Joint Communication Scheduling and Interconnect Synthesis for FPGA-based Many-Core Systems

Alessandro Cilardo, Edoardo Fusella, Luca Gallo, Antonino Mazzeo

Department of Electrical Engineering and Information Technologies, University of Naples Federico II
via Claudio 21, 80125 Napoli, Italy, Email: acilardo@unina.it

***Abstract*—This work proposes an automated methodology for optimizing FPGA-based many-core interconnect architectures. Based on the application communication requirements, the methodology concurrently defines the structure of the interconnect and the communication task scheduling, taking into account possible dependencies between tasks under given area constraints. The resulting architecture improves the level of communication parallelism that can be exploited while keeping area costs low. The paper thoroughly describes the proposed approach and discusses a few case-studies showing the impact of the proposed technique.**

I. Introduction and Related Work

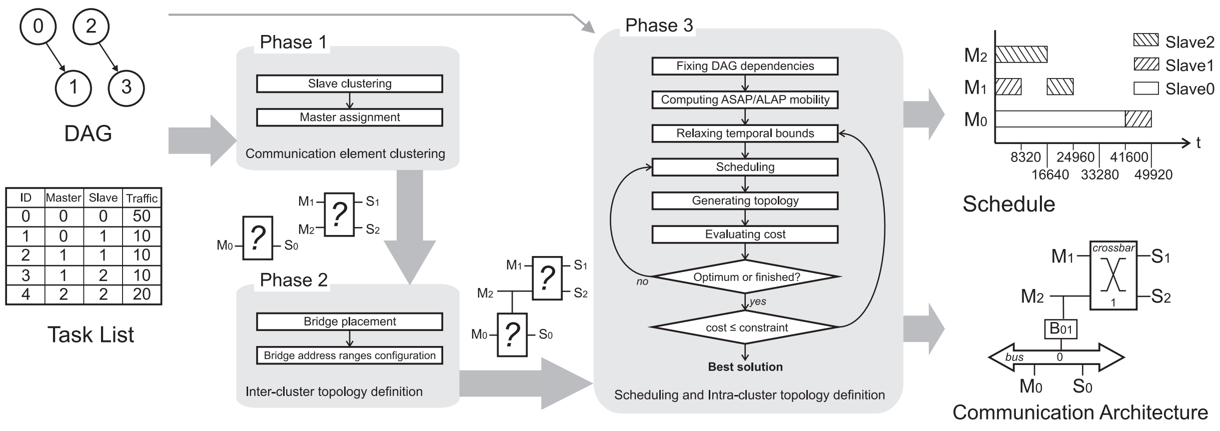
Multi-processor systems on chip (MPSoCs), particularly those based on reconfigurable hardware, can be designed to fit the requirements of a specific application [1], [2]. This possibility does not only involve the choice of the processing units, but also the definition of the interconnect. Due to the large spectrum of choices in terms of technologies, components, and topologies, determining the interconnect solutions that best suit given applications requirements is a non-trivial task [3]. Because of the popularity it has been gaining during the last years, crossbar-based interconnect design is addressed by several works. [4] and [5] focus on the synthesis of a single optimized crossbar. While this approach could be useful for small- or medium-sized designs, especially on ASIC technologies, a single crossbar interconnecting a large number of components on an FPGA may have a prohibitive cost and incur high latencies. In order to overcome the scalability issues of single-crossbar solutions, several approaches based on cascaded crossbars have been proposed [6], leading to topologies similar to those generated by our methodology, although they only target ASIC design flows. However, constraining the interconnect architecture to using only crossbars, as opposed to shared buses, results in higher area costs [7], particularly for FPGA-based implementations. [8] addresses this observation by supporting the generation of heterogeneous interconnects made of crossbars and shared buses. None of the previous works takes into account dependency constraints between communication tasks, which is essential to understanding the actual communication timing and properly dimension the interconnect avoiding underutilized interconnection resources, i.e. wasted area on the chip. [9] and [10] address the concurrent problem of scheduling and interconnection synthesis. They both express the cost of a point in the design space as the total number of links, which however may not correctly represent the hardware cost of a heterogeneous communication infrastructure, particularly for crossbars. Furthermore, they do not consider shared buses because they are only oriented to point-to-point links. While simplifying the problem, this assumption clearly lacks generality. Unlike most previous works on the automated synthesis of interconnect topologies, the proposed methodology, discussed in Section II, addresses the scheduling problem based on communication task dependencies, supporting joint communication scheduling/interconnect synthesis optimization. Targeted at FPGA technologies, the approach combines crossbars and shared buses, connected through bridges, yielding a scalable structure and enabling efficient communication patterns. In fact, the resulting architecture improves the level of communication parallelism that can be exploited, possibly allowing multiple paths to be used concurrently while keeping area costs low, as shown by a few case-study applications presented in Section III.

II. Proposed Methodology

We assume that the communication traffic is made up of *communication tasks* which are non-preemptive atomic entities with an arbitrary load (amount of transmitted bytes) transferred in a burst mode. Notice that two different tasks can have the same master/slave pair. This allows modeling any traffic pattern. The proposed methodology takes as input a communication Task List (TL) containing, for each task, the master and the slave involved and the amount of bytes to be transferred, along with a Directed Acyclic Graph (DAG) describing possible inter-task precedence relationships, i.e. dependency constraints. Under a given area constraint, the methodology finds a synthesizable topology specification based on a heterogeneous bus/crossbar architecture minimizing the target cost function, together with a compatible minimum-latency communication task schedule. Note that the problem of communication scheduling and synthesizable topology definition are deeply interrelated. Scheduling affects the final topology as it might require dedicated resources allowing certain tasks to be performed concurrently. Our methodology jointly considers the two aspects. The proposed topology synthesis flow, shown in Figure 1, consists of three phases, described in the following.

1) Communication element clustering: The first phase generates *local domains*, or clusters, i.e. subsets of nodes in the interconnect that can directly communicate with each other. Domains are defined so as to maximize the local, i.e. intra-cluster, traffic matching the traffic patterns induced by a given application. The phase is in turn structured in two steps: 1) hierarchical slave clustering, and 2) master assignment to

978-3-9815370-2-4/DATE14/©2014 EDAA



ID	Master	Slave	Traffic
0	0	0	50
1	0	1	10
2	1	1	10
3	1	2	10
4	2	2	20

Fig. 1. Proposed interconnect synthesis flow

the slave clusters. The first step performs an agglomerative hierarchical clustering in order to fix the number of local domains and determine the partitioning of the slave nodes. To cope with this problem, slaves are represented in a Euclidean n-space. For each slave h, we build an array containing N_m elements, where each element i represents the fraction of the total traffic from/to slave h involving master i and N_m is the total number of masters. Then, in order to decide which clusters should be combined, the Euclidean distance is used as a measure of dissimilarity between slaves. The clustering algorithm proceeds by merging clusters until it meets a stop condition depending on the worst-case area occupation (computed assuming all possible bridges between clusters and full matrices for intra-cluster communication). Notice that, with no area constraints, the clustering iterations would lead to a single crossbar. This is consistent with our goals because a single large crossbar guarantees the lowest possible communication overhead. With increasingly stringent area constraints, on the other hand, we will have a growing number of smaller clusters. The second step in the clustering phase assigns the masters to the clusters with which they exchange most data in order to keep as much communication as possible within single clusters (intra-cluster communication) and minimize the communication through bridges. Following this phase, masters and slaves are divided in local domains whose internal and external topologies are still to be defined.

2) Inter-cluster topology definition: The second phase determines the global inter-cluster architecture by connecting the clusters through bridges in order to make all inter-cluster communications feasible. By properly mapping the address spaces in each bridge, furthermore, multiple physical paths between different domains can be realized. Multiple paths introduce flexibility in the topology as they create a further opportunity for balancing the load across the interconnect. The phase is structured in two steps. First, we define the number and the position of the bridges by solving an optimum branching problem. We rely on a well-known technique, i.e. Edmonds' algorithm, setting the weights on the arcs to the inverse of the communication requirements between clusters, thereby prioritizing the arcs having higher requirements. Then, the bridge address ranges are defined by solving a path balancing problem [11], enabling an effective exploitation of the parallel communication paths available in the topology.

3) Scheduling and intra-cluster topology definition: The last phase defines the internal implementation of each cluster, along with the communication task scheduling. It is based on an iterative procedure, made of several steps, and outputs an optimal communication task schedule (in terms of latency) along with a global topology containing enough resources to perform the schedule found. As the first step, the DAG is transformed in order to take into account and preemptively solve any potential structural conflict due to slave and bridge accesses. The resulting DAG will comply with the following rule: *all tasks using the same slave or bridge must be connected by a single path.* This means that there will be a partial order relation for slave and bridge accesses. This is achieved by prioritizing tasks that, in an ASAP schedule, start first. Once the DAG has been transformed, the second step computes the ASAP and ALAP schedule and the mobility values. An iterative subphase takes then place, starting from the third step, which fixes a temporal bound, subsequently relaxed at each iteration of the outer loop. Since we are optimizing the global execution time, we start with the minimum temporal bound, i.e. the latency of the ASAP schedule. In fact, ASAP scheduling finds the minimum latency schedule in an unconstrained problem [12]. At each iteration of the inner loop, we calculate the best schedule, in terms of area, under that temporal bound (fourth, fifth, and sixth steps). This process relies on an algorithm to find the synthesizable architecture with the minimum degree of parallelism allowing a certain schedule to be run (fifth step). Then, the cost of the whole architecture is quantified by using a suitable area model. The last step checks if the area of the architecture found meets the given constraints. If this is not the case, we go back to the third step where the temporal bound is relaxed and the mobility values are recomputed. In the following, a few essential aspects of the above flow are described in more detail.

a) Relaxing the temporal bound: The granularity of the temporal bound, relaxed at each repetition of the third step of Phase 3, must guarantee a comprehensive exploration of the available design choices. Of course, relaxing the bound by a single clock cycle at each step would be infeasible. We chose to relax the bound by the minimum time allowing a task to move enough to eliminate an overlap in the schedule. In fact, less overlapping leads to less concurrency, and hence an architecture taking less area.

b) Scheduling algorithms: The scheduling algorithm determines the schedule corresponding to the synthesizable topology with the lowest area under a given temporal bound satisfying all the constraints expressed by the modified DAG (including slave and bridge conflicts). We evaluated six different scheduling algorithms that, due to the lack of space, cannot be described thoroughly here: random search, two different exhaustive search approaches, a genetic algorithm, a Priority-based List Scheduling, and a randomized Priority-based List Scheduling. The random search did not give satisfying results, while the two exhaustive approaches are too expensive and can be used only for small-sized problems. The priority-based list approach tries, at each step, to make the best move as possible within a list of permitted moves, i.e. those satisfying the dependency constraints. The list is ordered according to the area cost incurred by the topologies associated with each potential move. The procedure to derive a topology, given a schedule, is explained in Section II-4. The Priority-based List Scheduling approach is likely to yield the optimal solution only if we start from an initial schedule already reasonably close to the optimum; otherwise, it gets stuck at a local minimum (a point not having better neighboring solutions). The genetic optimization performs good on average, but cannot discover local minima with the same speed as the randomized Priority-based List Scheduling, which turned out to be the best choice. This algorithm is essentially an optimized version of the Priority-based List Scheduling approach. Unlike its basic version, it avoids getting stuck at a local minimum by recording the solution and generating another random starting point, possibly falling in a different region of attraction.

4) Schedule evaluation: All the used scheduling algorithms rely on a procedure to evaluate the task schedule. In that respect, we need to find the lowest cost architecture that exhibits enough parallelism to accommodate the identified scheduling. Concerning the derivation of valid intra-cluster topologies, we rely on compatibility graphs [12] to determine the number of master and slave ports needed by local interconnects.

Definition 1: Given a Communication Scheduling, two tasks t_1 and t_2 are *compatible* if and only if they do not overlap in time.

Definition 2: Two masters (slaves), including bridge master (slave) ports, are *compatible* if and only if all the tasks involving them are compatible and they belong to the same cluster.

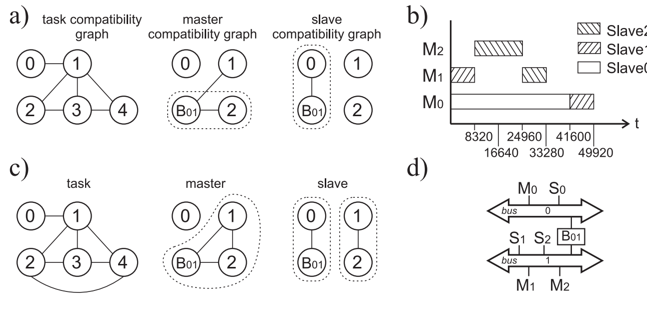


Fig. 2. Deriving a topology from given a schedule. (a) Compatibility graphs for the schedule in Figure 1. (b) An enhanced schedule, with less concurrency, for the same application. (c) Its compatibility graphs. (d) The derived topology.

From the above definitions, the construction of compatibility graphs for communication tasks and then for masters and slaves is straightforward. Starting from the compatibility graphs, for each connected component, we solve a clique-partitioning problem [12] to identify the minimum number of cliques. Then, we assign a master (slave) port to each clique in its cluster. If for a cluster a single clique is identified, both in the master and the slave compatibility graphs, then a single shared bus is enough because all tasks are compatible, hence sequential. Otherwise a crossbar is necessary. If there are no compatible masters (slaves) in a cluster, then a full crossbar will be used. Figure 2 shows the differences between the compatibility graphs for two different schedules. In Figure 2.a, masters M_1 and M_2 and slaves S_1 and S_2 are incompatible with each other. The derived topology, consisting of a crossbar for cluster C_1 and a shared bus for C_0, is shown on the right part of Figure 1. Notice that masters M_2 and B_{01} are compatible and, hence, they share the same crossbar port. Figure 2.b shows a different schedule that removes the above incompatibility (Figure 2.c). The solution of the clique-partitioning problem is highlighted by the dotted lines. This leads to a less expensive architecture, shown in Figure 2.d.

III. Experiments and Case Studies

We carried out our experiments on a ZedBoard FPGA board by Avnet Design Services, equipped with a device of the Xilinx Zynq™-7000 family, embedding reconfigurable hardware and a hard-core ARM processor. The custom interconnect and the communicating elements are mapped onto the reconfigurable fabric. In particular, salve nodes might be memory modules or any other memory-mapped hardware peripheral, possibly generated by means of high-level synthesis [13], [14], or manually optimized for application-specific purposes [15], [16]. The ARM processor, on the other hand, runs a software routine monitoring task execution and enforcing the previously determined schedule by controlling the communicating elements. The communication architecture synthesis flow uses the Xilinx AXI components compliant with the AMBA® AXI version 4 specification from ARM. We built accurate analytical models for the evaluation of the area cost and the latency, obtained by interpolation of extensive RTL synthesis results. We tested our method for five benchmark applications, whose DAGs were generated using the TGFF package [17], removing possible concurrent tasks with the same master. The characteristics of the benchmarks are summarized in Table I, where N_t, N_m, and N_s are the number of tasks, masters, and slaves, respectively. Table I also gives the number of clusters N_c found after the Communication Element Clustering phase. To appreciate the overall impact of our method, we compared

TABLE I. Characteristics of the Benchmarks

	N_t	N_m	N_s	N_c
APP II	13	5	5	2
APP III	25	7	8	3
APP IV	31	8	10	4
APP V	43	11	14	5
APP VI	62	12	16	6

it with a previous similar approach [8]. It is important to point out that [8] does not perform the scheduling step but only relies on aggregated parameters (i.e. the total amount of traffic exchanged between master/slave pairs) to automate the interconnection design. Figure 3 summarizes the results obtained by applying the two methodologies to all the test applications. We compared them also with a full crossbar implementation. Each point in the plot is derived by applying the methodology to a specific benchmark under a fixed area constraint ranging between 30% and 70% of the area of a full crossbar implementation. For each experiment, the points related to the crossbar implementation are the solution of the corresponding unconstrained problem, exhibiting the minimum possible latency at the price of a highly oversized area, while the other points are Pareto-optimal points. Nevertheless, our approach is still able to obtain a latency that is roughly the same as the full crossbar: on average the latency is only 13% larger compared to the latency obtained with a full crossbar implementation, while the area is 48% smaller. Using the approach in [8] we have a performance degradation instead: on average the latency is 40% larger than our approach, while the area is roughly the same. In fact, not considering dependency relationships, [8] may infer a crossbar even when it is not strictly necessary as well as buses in cases where some communication tasks can be parallelized. In other words, it may cause the serialization of tasks on the critical path, increasing the execution time. The points corresponding to full crossbars, as expected, are associated with the largest area.

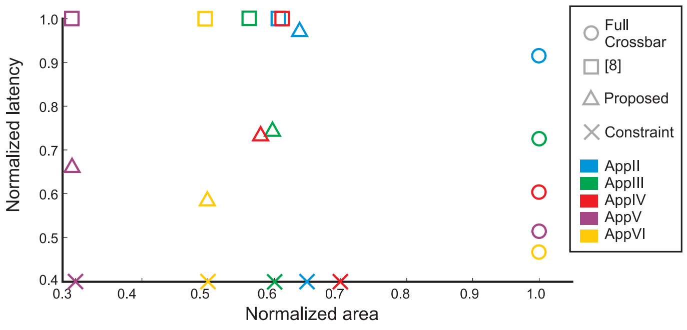


Fig. 3. A plot showing different solutions for various benchmarks and scheduling algorithms

To better explain these results, we provide Table II. Element (i, j) in the table represents the average fraction of time in which the communication links are used by application i using approach j. A low value means wasted area due to low channel usage. A value equal to 1 would mean a perfect interconnection for the application. Notice that the approach in [8] (as well as the use of a single crossbar architecture) may lead to an underutilized network because it does not consider dependency relationships, which of course are very likely to be found in any real application.

TABLE II. COMMUNICATION CHANNEL USAGE

	Full Crossbar	[8]	Our Approach
APP II	0.55	0.5771	0.7877
APP III	0.4383	0.4484	0.5546
APP IV	0.5053	0.4760	0.5496
APP V	0.3912	0.3214	0.4872
APP VI	0.4013	0.3075	0.4782

IV. Conclusion

We presented an automated methodology for the synthesis of complex heterogeneous on-chip interconnects made of crossbars, buses, and bridges. Our solution is based on a method for balancing the load across the bridges, as well as a novel approach to combined communication scheduling and interconnect generation. Experimental results show that our approach can synthesize designs made of dozens of IP cores with low communication overheads and area costs, exhibiting encouraging improvements over alternative proposals.

References

[1] A. Cilardo, L. Gallo, and N. Mazzocca, “Design space exploration for high-level synthesis of multi-threaded applications,” *Journal of Systems Architecture*, vol. 59, no. 10, pp. 1171–1183, 2013.

[2] A. Cilardo, D. Socci, and N. Mazzocca, “ASP-based optimized mapping in a Simulink-to-MPSoC design flow,” *Journal of Systems Architecture*, 2014.

[3] K. Lahiri, A. Raghunathan, and S. Dey, “Design space exploration for optimizing on-chip communication architectures,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 6, pp. 952–961, 2004.

[4] S. Pasricha, N. Dutt, and M. Ben-Romdhane, “Constraint-driven bus matrix synthesis for MPSoC,” in *Procs. of the 2006 Asia and South Pacific Design Automation Conference*. IEEE Press, 2006, pp. 30–35.

[5] S. Murali, L. Benini, and G. De Micheli, “An application-specific design methodology for on-chip crossbar generation,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 7, pp. 1283–1296, 2007.

[6] M. Jun, D. Woo, and E.-Y. Chung, “Partial connection-aware topology synthesis for on-chip cascaded crossbar network,” *IEEE Trans. on Computers*, vol. 61, no. 1, pp. 73–86, 2012.

[7] V. Lahtinen, E. Salminen, K. Kuusilinna, and T. Hamalainen, “Comparison of synthesized bus and crossbar interconnection architectures,” in *Procs. of the 2003 Int. Symposium on Circuits and Systems, 2003. ISCAS’03*, vol. 5. IEEE, 2003, pp. V–433.

[8] A. Cilardo, E. Fusella, L. Gallo, and A. Mazzeo, “Automated synthesis of FPGA-based heterogeneous interconnect topologies,” in *Procs. of the 23rd Int. Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2013, pp. 1–8.

[9] N. K. Bambha and S. S. Bhattacharyya, “Interconnect synthesis for systems on chip,” in *Procs. of the 4th IEEE Int. Workshop on System-on-Chip for Real-Time Applications*. IEEE, 2004, pp. 263–268.

[10] N. K. Bambha and S. S. Bhattacharyya, “Joint application mapping/interconnect synthesis techniques for embedded chip-scale multiprocessors,” *IEEE Trans. on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 99–112, 2005.

[11] N. R. Devanur and U. Feige, “An $o(n \log n)$ algorithm for a load balancing problem on paths,” in *Algorithms and Data Structures*. Springer, 2011, pp. 326–337.

[12] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, 1st ed. McGraw-Hill Higher Education, 1994.

[13] A. Cilardo, P. Durante, C. Lofiego, and A. Mazzeo, “Early prediction of hardware complexity in HLL-to-HDL translation,” in *Procs. of the Int. Conference on Field Programmable Logic and Applications (FPL)*, 2010, pp. 483–488.

[14] A. Cilardo, L. Gallo, A. Mazzeo, and N. Mazzocca, “Efficient and scalable OpenMP-based system-level design,” in *Procs. of Design, Automation Test in Europe Conference (DATE)*, 2013, pp. 988–991.

[15] A. Cilardo, “Fast parallel $GF(2^m)$ polynomial multiplication for all degrees,” *IEEE Trans. on Computers*, vol. 62, no. 5, pp. 929–943, 2013.

[16] A. Cilardo, N. Mazzocca, and A. Mazzeo, “Representation of elements in F_{2^m} enabling unified field arithmetic for elliptic curve cryptography,” *IET Electronics Letters*, vol. 41, no. 14, pp. 798–800, Jul. 2005.

[17] R. P. Dick, D. L. Rhodes, and W. Wolf, “TGFF: task graphs for free,” in *Procs. of the 6th Int. Workshop on Hardware/software codesign*. IEEE Computer Society, 1998, pp. 97–101.