FPGA-area Allocation for Partial Run-Time Reconfiguration

Elena Moscu Panainte, Koen Bertels, and Stamatis Vassiliadis
Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2600 GA Delft, The Netherlands
Phone: +31 15 2786249 Fax: +31 15 2784898
E-mail: {elena|koen|stamatis}@ce.et.tudelft.nl

Abstract— **Although the new generations of FPGAs provide support for partial and dynamic configuration, the huge reconfiguration latency is still a major shortcoming of the current FCCMs . Software and hardware techniques (compiler optimizations, configuration prefetching) have been used in order to reduce the impact of the configuration overhead on the overall performance. Nevertheless, these techniques may not produce significant performance improvements when the hardware implementations of the operations executed on the FPGA are not properly placed on the target FPGA. For example, when three configurations should be placed on the target FPGA but only two of them can fit in the available FPGA area, then the placement of the third tasks will overlap at least with one other configuration. In such cases, a strategy is required to determine the optimal tasks placement on the target FPGA.**

In this paper, we propose two FPGA-area allocation algorithms for the tasks executed on the reconfigurable hardware. The goal is to minimize the FPGA-area which is reconfigured at runtime, taking into account the application runtime features. More specifically, we use the reconfiguration frequency for the target application to guide the allocation algorithms. Two scenarios are discussed: the first one corresponds to the case when all hardware operations must be placed/executed on the target FPGA while in the second scenario, a hardware operation can be switched to its pure software execution on the core processor in order to reduce the pressure/competition for the FPGA area. The FPGA-area allocation problem is formulated as a 0-1 integer linear programming (LP) problem and efficient LP solvers are used for finding the optimal solutions.

I. Introduction

Although the new generations of FPGAs provide support for partial and dynamic configuration, the huge reconfiguration latency is still a major shortcoming of the current FCCMs (see [1], [2]). Software and hardware techniques (compiler optimizations, configuration prefetching) have been used in order to reduce the impact of the configuration overhead on the overall performance. Nevertheless, these techniques may not produce significant performance improvements when the hardware implementations of the operations executed on the FPGA are not properly placed on the target FPGA. For example, when three configurations should be placed on the target FPGA but only two of them can fit in the available FPGA area, then the placement of the third tasks will overlap at least with one other configuration. In such cases, a strategy is required to determine the optimal tasks placement on the target FPGA.

In this paper, we propose two FPGA-area allocation algorithms for the tasks executed on the reconfigurable hardware. The goal is to minimize the FPGA-area which is reconfigured at runtime and improve the overall performance, taking into account the application runtime features. More specifically, we use the reconfiguration frequency for the target application to guide the allocation algorithms. Two scenarios are discussed: the first one corresponds to the case when all hardware operations must be placed/executed on the target FPGA while in the second scenario, a hardware operation can be switched to its pure software execution on the core processor in order to reduce the pressure/competition for the FPGA area. The FPGA-area allocation problem is formulated as a 0-1 integer linear programming (LP) problem and efficient LP solvers are used for finding the optimal solutions.

The paper is organized in five sections. The background and related work is presented in the following section. Next, we discuss some motivational examples and define the FPGA-area allocation problem addressed in this paper. The proposed allocation algorithms are detailed in section IV. Finally, we present conclusions and future work.

II. Background and Related Work

In this paper, we assume the Molen programming paradigm [3] for FCCMs (Field-programmable Custom Computing Machines) with a core processor (GPP) and reconfigurable hardware (usually implemented as an FPGA).

The reconfigurable hardware is controlled by two instructions: i) SET for the FPGA configuration for a reconfigurable operation (Rop) and ii) EXECUTE for the Rop execution on the FPGA. The Molen compiler [4] generates code for reconfigurable computing platforms following the Molen programming paradigm. An important compiler optimization (see [2]) included in the Molen compiler is to reduce the redundant SET instructions taking into account the predefined FPGA-area conflicts between the considered Rops. In consequence, the compiler optimization will benefit from an efficient FPGA-area allocation that minimizes the FPGA-area overlaps for a target application.

Previous approaches for FPGA-area allocation are mainly focused on cases where the whole application is decomposed in tasks which are all executed on the FPGA. In [5], an optimal module placement based on packing classes is proposed. A backtracking solution with bounding heuristics is presented in [6]. The proposed solutions require detailed information (such as data flow graphs, dependency graphs of tasks) about the application's features and regular application behavior. Another approach (see [7] [8] [9]) is the task allocation in an operating system for reconfigurable computing. In such cases, information about specific application behavior cannot be used in order to guide this allocation, thus optimization opportunities can be lost. Other related work [10] addresses compiler optimization for reducing the number of redundant FPGA configurations based on a predefined FPGA-area allocation. In the current paper, we propose two FPGA-area allocation algorithms that reduce furthermore the number of FPGA configurations by minimizing the total reconfigured area for a given trace of execution.

III. Problem Overview and Definition

Motivational Example: In order to clearly define the FPGA-area allocation problem, we use a motivational example (Figure 1(a)) which sketches an FPGA device and the area requirements for three operations implemented on the FPGA. In this paper, we assume FPGAs with column-based reconfiguration (the reconfiguration may only be performed for a full column of CLBs of the chip) such as the well-known Xilinx Virtex devices. For one application that uses the three hardware operations, a simple FPGA area allocation (presented in 1(b)) places all operations starting with the first column. Due to the FPGA area overlaps, such allocation requires the FPGA reconfiguration before each execution of the considered operations. As shown in [2] and [10], FPGA reconfiguration is slow and thus, repetitive FPGA reconfigurations can produce a significant performance decrease. In consequence, a better FPGA-area allocation is required in order to reduce the reconfiguration overhead. An allocation strategy is possible only when the placement of the hardware operations is not predefined.

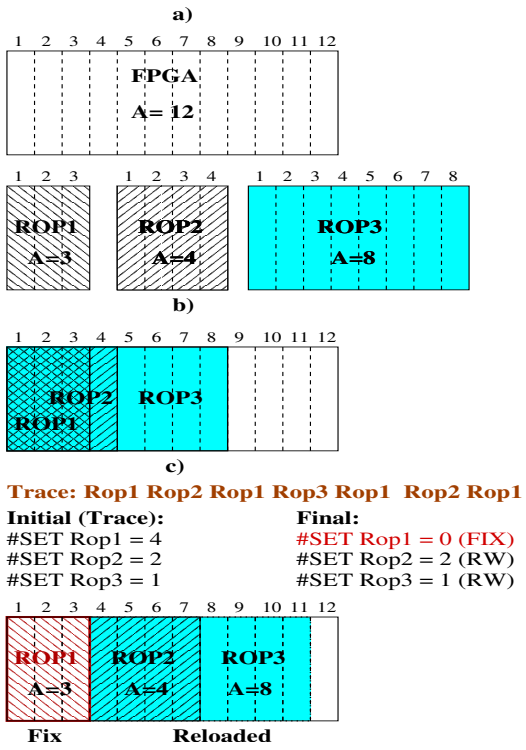

Fig. 1. Example: a) Total FPGA-area and required area for three hardware operations; b) a simple FPGA-area allocation c) optimal allocation based on the execution trace

Two important observations can be made regarding the example from Figure 1. The first observation is that the three considered operations cannot fit together on the FPGA as the sum of the area of their hardware implementations exceeds the total available FPGA-area. The second observation concerns the simple allocation strategy, where there is unused FPGA-area while parts of the FPGA have to be reconfigured before each execution. For the considered example, even when the Rop2 and Rop3 do not have overlapping FPGA-area, the placement of Rop1 will introduce FPGA-area overlaps with one of the two operations.

In order to determine an efficient FPGA-area allocation, we propose an approach that divides the hardware operations in two categories: FIX and RW. An operation is called FIX if it has no overlapping area with any other hardware operations in the considered application. Such a FIX operation requires only one initial FPGA configuration (which can be preloaded and can be neglected). An operation is called RW (reconfigurable) if its area overlaps with other operations and it has to be configured before each execution.

Loosely stated, the main idea of our approach is to minimize the reconfigured FPGA-area based on the reconfigu-

ration frequency of each operation. Using profiling information, we determine the execution order for the hardware operations (called trace) and compute the reconfiguration frequency in the trace. The goal is to allocate the larger and frequently reconfigured operations as FIX operations. The example shown in Figure 1(c) presents the optimal FPGA-area allocation for a given execution trace. We can observe the elimination of hardware configurations for the operations allocated as FIX operations (Rop1 in this example). The selection of the FIX operations is based on 0-1 linear programming and is explained in Section IV. The used terminology and a formal description of the allocation problem is presented in the rest of this section.

Problem statement: We represent a set of n reconfigurable hardware operations (Rops) as $ROP = \{Rop_1, Rop_2, ..., Rop_i, ..., Rop_n\}$, where each operation Rop_i occupies for its hardware implementation an FPGA-area A_i. The total available area of the target FPGA device is S. Although in this paper we address the case when the reconfiguration is column-based (the area is expressed as the number of columns), the extension to the 2D or 3D cases is straightforward. An execution trace is a sequence of Rops that are executed for a set of representative input data for the target application and it is represented as $T : Rop_i, Rop_j, ..., Rop_k, ...$ A trace is normalized if it does not contain two identical consecutive Rops. This normalization represents the fact that consecutive hardware reconfigurations for the same Rop are redundant and can be eliminated by compiler optimization (see [2]) or hardware prefetching. For each $Rop_i \in ROP$ and a normalized trace T, the reconfiguration frequency $n(T)_i$ represents the number of occurrences of Rop_i in the trace T.

As previously explained, the idea of our approach is to divide the ROP set in two subsets FIX and RW, where $ROP = FIX \bigcup RW$ and $FIX \bigcap RW = \emptyset$. The Rops in the FIX set will have a dedicated area allocated on the FPGA that is not used by other Rops (they do not have area overlaps with other Rops). The advantage is that the FIX Rops will not require an FPGA reconfiguration before their executions. The total area occupied by the FIX Rops is $\sum_{Rop_j \in FIX} A_j$. The Rops in RW set are the operations that have area overlaps. The reconfiguration overhead is proportional with the FPGA-area which is reconfigured at runtime. The aim is to minimize the total reconfigured area (the sum of the area of the Rops from RW multiplied by their reconfiguration frequency) which corresponds to the minimization of the reconfiguration overhead and implicitly, to the improvement of the overall performance gain. A formal description of this problem is as follows:

Problem Given a set $ROP = \{Rop_1, ..., Rop_i, ..., Rop_n\}$, a total available FPGA-area S, a normalized execution trace T, each Rop_i having an FPGA-area A_i and the reconfiguration frequency $n(T)_i$, find $RW \subseteq ROP$ that minimizes the reconfigured area $\sum_{Rop_i \in RW} n(T)_i * A_i$, under the following constraint:

- $\forall Rop_k \in RW, A_k + \sum_{Rop_j \in FIX} A_j \leq S$, where $FIX = ROP - RW$.

The constraint represents the requirement that any RW Rop must have enough available area to coexists on the FPGA at the execution time with all FIX Rops. Implicitly, as the FPGA-area is a positive number, the constrain expresses also the requirement that all FIX Rops should fit together on the target FPGA. Once the RW set has been determined for the above mentioned problem, an effective FPGA-area allocation is straightforward. Assuming that A_i represents the number of required columns, an FPGA-area allocation associates with each Rops, the number of the first column where A_i is placed. In the first step, the FIX Rops are consecutively allocated on the FPGA. In the second step, the RW Rops are all allocated at the end of the FPGA-area allocated for the FIX Rops.

IV. FPGA-area Allocation Algorithms

For the problem defined in the previous section, we propose its formulation as an integer linear pseudo-Boolean (0-1) programming problem and consequently, the solutions can be determined using efficient solvers (see [11]). More specifically, we propose two scenarios. The first case (associated with the FIX/RW Algorithm) corresponds to the above mentioned problem, where the Rops are placed in the FIX or in the RW (Reloaded) part on the FPGA. In the second case (corresponding to the FIX/RW/SW Algorithm), we assume than an Rop can have three options for execution: on the FIX or RW part or additionally, it can be switched to its software execution (on GPP). The last options can be preferred for those Rops where the huge reconfigurations latency consumes the gain produced by the fast execution on the FPGA. In the rest of this section, we introduce in detail the two FPGA-area allocation algorithms.

A. FIX/RW FPGA-area Allocation Algorithm

As previously presented, we translate the FPGA-area allocation problem in a 0-1 linear programming problem to produce an optimal solution using efficient solvers.

0-1 Selection In the considered case, any Rop can be executed on the FIX or RW part of the FPGA. In consequence, we associate with any Rop_i a variable x_i such that

$x_i = \begin{cases} 0 & \text{if } Rop_i \in FIX \\ 1 & \text{if } Rop_i \in RW \end{cases}$. Finding the optimal partition of ROP in FIX and RW is reduced to finding the optimal 0-1 values for all x_i.

Objective function In the problem definition in Section III, the minimization of the reconfigured area $\sum_{Rop_i \in RW} n(T)_i * A_i$ can be expressed as the following objective function $\sum_{Rop_i \in ROP} n(T)_i * A_i * x_i$. If Rop_i is a FIX Rop, then $x_i = 0$ and it does not increase the reconfigured area as it does not need any configuration. In consequence, only the contribution of the RW Rops is included in the minimization objective function.

The system of linear pseudo-Boolean inequalities of the linear programming problem formulation corresponds to the constrains included the initial problem. The constraint that $\forall Rop_k \in RW$, $A_k + \sum_{Rop_j \in FIX} A_j \leq S$ can be expressed as follows:

$$\begin{cases} A_1 * x_1 + \sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S \\ A_2 * x_2 + \sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ A_i * x_i + \sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ A_n * x_n + \sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S \end{cases}$$

This system of inequalities should be interpreted as follows: (1) The term $\sum_{Rop_j \in ROP} A_j * \bar{x}_j$ represents the permanently configured FPGA-area occupied by FIX Rops:

$$\left.\begin{array}{l} \sum_{Rop_j \in ROP} A_j * \bar{x}_j = \sum_{Rop_j \in FIX} A_j * \bar{x}_j + \sum_{Rop_j \in RW} A_j * \bar{x}_j \\ Rop_j \in RW \Longrightarrow x_j = 1 \Longrightarrow \bar{x}_j = 0 \end{array}\right\} \Longrightarrow$$

$$\sum_{Rop_j \in ROP} A_j * \bar{x}_j = \sum_{Rop_j \in FIX} A_j * \bar{x}_j.$$

(2)The second observation regards the first term in the inequalities, namely $A_i * x_i$. For the cases when $Rop_i \in FIX \Longrightarrow x_i = 0$, the term $A_i * x_i$ can be eliminated. The ith inequality is transformed in $\sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S$ which represents the constraint that the total area allocated for FIX Rops should be smaller or equal than the total available FPGA-area S. Similarly, for the cases when $Rop_i \in RW \Longrightarrow x_i = 1$, the inequality is transformed in $A_i * x_i + \sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S$ which represents the constraint that an RW Rop has to fit on the FPGA together with all FIX Rops.

In our model implementation, each ith inequality should not contain both x_i and $\bar{x}_i$; thus it can be reduced as follows:

$$A_i * x_i + \sum_{j=1}^{n} A_j * \bar{x}_j \leq S \iff A_i * x_i + A_i * \bar{x}_i + \sum_{j=1}^{i-1} A_j * \bar{x}_j + \sum_{j=i+1}^{n} A_j * \bar{x}_j \leq S \iff$$

$$A_i * (x_i + \bar{x}_i) + \sum_{j=1}^{i-1} A_j * \bar{x}_j + \sum_{j=i+1}^{n} A_j * \bar{x}_j \leq S \iff \sum_{j=1}^{i-1} A_j * \bar{x}_j + \sum_{j=i+1}^{n} A_j * \bar{x}_j \leq S - A_i$$

min:	+2*39*x1	+ 3*13*x2	+ 3*16*x3;	
C1:		+ 13*$\bar{x}$2	+ 16*$\bar{x}$3	$\leq$ 58 - 39
C2:	+ 39*$\bar{x}$x1		+ 16*$\bar{x}$x3	$\leq$ 58 - 13
C3:	+ 39*$\bar{x}$x1	+ 13*$\bar{x}$x2		$\leq$ 58 - 16

Fig. 2. The linear problem description for the MPEG2 encoder and FIX/RW Algorithm

Example An example is presented in Figure 2, for three Rops with $A_1 = 39, A_2 = 13, A_3 = 16, n(T)_1 = 2, n(T)_2 = 3, n(T)_3 = 3$ and $S = 58$. The solution to this problem is $\{x_1 = 0, x_2 = 1; x_3 = 1\}$, which corresponds to $FIX = \{Rop_1\}$ and $RW = \{Rop2, \quad Rop3\}$.

B. FIX/RW/SW FPGA-area Allocation Algorithm

The FIX/RW algorithm previously presented has two important limitations: i) it cannot find a viable FPGA allocation if there is an Rop_i with $A_i > S$ because the constraint set is unsatisfiable; and ii) although the FPGA execution is (usually) faster than the software execution for any Rop, the reconfiguration overhead can significantly increase the overall execution time. In order to eliminate these rigid limitations, we propose the FIX/RW/SW algorithm where the Rops can additionally be switched to software execution. The FPGA-area allocation problem can again be formulated as 0-1 LP problem including the following components.

0-1 Selection In this case, a Rop has three options for execution: on the FIX or RW part on the FPGA or additionally in software (SW). The allocation problem involves the division of ROP in three subsets FIX, RW and SW, such that $ROP = FIX \bigcup RW \bigcup SW$ and $FIX \bigcap RW = \emptyset$, $FIX \bigcap SW = \emptyset$, $RW \bigcap SW = \emptyset$. These options can be expressed using three boolean variables for each Rop_i, namely $xfix_i, xrw_i$ and xsw_i, where $xfix_i = \begin{cases} 1 & \text{if } Rop_i \in FIX \\ 0 & \text{if } Rop_i \notin FIX \end{cases}$ and similar for xrw_i and xsw_i. Moreover, a Rop must be included in only one subset; this constraint can be expressed as $xfix_i + xrw_i + xsw_i = 1$. Finding the optimal partition of ROP in FIX, RW and SW is reduced to finding the optimal 0-1 values for all $xfix_i, xrw_i$ and xsw_i.

$$
\begin{array}{lllll}
\text{min:} & +cost\_fix_1 * xfix_1 & +cost\_fix_2 * xfix_2 & +cost\_fix_3 * xfix_3 + \\
& +cost\_rw_1 * xrw_1 & +cost\_rw_2 * xrw_2 & +cost\_rw_3 * xrw_3 + \\
& +cost\_sw_1 * xsw_1 & +cost\_sw_2 * xsw_2 & +cost\_sw_3 * xsw_3 +
\end{array}
$$

$$
\begin{array}{llllll}
\text{C1:} & xfix_1 & +xrw_1 & +xsw_1 & = 1 & \\
\text{C2:} & xfix_2 & +xrw_2 & +xsw_2 & = 1 & \\
\text{C3:} & xfix_3 & +xrw_3 & +xsw_3 & = 1 & \\
\text{C4:} & 39\,xrw_1 & +39\,xfix_1 & +13\,xfix_2 & +16\,xfix_3 & \leq 58 \\
\text{C5:} & 13\,xrw_2 & +39\,xfix_1 & +13\,xfix_2 & +16\,xfix_3 & \leq 58 \\
\text{C4:} & 16\,xrw_3 & +39\,xfix_1 & +13\,xfix_2 & +16\,xfix_3 & \leq 58
\end{array}
$$

Fig. 3. The linear problem description for the MPEG2 encoder and FIX/RW/SW Algorithm

Objective function In the problem definition of the previous FIX/RW Algorithm, the goal of the objective function is the minimization of the total reconfigured area. This function cannot be used in the current scenario as all Rops can be switched to their software execution; thus in the FIX/RW/SW algorithm, the goal is the performance gain. The new objective function is the minimization of the execution time for the considered Rops and is expressed as $\sum_{i=1}^{n} cost\_fix_i * xfix_i + \sum_{i=1}^{n} cost\_rw_i * xrw_i + \sum_{i=1}^{n} cost\_sw_i * xsw_i$, where $cost\_fix_i/cost\_rw_i/cost\_sw_i$ represent the total execution time for Rop_i in FIX/RW/SW respectively and their values can be determined using profiling information and estimations.

Linear Pseudo-Boolean Inequalities The system of linear pseudo-Boolean inequalities of the linear programming problem formulation is similar to the previous FIX/RW system:

$$
\begin{cases}
A_1 * xrw_1 + \sum_{j=1}^{n} A_j * xfix_j \leq S \\
A_2 * xrw_2 + \sum_{j=1}^{n} A_j * xfix_j \leq S \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
A_i * xrw_i + \sum_{j=1}^{n} A_j * xfix_j \leq S \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
A_n * xrw_n + \sum_{j=1}^{n} A_j * xfix_j \leq S
\end{cases}
$$

The main idea is the same as in the previous algorithm: each RW Rop must have allocated enough FPGA-area to fit with all FIX Rops on the FPGA.

As a final observation for both algorithms, we notice that the generated FPGA-area allocations will preserve the application semantics even if the input execution trace T is not a representative trace. In such cases, some performance gain may be lost, but the application has the correct behavior.

Example One linear model for the three Rops presented previously presented and FIX/RW/SW Algorithm is presented in Figure 3. For the estimated costs, the solution to this linear problem is $\{xfix_1 = 1, xfix_2 = 1, xsw3 = 1\}$, while the other boolean variables are zero.

V. Conclusions

In this paper, we have presented two FPGA-area allocation algorithms for minimizing the huge reconfiguration overhead of the current FPGAs.Two scenarios have been proposed: the traditional placement problem when all Rops are executed on the FPGA and additionally, the case when any Rop can be switched to its software execution. The algorithms incorporate 0-1 LP solvers and use the reconfiguration frequency for finding the optimal Rops allocation.

In our future work, we will extend the allocation algorithms to take into account not only the reconfiguration frequency, but also the reconfiguration order of the considered Rops in a representative execution trace; we notice that this new problem is not a linear programming problem. We also intend to exploit the parallelism for Rops execution on the FPGA and between the FPGA and the core processor.

References

[1] Moscu Panainte, E., Bertels, K., Vassiliadis, S.: Dynamic hardware reconfigurations: Performance impact on mpeg2. In: Proceedings of SAMOS. Volume 3133., Samos, Greece, Springer-Verlag Lecture Notes in Computer Science (LNCS) (2004) 284–292

[2] Panainte, E.M., Bertels, K., Vassiliadis, S.: Instruction scheduling for dynamic hardware configurations. In: Proceedings of Design, Automation and Test in Europe (DATE 05), Munich, Germany (2005) 100–105

[3] Vassiliadis, S., Wong, S., Gaydadjiev, G.N., Bertels, K., Kuzmanov, G., Moscu Panainte, E.: The Molen Polymorphic Processor. IEEE Transactions on Computers **53(11)** (2004) 1363–1375

[4] Moscu Panainte, E., Bertels, K., Vassiliadis, S.: The PowerPC backend molen compiler. In: FPL. Volume 3203., Antwerp, Belgium, Springer-Verlag Lecture Notes in Computer Science (LNCS) (2004) 434–443

[5] Fekete, S., Khler, E., Teich, J.: Optimal fpga module placement

with temporal precedence constraints. In: Proceedings of Design, Automation and Test in Europe (DATE 01). (2001) 658–665
[6] Maestre, R., Kurdahi, F.J., Bagherzadeh, N., Singh, H., Hermida, R., Fernndez:, M.: Kernel scheduling in reconfigurable computing. In: Proceedings of Design, Automation and Test in Europe (DATE '99). (1999) 90–96
[7] George, M.A., Pink, M., Kearney, D., Wigley, G.: Efficient allocation of fpga area to multiple users in an operating system for reconfigurable computing. In: Proceedings of Engineering of Reconfigurable Systems and Algorithms (ERSA02). (2002) 238–242
[8] Walder, H., Platzner, M.: Online scheduling for block-partitioned reconfigurable devices. In: In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Munich, Germany (2003) 290–295
[9] Dales, M.: Managing a reconfigurable processor in a general purpose workstation environment. In: In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Munich, Germany (2003) 10980–10985
[10] Moscu Panainte, E., Bertels, K., Vassiliadis, S.: Interprocedural compiler optimization for dynamic hardware reconfigurations. In: Proceedings of SAMOS, Samos, Greece, Springer-Verlag Lecture Notes in Computer Science (LNCS) (2005)
[11] Barth, P.: A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Research Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany (1995)