MINIMIZING COMMUNICATION COST FOR RECONFIGURABLE SLOT MODULES

*Sándor P. Fekete, Jan C. van der Veen**

Department of Mathematical Optimization
Braunschweig University of Technology
Braunschweig, Germany
email: {s.fekete, j.van-der-veen}@tu-bs.de

Mateusz Majer†, Jürgen Teich

Department of Computer Science 12
University of Erlangen-Nuremberg
Erlangen, Germany
email: {majer, teich}@cs.fau.de

ABSTRACT

We discuss the problem of communication-aware module placement in array-like reconfigurable environments, such as the Erlangen Slot Machine (ESM). Bad placement of modules may degrade performance due to increased signal delays and wastes chip space for the reconfigurable multiple bus. We present integer linear programming (ILP) formulations that address both of these problems; both ILPs can be used stand-alone or as building blocks for more involved mathematical models. We validate our models by demonstrating their usefulness for a set of realistic benchmarks.

1. INTRODUCTION

When trying to fully exploit the enormous practical potential of dynamically reconfigurable devices such as FPGAs, a crucial issue is intermodule communication, especially in the context of module placement. This problem has yet to be solved satisfactorily.

Existing techniques for FPGAs such as by Yang et al. [1] try to reduce wiring congestion by estimating edge congestion and bin congestion. Congested regions are relieved using local improvement techniques and solving an integer linear program (ILP) for multiple congested regions. This and other algorithms, such as [2, 3], tend to be very slow, and do not generate high-quality placements. In addition, they cannot cope with problems arising from dynamic placement and routing; high run-times are also due to the fine-grain view of communicating modules.

In the online (dynamic) case with the assumption that module placement requests are generated at run-time, many approaches have been proposed, such as Bazargan et al. [4] and Yuh [5]: a reconfigurable module is modeled by a 3D box, requiring a fixed amount of resources in the $x-$ and $y-$directions, with the third dimension modeling execution time. Obvious requirements for online placement algorithms are fast runtime and good quality, typically measured in terms of low fragmentation and low rejection rate of requests. These first approaches have been extended by algorithms for placement that also model the communication effort between modules and between modules and pins: examples include [6] that considers *routing-conscious placement* for positioning a requested module such that the average Euclidean distance [6] of the placed module to all required connections is minimized; another example is [7], which uses the more realistic Manhattan distance.

All these approaches still lack a practical proof of concept because only little research has been spent on dynamically routing signals between dynamically placed modules such as automatic circuit switching (see, e.g., [8]) or dynamic networks on a chip (DyNoC, see, e.g., [9, 10, 11]).

For the offline (static) case, Teich et al. [12] showed that the problem of optimally placing a set of independent modules in space and time is a 3D strip packing problem. A breakthrough was achieved here by the reduction of the search space to equivalence classes called *packing classes*. This approach has been extended in [13] to include temporal precedence constraints between communicating modules for finding legal temporal placements. However, the concepts have yet to be verified on real hardware, mainly due to the lack of FPGA-based platforms that allow free placement of 2D modules in time, or with great restrictions about how modules can communicate with each other [14].

Dealing with the first restriction was the driving force behind introducing of a new FPGA-based platform called *Erlangen Slot Machine* [15, 16], see also Section 2 for a proper description. The architecture is slot-oriented and allows reconfiguring modules in slots of either static or dynamically adjustable width. For inter-module communication, the concept of a *reconfigurable multiple bus* [17] has been adopted and implemented using FPGA technology [16]; we call this communication medium RMBoC. For a partition of the FPGA into s slots, the RMB consists of m segments of k bits each. Each segment may be switched dynamically in order to create proper connections between two communicating modules independent from the placement.

*Supported by DFG grant FE 407/8-2, project "ReCoNodes", as part of the Priority Programme 1148, "Reconfigurable Computing".

†Supported by DFG grant TE 163/14-2, project "ReCoNodes", as part of the Priority Programme 1148, "Reconfigurable Computing".

Thus, this existing architecture gives rise to the following problems. Given a set of $n \leq s$ communicating modules; find a placement that minimizes either a) the number of segments m, or b) the maximal number of segments a signal must cross from a source to a sink slot. The second objective means minimization of the maximal delays.

The rest of this paper is organized as follows: In Section 2, we introduce the Erlangen Slot Machine and describe how it can overcome many problems of existing FPGA-based reconficurable platforms with respect to dynamic reconfigurabilty and full relocatability of synthesized hardware modules. Section 3 describes the resulting static optimization problems. In Section 4, we show corresponding ILP formulations. In Section 5, we present extended case studies including hard-to-solve artificial, but also real-world demo examples. We conclude with suggestions for future work.

2. THE ERLANGEN SLOT MACHINE (ESM)

The main idea of the Erlangen Slot Machine [15, 16] architecture is to accelerate application development as well as research in the area of partially reconfigurable hardware. The advantage of the ESM platform is its unique slot-based architecture; this allows the slots to be used independently by delivering peripheral data through a separate crossbar switch, as shown in Figure 1. The ESM architecture is based on the flexible decoupling of the FPGA I/O-pins from a direct connection to an interface chip. This flexibility allows placing application modules independently at run-time in any available slot. Thus, run-time placement is not constrained through physical I/O-pin locations, as the I/O-pin routing is done automatically in the crossbar, thus solving the I/O pin dilemma in hardware.

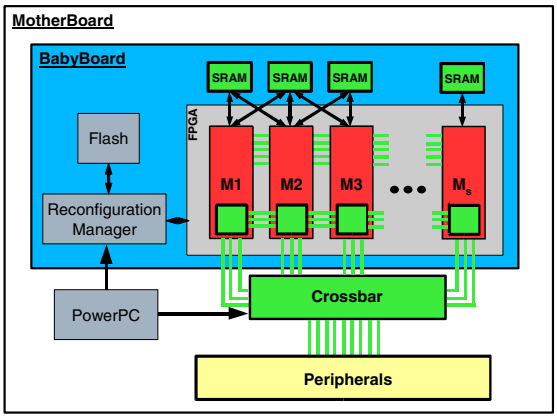

Fig. 1. Schematic overview of the ESM architecture, showing BabyBoard with slots for placing modules, connected by the reconfigurable modular bus, sitting on top of the MotherBoard.

The ESM platform is centered around one FPGA that serves as the main reconfigurable engine, and a second FPGA that realizes the crossbar switch. They are separated into two physical boards called BabyBoard and MotherBoard and implemented using a Xilinx Virtex-II 6000 and a Xilinx Spartan-II 600 FPGA. Figure 1 shows the slot-based architecture of the ESM consisting of the Virtex-II FPGA, local SRAM memories, configuration memory and a reconfiguration manager. The top pins in the north of the FPGA connect to local SRAM banks. These SRAM banks solve the problem of restricted intra-module memory, e.g., in the case of video applications. The bottom pins in the south connect to the crossbar switch. Therefore, a module can be placed in any free slot and have its own peripheral I/O-links together with dedicated local external memory. Each slot of up to 6 slots can access a local SRAM bank.

2.1. Inter-module Communication

One of the central limiting factors for the wide use of partial dynamic reconfiguration is the problem of inter-module communication; it has yet to be solved satisfactorily. Each module that is placed on one or more slots on the device must be able to communicate with other modules. For the ESM, we provide four simultaneously useable methods for communication among different modules:

(1) The first one is a direct communication using bus macros between adjacently placed modules. On the ESM, bus macros are used to realize a direct communication between adjacently placed modules, providing fixed communication channels that help to keep the signal integrity upon reconfiguration. Because only four signals can be passed for each bus macro, the number of bus macros needed for connecting a set of n signals between two placed modules is $n/4$.

(2) Secondly, it is possible to use SRAMs or BlockRAMs for shared memory communication. However, only adjacent modules can utilize these two modes of communication, which are as follows. First, dual-ported BlockRAMs can be used for implementing communication among two neighbor modules working in two different clock domains. The sender writes on one side, while the receiver reads the data on the other side. The second possibility uses external RAM. This is particularly useful in applications in which each module must process a large amount of data and then sends the processed data to the next module, as in the case of video streaming. On the ESM, each SRAM bank can be accessed by the module placed below, as well as those neighbors placed right and left. A controller is used to manage the SRAM access.

(3) For modules placed in non-adjacent slots, we provide a dynamic signal switching communication architec-

ture called Reconfigurable Multiple Bus (RMB) [18, 19, 8]. In its basic definition, the Reconfigurable Multiple Bus architecture consists of a set of processing elements or modules, each possessing an access to a set of switched bus connections to other processing elements. The switches are controlled by connection requests between individual modules. The RMB is a one-dimensional arrangement of switches between N slots. In our FPGA implementation, the horizontal arrangement of parallel switched bus line segments allows for the communication among modules placed in the individual slots. The request for a new connection is done in a wormhole fashion, where the sender (a module in slot S_k) sends a request for communication to its neighbor (slot S_{k+1}) in the direction of the receiver. Slot S_{k+1} sends the request to slot S_{k+2}, etc., until the receiver receives the request and returns an acknowledgment. The acknowledgment is then sent back in the same way to the sender. Each module that receives an acknowledgment sets its switch to connect two line segments. Upon receiving the acknowledgment, the sender can start the communication (circuit routing). The wired and latency-free connection is then active until an explicit release signal is issued by the sender module. The concept of an RMB was first presented in [19] and extended later in [18] with a compaction mechanism for quickly finding a free segment. The ESM constitutes the first implementation of this concept in real hardware.

(4) Finally, communication between two different modules can also be realized through the external crossbar.

3. MATHEMATICAL MODEL

As stated in the introduction, our goal is to allocate the modules of an application to the slots of the ESM, such that the number of parallel RMB bus segments or the maximal length of a connection between connected modules is minimized. In this section we describe technical notation and list the assumptions about the ESM and the RMB that are implicit in the integer linear programs (ILPs) to be introduced in the next section.

Given an array-like architecture (such as the ESM) that has s identical slots. As the ESM is a multiapplication platform, some of the slots may be occupied by other applications. Consequently let $\mathcal{S} \subseteq \{1, \ldots, s\}$ denote the set of available slots. Each slot has width w. There may be some extra space between two neighbouring slots j and $j+1$; this extra space is denoted by e_j. The distance between the centers of two slots $j < l$ is given by

$$d_{jl} = d_{lj} = |l - j|w + \sum_{i=j}^{l-1} e_i.$$

On this ESM we have to place an application consisting of $n \leq |\mathcal{S}|$ modules. Each of the modules is capable of executing a certain task. Some of these modules can have placement restrictions, e.g. they may require certain pins that are availabe in the first or last slot only. Let $P_i \subseteq \mathcal{S}$ denote the (possibly restricted) set of slots into which a module i can be placed. Each module may wish to communicate with other modules; in the sequel we concentrate on connections via the RMB only, after removing those that are dealt with by other means. Each of these communication links can occupy one or more RMB bus segments. The implementation of the RMB, the RMBoC, has been primarily designed for the use of the ESM in online scenarios; as a consequence, it supports unidirectional communication only. Therefore, the communication graph is a directed graph $G = (V, A)$ with weight function $t : A \to \mathbb{N}$, indicating the number of bus segments necessary for realizing an edge on the RMB.

In this more formal setting, the task of placing modules on the ESM in order to minimize one of the two objectives given above reduces to the problem of placing the vertices of the communication graph on the integer points of the line. These problems are variations of the classical optimization problem called *minimum bandwidth problem* (MBW); see [20] for a recent computational study.

4. ILP MODELS

In this section we concentrate on two ILP models that map modules to slots such that

- the number of parallel bus segments is minimized (see subsection 4.1)
- the maximum distance between any two connected modules is minimized subject to a restricted number of parallel bus segments (see subsection 4.2.)

In these ILPs there are two kinds of variables. Variable $x_{ij} \in \{0, 1\}$ is a binary variable indicating whether a module $i \in \{1, \ldots, n\}$ is placed in slot $j \in \{1, \ldots \mathcal{S}\}$ or not. If a slot j is not available for the current application, the variables x_{ij} are fixed to zero for all modules i. If a module i must not be placed in slot j, i.e., $j \notin P_i$, x_{ij} is fixed to zero as well. For technical reason we also have variables $0 \leq x_{ijkl} \leq 1$. Even though these variables are not binary, they can only take values in $\{0, 1\}$: if two modules i and k that are connected by an edge ik are placed in slots j and l respectively x_{ijkl} is set to one; otherwise it is set to zero. We will discuss our ILPs for these types of variables in the next two sections.

For illustrating these ILPs, we apply them to our ESM-based implementation of the classical Pong video game. This video game consists of four modules for user input, racket position calculation, ball position calculation, and video interface. The user input module sends data to the racket position module. The racket position is needed in the ball position calculation and in the display module. The position

of the ball is sent to to the display module. All in all, the communication graph has four vertices and five edges with edge weights in $\{4, 20, 38\}$.

4.1. Minimizing the Number of Parallel Segments

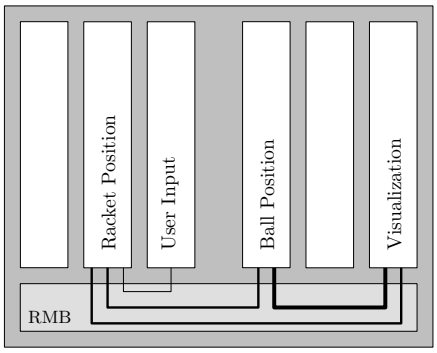

Fig. 2. An optimal solution of the ILP presented in section 4.1 for the Pong example. All four modules are placed on the ESM. The thin, medium, and heavy lines connecting the modules represent 4, 20, and 38 segments, respectively. The maximum number of parallel segments is 58. Note that the length of the longest connection is 4.

Each of the modules i has to be placed in any of the available slots. On the other hand, each of the slots j can hold at most one module. This can be expressed by the following two assignment constraints:

$$\sum_{j \in \mathcal{S}} x_{ij} = 1 \qquad i \in \mathcal{M}, \quad (1)$$

$$\sum_{i=1}^{n} x_{ij} \leq 1 \qquad j \in \mathcal{S}. \quad (2)$$

In this section we aim at minimizing the maximum number of bus segments in use. In order to count the number t_s of parallel segments crossing the border of slots s and $s+1$, we could formulate a constraint like

$$t_s = \sum_{jl \in A : j \leq s < l} \sum_{i=1}^{n} \sum_{k=1}^{n} t_{jl} x_{ij} x_{kl}. \quad (3)$$

Unfortunately, this equation is quadratic in x. As x_{ij} and x_{kl} are binary variables, this constraint can be linearized. To do so we introduce variables x_{ijkl} and add the following constraint:

$$x_{ijkl} \geq x_{ij} + x_{kl} - 1. \quad (4)$$

Equation (4) is nothing but a logical AND. x_{ijkl} is set to one if and only if neither of x_{ij} and x_{kl} is set to zero. Replacing the product in (3) by the new variables results in

$$t_s = \sum_{jl \in A : j \leq s < l} \sum_{i=1}^{n} \sum_{k=1}^{n} t_{jl} x_{ijkl}. \quad (5)$$

Adding the equation

$$t_s \leq T \qquad s \in \mathcal{S} \setminus \max_{j \in \mathcal{S}} j, \quad (6)$$

we can now formulate the integer linear program

$$\text{Minimize } T \quad (7)$$

subject to

(1), (2), (4), (5), and (6)

$$x_{ij} = 0 \qquad i \in \{1, \ldots, n\}, j \notin \mathcal{S}, \quad (8)$$

$$x_{ij} = 0 \qquad i \in \{1, \ldots, n\}, j \notin P_i, \quad (9)$$

$$x_{ij} \in \{0, 1\}, \quad (10)$$

$$0 \leq x_{ijkl} \leq 1, \quad (11)$$

Applying this ILP to the Pong application described above yields a solution as shown in Figure 2. The minimal number of parallel segments is 58. Even though this solution is optimal, it has the deficiency of unnecessarily long connections. We will address this problem in the next subsection.

4.2. Minimize Segment-Constrained Bandwidth

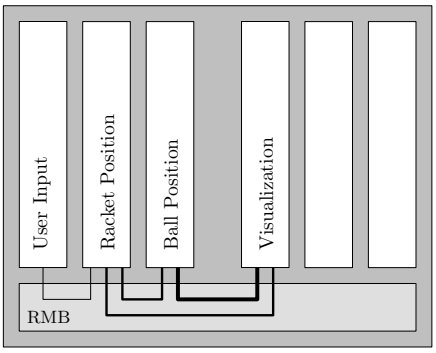

Fig. 3. An optimal solution of the ILP presented in section 4.2 for the Pong example. The solution is restricted to a minmum number of 58 parallel segments. The length of the longest connection is now 2.

Making use of the ILP presented in the previous section, we are able to compute the minimal number parallel segments in use, thereby minimizing the space overhead caused by the RMB. Given this minimal number of segments or at least the number T of parallel bus segments available, the

ILP presented in this subsection finds a placement of the modules that minimizes the length L of the longest edge between any two connected modules. This number L can be determined by adding the linear inequality

$$L \geq d_{jl} x_{ijkl} \qquad ik \in A, j, l \in \mathcal{S} \quad (12)$$

to the ILP described above. Given this additional constraint, the ILP can be formulated as

$$\text{Minimize } L \quad (13)$$
$$\text{subject to}$$
$$(1), (2), (4) - (6), \text{ and } (8) - (12).$$

Applying this ILP with T set to 58 to the Pong application results in a placement as in Figure 3. The length of the longest edge has been reduced to 2.

5. CASE STUDY AND RESULTS

The above optimization problems produce placements that are best possible with respect to number of parallel segments (i. e. area requirement oft the RMB) or wire length (i. e. delay). Solving the resulting ILPs can be done in less than a tenth of a second for the small example used for illustration in the previous section. In this section we demonstrate the ability of our approach to tackle much larger problems.

n	$\|E_1\|$	time/s	$\|E_2\|$	time/s	$\|E_3\|$	time/s
6	8	0.04	8	0.04	10	0.05
8	10	0.14	10	0.13	13	0.15
10	12	0.45	13	0.34	16	0.46
12	14	0.82	16	0.96	20	1.21
14	16	1.80	18	2.55	23	3.72
16	18	4.40	21	2.54	26	18.15

Table 1. Results for the ILP presented in section 4.1. Here n is the number of vertices, $|E_1| = n+2$, $|E_2| = \lfloor \frac{4n}{3} \rfloor$, and $|E_1| = \lfloor \frac{5n}{3} \rfloor$ are the numbers of edges of the three types of communication graphs considered. After performing 7 runs, time/s is the arithmetic mean taken over five runs, deleting the best and the worst result.

For this purpose we have randomly generated a large number of artificial benchmark instances. We assume an application consisting of n modules; all of these modules require some kind of communication with other modules; as in practice, the communication graph is sparse. Like in most applications, data is passed from one module to the next and we add an edge $(i, i+1)$ for $i \in \{1, \ldots n-1\}$ to the communication graph. We randomly add edges between unconnected modules, until the graph has a certain number of edges. For each n we have three different types of graphs: G_1, G_2, and G_3, having $|E_1| = n+2$, $|E_2| = \lfloor \frac{4n}{3} \rfloor$, and $|E_3| = \lfloor \frac{5n}{3} \rfloor$ edges, respectively. The edge weight of each edge is taken from the set $\{1, \ldots, 8\}$ uniformly at random. This application has to be placed on an ESM with n slots.

For each n we have generated seven instances for each of the graph types. The resulting ILPs were solved on an AMD Athlon64 X2 3800+, equipped with 2GB RAM, running under Linux, using CPLEX 10.0 as the ILP solver. The instances with the best and the worst running time were discarded; we report only on the average running time of the five remaining instances.

n	$\|E_1\|$	time/s	$\|E_2\|$	time/s	$\|E_3\|$	time/s
6	8	0.14	8	0.15	10	0.19
8	10	0.89	10	0.89	13	1.43
10	12	3.90	13	5.25	16	10.96
12	14	30.06	16	44.64	20	80.95
14	16	153.74	18	573.27	23	775.71
16	18	2655.44	21	2812.81	26	> 3600.00

Table 2. Results for the ILP presented in section 4.2. All columns are as in Table 1.

In Table 1 we give our results for the ILP presented in section 4.1. Table 2 shows our results for the ILP of section 4.2. Our current implementation of the RMB for the ESM supports slots for six modules only. The first line of either table clearly demonstrates that optimizing for our status quo poses no challenge to these ILP formulations with state-of-the-art ILP solvers. However, each of the six ESM slots decomposes into three so-called micro slots. There is a total of four extra micro slots, located on the left, in the center, and on the right of the ESM, so the current ESM has 22 micro slots. Due to the space overhead of the RMB we are not planning to connect each of the micro slots to the RMB; 16 modules is a realistic number, which we can still solve reasonably well.

As can be seen from Table 1, the running time increases very moderately as the number of edges increases. This is due to the fact that all three graph types are rather sparse. Increasing the number of modules has a stronger effect on the running time. In any case, all instances can be solved within very reasonable time.

The basic same effects can be seen in Table 2. As this problem is a constrained version of the bandwidth problem, the consequences are much stronger. While running times for the first table were almonst negligible, we are currently not able to solve instances with more than 16 modules in less than an hour.

6. CONCLUSION

In this paper we describe how to deal with the practical problem of communication on a reconfigurable device. Based on

an array-like architecture, we get a set of important optimization problems that can be solved with ILP techniques.

There is a considerable number of extensions and generalizations that warrant further study. Non-uniform widths of channels lead to additional selection problems; non-uniform widths of modules lead to additional packing problems; broadcast of signal values to multiple listening modules requires extension of the RMB to multicast.

Another set of problems arises from geometries that are more general than the ESM: when progressing from a single linear channel of slots to a more tree-like architecture, the combinatorial structure of routing gets more involved.

For all of those problems we are optimistic that the combination of practical know-how with expertise from mathematical optimization will lead to further progress.

7. REFERENCES

[1] X. Yang, M. Wang, R. Kastner, S. Ghiasi, and M. Sarrafzadeh, "Fast Placement Approaches for FPGAs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 8, no. 3, pp. 316–333, 2003.

[2] J. Shi and D. Bhatia, "Performance Driven Floorplanning for FPGA Based Designs," in *International Symposium on Field Programmable Gate Arrays(FPGA)*, Monterey, CA, U.S.A., 1997, pp. 112–118.

[3] R. G. Tessier, "Fast place and route approaches for fpgas," Ph.D. dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge, MA, U.S.A., 1999.

[4] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast Template Placement for Reconfigurable Computing Systems," *IEEE Design and Test of Computers*, vol. 17, pp. 68 – 83, 2000.

[5] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang, "Temporal Floorplanning Using the T-tree Formulation," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, U.S.A., 2004, pp. 300–305.

[6] A. Ahmadinia, C. Bobda, M. Bednara, and J. Teich, "A New Approach for On-line Placement on Reconfigurable Devices," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS) / Reconfigurable Architectures Workshop (RAW)*, Santa Fe, NM, U.S.A., 2004, p. 134.

[7] A. Ahmadinia, C. Bobda, S. Fekete, J. Teich, and J. van der Veen, "Optimal Routing-Conscious Dynamic Placement for Reconfigurable Devices," in *Field-Programmable Logic and Applications, International Conference*, Antwerp, Belgium, 2004, pp. 847–851.

[8] A. Ahmadinia, C. Bobda, J. Ding, M. Majer, J. Teich, S. Fekete, and J. van der Veen, "A Practical Approach for Circuit Routing on Dynamically Reconfigurable Devices," in *Proc. of the 16th IEEE International Workshop on Rapid System Prototyping (RSP)*, Montreal, Canada, 2005, pp. 84–90.

[9] C. Bobda, A. Ahmadinia, D. Koch, M. Majer, and J. Teich, "A Dynamic NoC Approach for Communication in Reconfigurable Devices," in *Field-Programmable Logic and Applications, International Conference*, Antwerp, Belgium, 2004, pp. 1032–1036.

[10] M. Majer, C. Bobda, A. Ahmadinia, and J. Teich, "Packet Routing in Dynamically Changing Networks on Chip," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS) / Reconfigurable Architectures Workshop (RAW)*, Denver, CO, U.S.A., 2005, p. 154.

[11] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, and J. van der Veen, "DyNoC : A Dynamic Infrastructure for Communication in Dynamically Reconfigurable Devices," in *Field-Programmable Logic and Applications, International Conference*, Tampere, Finland, 2005, pp. 153–158.

[12] J. Teich, S. Fekete, and J. Schepers, "Compile-Time Optimization of Dynamic Hardware Reconfigurations," in *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, NV, U.S.A., 1999, pp. 1097–1103.

[13] S. P. Fekete, E. Köhler, and J. Teich, "Optimal FPGA Module Placement with Temporal Precedence Constraints," in *Proceedings of the Design Automation and Test in Europe (DATE)*, Munich, Germany, 2001, pp. 658–665.

[14] H. Walder and M. Platzner, "Online Scheduling for Block-partitioned Reconfigurable Devices," in *Proceedings of the Design Automation and Test in Europe (DATE)*, Munich, Germany, 2003, pp. 290–295.

[15] C. Bobda, M. Majer, A. Ahmadinia, T. Haller, A. Linarth, J. Teich, S. Fekete, and J. van der Veen, "The Erlangen Slot Machine: A Highly Flexible FPGA-Based Reconfigurable Platform," in *IEEE Symp. on FPGAs and Custom Computing Machines (FCCM)*, Napa, CA, U.S.A., 2005, pp. 319–320.

[16] C. Bobda, M. Majer, A. Ahmadinia, T. Haller, A. Linarth, and J. Teich, "Increasing the flexibility in fpga-based reconfigurable platforms: The erlangen slot machine," in *IEEE 2005 Conference on Field-Programmable Technology (FPT)*, Singapore, Singapore, dez 2005, pp. 37–42.

[17] H. A. ElGindy, A. K. Somani, H. Schroeder, H. Schmeck, and A. Spray, "RMB - A Reconfigurable Multiple Bus Network," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, San Jose, CA, U.S.A., 1996, pp. 108–117.

[18] H. A. ElGindy, A. K. Somani, H. Schröder, H. Schmeck, and A. Spray, "RMB - a reconfigurable multiple bus network," in *Proceedings of the Second International Symposium on High-Performance Computer Architecture (HPCA-2)*, San Jose, California, Feb. 1996, pp. 108–117.

[19] R. Vaidyanathan and J. L. Trahan, *Dynamic Reconfiguration: Architectures and Algorithms*. IEEE Computer Society, 2003.

[20] A. Caprara and J.-J. Salazar-Gonzalez, "Laying out sparse graphs with provably minimum bandwidth," *INFORMS Journal on Computing*, vol. 17, pp. 356–373, 2005.