \in \mathcal{C}_{[C_3,C_4]} : |C_i| > 1} (\omega(C_i)) \\ &\quad + n_{arb} \cdot cost_{arb} \\ &= 1000 \cdot 0 + 4 \cdot 8 + 40 \cdot 0.3 + 3 \cdot 10 \\ &= 74. \end{aligned}$$

If the merge is executed, two clusters exist whose vertices are a subset of the merged cluster $C_{(3,4)}$ and therefore examined during look-ahead: C_1 and C_2. C_2 will be selected for look-ahead because the lower cluster costs incur lower total costs. The look-ahead costs, as calculated in Line 12, are listed in the second column of Table 1. The average costs are shown in column three while column four contains the effective cost which will be assigned to $cost_{new}$.

Merging pair (C_3, C_5) results in the same costs $cost_{merge}$ as for merging pair (C_3, C_4) since the cluster weights $\omega(C_4)$ and $\omega(C_5)$ are identical. However, no look-ahead costs $cost_{la}$ are calculated since there is no other cluster C_i such that $\mathcal{P}(C_i) \subseteq \mathcal{P}(C_{(3,5)})$. Thus, $cost_{new}$ will be set to $cost_{merge}$.

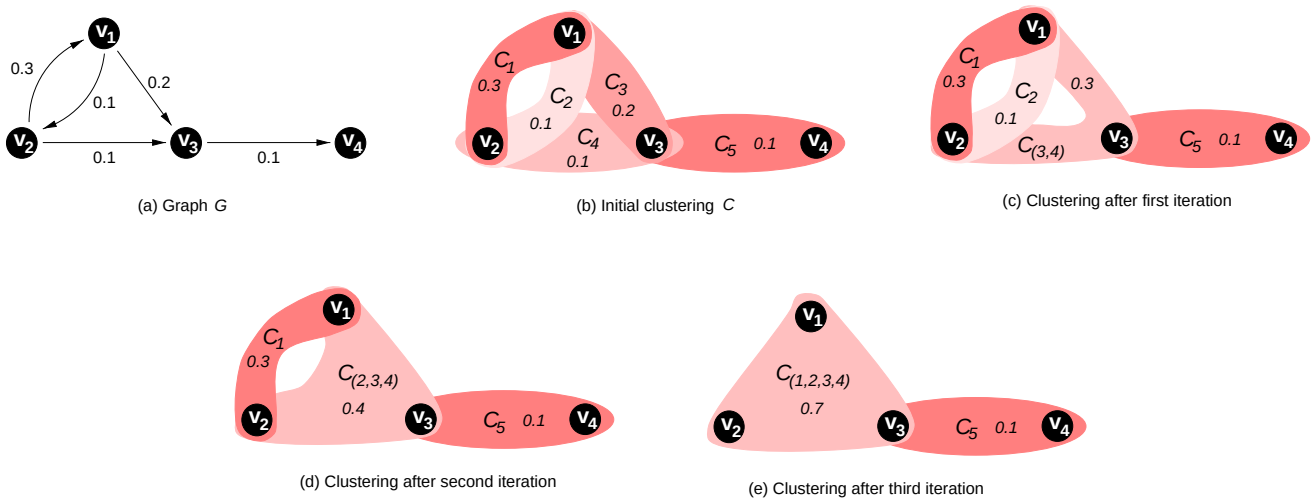

Figure 1: Clustering example

	$cost_{merge}$	$cost_{la}$	Average	$cost_{new}$
$f_{c,40}(\mathcal{C}_{[C_3,C_4]})$	74	70	72	72
$f_{c,20}(\mathcal{C}_{[C_3,C_4]})$	68	62	65	65
$f_{c,10}(\mathcal{C}_{[C_3,C_4]})$	65	58	61.5	61.5

Table 1: Costs after merging pair (C_3, C_4)

For a merge of pair (C_4, C_5) the following costs for $cost_{new}$ are achieved: $f_{c,40}(\mathcal{C}) = 70$, $f_{c,20}(\mathcal{C}) = 66$ and $f_{c,10}(\mathcal{C}) = 64$.

This step demonstrates the benefit gained from look-ahead. Although merging (C_4, C_5) leads to lower costs for a single merge, as indicated by the value of $cost_{merge}$, the minimum $cost_{new}$ are achieved by merging (C_3, C_4) for $c_t = 20$ and $c_t = 10$ because the future optimization potential is larger. The effect of c_t is also shown: for $c_t = 40$ (C_4, C_5) would be chosen. This would also be the final solution since no further improvement can be achieved. Furthermore, if the initial solution did not exhibit port conflicts, no merges would be performed since all merges exceed the initial cost of 40. The result of clustering assuming $c_t = 20$ or $c_t = 10$ is shown in Fig. 1(c).

Iteration 2:

In the second iteration, there are six candidates for merging since no more port violations have to be considered. When clustering any two of C_1, C_2 and $C_{(3,4)}$ $cost_{la}$ will be the same because the look-ahead extends the two selected clusters by the third one. Considering $cost_{merge}$, merging $(C_2, C_{(3,4)})$ will lead to the cheapest solution. Merging C_5 with any of the other three clusters is also too expensive since arbitration logic has to be inferred for all four processes in these cases. We therefore limit our analysis to the pair $(C_2, C_{(3,4)})$. The costs calculated for this merge are listed in Table 2.

	$cost_{merge}$	$cost_{la}$	Average	$cost_{new}$
$f_{c,20}(\mathcal{C}_{[C_2,C_{(3,4)}]})$	62	60	61	61
$f_{c,10}(\mathcal{C}_{[C_2,C_{(3,4)}]})$	58	53	55.5	55.5

Table 2: Costs after merging pair $(C_2, C_{(3,4)})$

For both trade-off factors, $c_t = 20$ and $c_t = 10$ the resulting costs are lower than the best cost achieved in the previous iteration. Therefore clusters C_2 and $C_{(3,4)}$ are merged, resulting in the clustering shown in Fig. 1(d).

Iteration 3:

In iteration three, the following merges are possible: $(C_1, C_{(2,3,4)})$, (C_1, C_5) and $(C_{(2,3,4)}, C_5)$. The first operation yields the lowest cost, as shown in Table 3. The resulting clustering is shown in Fig. 1(e).

	$(C_1, C_{(2,3,4)})$	(C_1, C_5)	$(C_{(2,3,4)}, C_5)$
$f_{c,20}(\mathcal{C})$	60	79	65
$f_{c,10}(\mathcal{C})$	53	73	58.5

Table 3: Costs after third iteration

If the maximum bit width of the transfers in C_1 was one, i.e. $\beta(C_1) = 1$, this merge would not be executed

for $c_t = 20$ and $c_t = 10$ in order to avoid blocking of the eight bit bus by single bit transfers. For $c_t < 4$, however, the merged solution yields lower costs.

Iteration 4:

In this iteration, no further improvements can be achieved. Merging the two remaining clusters results in $f_{c,20}(\mathcal{C}) = 64$ and $f_{c,10}(\mathcal{C}) = 56$ which in both cases exceeds the costs achieved in the previous iteration. The algorithm thus terminates, returning the clustering generated in iteration three and depicted in Fig. 1(e) as the final solution.

5 Conclusion and Future Research

In this paper we have presented an approach to derive communication topologies for complex systems from system-level communication characteristics. A topology suited for cost efficient implementation is constructed under consideration of port resources and channel access probabilities, thus allowing comfortable area/performance trade-offs. Due to the modest computational complexity the designer can iteratively adjust the topology according to her/his needs.

Future research will focus on improving the accuracy of the cost function. Its quality can be enhanced further by incorporating different classes of interface and arbitration logic depending on whether the process only writes to the channel, only reads from the channel or is doing both, reading and writing.

References

[1] T.-Y. Yen and W. Wolf. Communication Synthesis for Distributed Embedded Systems. In *Proc. of Intl. Conference on Computer-Aided Design 95*, pages 288 – 294, San Jose, CA, November 1995.

[2] M. P. Marks. Future Directions in Microprocessor Technology. *IEEE Journal of Solid-State Circuits*, 30(4):371 – 374, April 1995.

[3] S. Vercauteren, B. Lin, and H. De Man. Constructing Application-Specific Heterogenous Embedded Architectures from Custom HW/SW Applications. In *Proc. of the 33rd Design Automation Conference*, Las Vegas, NV, June 1996.

[4] W. Wolf. Hardware-Software Co-Design of Embedded Systems. *Proc. of the IEEE*, 82(7):967 – 989, July 1994.

[5] S. Narayan and D. Gajski. Synthesis of System Level Bus Interfaces. In *Proc. of the European Design & Test Conference 94*, pages 395 – 399, Paris, France, February 1994.

[6] P. Chou, R. Ortega, and G. Borriello. Synthesis of the Hardware/Software Interface in Microcontroller-Based System. In *Proc. of the ICCAD 92*, pages 488 – 495, 1992.

[7] M. Gasteier and M. Glesner. Bus-Based Communication Synthesis on System-Level. In *Proc. of the 9th International Symposium on System Synthesis*, pages 65 – 70, La Jolla, CA, November 1996.

[8] A.S. Wenban, J.W. O'Leary, and G.M. Brown. Codesign of Communication Protocols. *IEEE Computer*, pages 46 – 52, December 1993.

[9] J. Gong, D. Gajski, and S. Bakshi. Model Refinement for Hardware-Software Codesign. In *Proc. of the European Design & Test Conference 96*, pages 270 – 274, Paris, France, March 1996.

[10] M. Gasteier, T. Hollstein, M. Münch, and M. Glesner. An Interactive Approach to Hardware/Software Co-Design. In *IFIP Workshop on Logic and Architecture Synthesis 96*, pages 211 – 218, Grenoble, France, December 1996.

[11] M. Gasteier and M. Glesner. Co-simulation of Mixed HW/SW Systems (orig: Cosimulation gemischter HW/SW-Systeme). In M. Glesner, editor, *Hardwarebeschreibungssprachen und Modellierungsparadigmen: 2. GI/ITG/GME-Workshop*, pages 60 – 69, Darmstadt, Germany, February 1996.