Automated placement of reconfigurable regions for relocatable modules

Tobias Becker, Markus Koester and Wayne Luk
Department of Computing
Imperial College London

***Abstract*— We present an automated method for finding feasible placements of regions on partially reconfigurable Field-Programmable Gate Arrays (FPGAs). Such reconfigurable regions are placed at design time, and can be allocated to different modules at run time. We consider regions that support relocatable modules. A model is introduced that enables masking out non-matching resources for relocatable modules, with an algorithm to find a suitable region placement for such modules. We also consider communication constraints and fragmentation. In a case study involving software-defined radio, we demonstrate that our algorithm increases the number of relocatable regions that can be placed on a device. The average configuration storage size is reduced by a factor of 5.2 when using the proposed relocation approach, which also leads to improvement in compilation time.**

I. Introduction

Run-time reconfigurable FPGAs allow the development of powerful and flexible systems. Computational tasks are implemented as reconfigurable modules and loaded into the FPGA when needed. This technique can increase performance and reduce area requirements and power consumption [1]. However, run-time reconfiguration demands additional design efforts [2]. The reconfigurable modules have to be isolated from the static part of the system. Also at run time, reconfiguration has to be managed and controlled.

Traditional FPGAs provide a regular and homogeneous fabric of identical configurable logic tiles. However, in recent years FPGA architectures became increasingly heterogeneous and now provide a range of hardened functional blocks [3]. The most common type of heterogeneous resources are block RAMs (BRAMs) and DSPs.

In this paper we consider the placement of partially reconfigurable regions (PR regions) on heterogeneous FPGAs where the allocation of reconfigurable modules is limited to these pre-defined regions. The need to pre-define regions is practical and consistent with current design techniques [2]. Placing these regions on the FPGA is a design-time task and regions have to satisfy the requirements of computational tasks that can be implemented in such regions. Later at run time, physical implementations of tasks, so-called reconfigurable modules, can be allocated to these regions. Our placement approach targets heterogeneous architectures and attempts to find regions that allow bitstream relocation such that one reconfigurable module can be allocated to all regions in the device. This approach has the benefits of reducing configuration storage and design time, since with relocation, only one version of each module needs to be produced and stored.

II. Background

Task placement for reconfigurable systems has often been presented as a run-time problem that allows fully flexible 2D-placement on the FPGA [4], [5]. This allows to optimise the placement of a reconfigurable module depending on its size and the available free area. However, such an approach suffers from fragmentation issues when several differently sized tasks are subsequently loaded into the device. Furthermore, research on 2D-placement often neglects the communication infrastructure that would be necessary to connect tasks. Another disadvantage is that there is no fixed number of tasks that can always be guaranteed to execute concurrently. The number of tasks that can operate in parallel depends on the size of these tasks as well as the current fragmentation of the device. Finally, fully-flexible 2D-placement methods are usually based on the assumption of a fully homogeneous device architecture. However, devices have become increasingly heterogeneous in recent times. These techniques are therefore not applicable to modern devices.

As an alternative, the device can be partitioned into several PR regions at design time. At run time, reconfigurable modules can be allocated to free regions. This approach is used in many state-of-the-art reconfigurable systems [6], [7], [8] since it is a practical design technique for current heterogeneous architectures [2]. It avoids fragmentation issues at run time and allows to include a communication infrastructure during the design phase. It also simplifies the problem of run-time allocation. Tasks can simply be assigned to free regions or, if congestions is a concern, priority-based decisions can be made [9].

However, most reconfigurable systems with predefined PR regions employ non-relocatable modules which are specific to one particular target region in the device. Hence, separate versions of each reconfigurable module need to be generated if several PR regions exist. Relocatable modules have the advantage of reducing the configuration storage size since only one bitstream version has to be stored. In a system with m regions, the storage size is reduced by a factor of m. Likewise, it reduces the time needed for physical implementation because only one version has to be mapped, placed and routed. With single configuration bitstreams reaching a size of tens of megabytes [3] and increasingly long place and route times, a reduction in storage size and design time can be an important design improvement.

Module relocation on heterogeneous architectures usually suffers from the fact that identical regions have to be found [10], [11], [12]. With the increasing level of heterogeneity in modern devices it can be difficult or even impossible to find identical regions. A recent method is proposed that simplifies relocation on heterogeneous architectures by identifying a compatible subset of components in the target regions [13]. Regions are not fully identical but non-matching resources are masked out and are only used for routing. At run time, a controller performs simple modifications to the bitstream in order to accommodate the module in different regions. However, the placement of reconfigurable regions is done manually.

The contribution of this paper is automating such a region placement. Unlike previous work which considers fully flexible placement on homogeneous architectures or pre-defined PR regions without relocation, our approach focusses on enabling relocation in practical systems with pre-defined PR-regions on heterogeneous devices.

III. FPGA architecture model

We consider a system where certain tasks are implemented in hardware as reconfigurable modules. A PR region is a dedicated region in the FPGA that can hold one reconfigurable module at a time.

Reconfigurable modules are dynamically allocated to PR regions depending on application requirements. A reconfigurable system usually contains a static part S and several PR regions R_i. The aim of this paper is to find a feasible placement for these PR regions allowing the inclusion of heterogeneous resources and communication constraints. Regions have to be non-overlapping and large enough to accommodate reconfigurable modules.

In the following we describe how we model architecture, application and implementation. The architecture can be a tile-based device such as Virtex-4 or Virtex-5 or a column-based device such as Virtex-2, Virtex-2 Pro or Spartan-3. The application is a reconfigurable system with n tasks, m PR regions and a static system part.

In order to describe resources required by a module or provided by a region, we introduce the n-tuple u. Each tuple component represents the number of a certain type of resource. For the remainder of this paper, we use a 3-tuple that represents the resources CLB, BRAM and DSP: $u = (u_{clb}, u_{bram}, u_{dsp})$. However, the tuple can be easily extended if necessary.

We model the FPGA architecture as a two-dimensional array, where each array element represents an atomic reconfigurable unit (PR unit). For example, in Xilinx Virtex-4 FPGAs, PR units span the height of a clock region and have a size of 16x1 CLBs, 4x1 BRAMs or 8x1 DSPs. Each array element contains a tuple u that represents the available resources in this PR unit. For instance, a CLB unit in Virtex-4 FPGAs would have $u = (16, 0, 0)$. When describing column-based architectures such as Virtex-2, PR units span the height of the entire device and the array therefore contains only one row of elements. The resources of the device are represented by u_{device} which is the sum of all resources in the device. A PR region R_i is a rectangular sub-array of the device and its available resources are represented by the tuple $u_a(R_i)$.

We can model the application as a set of permanent functions or tasks corresponding to the static system part S and a set of n reconfigurable tasks T_1 to T_n. Each task T_i is characterised by its resource requirements $u(T_i)$. The resource requirements of the static part is $u_r(S)$. The reconfigurable system provides m reconfigurable regions R_1 to R_m that hardware implementations of the tasks can be allocated to. The number of regions depends on how many tasks need to run concurrently. The resource requirement $u_r(R_i)$ of a region is defined by the combination of resource requirements of all tasks that can be allocated to this region. More specifically, it is the elementwise maximum of all $u(T_i)$ for all tasks T.

$$u_r(R_i) = \begin{pmatrix} \max(u_{clb}(T_1), \cdots u_{clb}(T_n)) \\ \max(u_{bram}(T_1), \cdots u_{bram}(T_n)) \\ \max(u_{dsp}(T_1), \cdots u_{dsp}(T_n)) \end{pmatrix} \quad (1)$$

A feasible PR region must satisfy this resource requirement, that is $u_a(R_i) \geq u_r(R_i)$. In our case all $u_r(R_i)$ are equal since the same set of tasks can be used in all regions. The resource requirement for the entire system is the sum of the static part and all PR regions: $u_{r,system} = u_r(S) + m \cdot u_r(R)$.

The first step in implementing a system is to choose a target device with $u_{device} \geq u_{r,system}$. However, this is only an initial estimate. The final system size can be larger because regions can cover more resources than initially required.

IV. Placement algorithm

We now want to find a feasible placement for all PR regions. For a placement to be feasible, regions have to be non-overlapping and large enough according to equation 1. PR regions should also allow for relocatable modules, making use of the technique presented in [13] if fully identical regions cannot be found. Relocatable modules

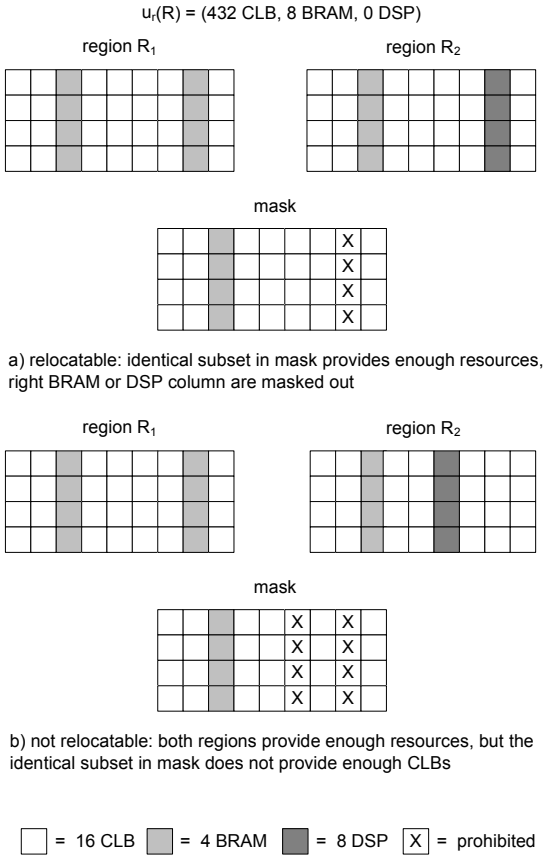

Fig. 1. Examples of relocatable and non-relocatable scenarios for PR regions with a resource requirement of 432 CLBs and 8 BRAMs.

can allow PR regions to have non-matching resources if the identical subset of resources is sufficient to implement the tasks. This is illustrated in figure 1. Non-matching resources can be masked out and are used for routing only. Additionally, we want to allow for communication constraints in order to insure system connectivity.

The key part of the algorithm is illustrated in figure 2. It involves finding one initial region by expanding the right boundary of the region until the resource requirements are met, that is all values in $u_a(R)$ have to be larger or equal to the ones specified in $u_r(R)$. Next, the region is shrunk by moving the left boundary up to the point where further shrinking would violate the resource requirements. This shrinking is done because a region might have to expand far to the right to gain enough of one type of resource. But it might also gain excessive amounts of other resources. These can be reduced by shrinking the left side as indicated in figure 2 a. The region has now minimal size and is locked down as a seed for all other possible regions. A mask that contains the type and relative location of all resources is created. In the following we try to find all possible additional regions that provide a sufficiently large identical subset of resources with the initial region. A sliding window with the same size of the initial region is moved along the device as illustrated in figure 2 b. Using the mask, we check if the underlying fabric provides the required resources in the correct location. If a match is found the region is locked down and added to a region list. If there are non-matching resources in the matching area then the mask is updated with prohibit values as illustrated in figure 1. We then proceed in the same manner to find all possible further regions. The search continues through all rows until the device is full and no more regions can be found. Throughout the search the algorithm keeps track of the resources used by a region list.

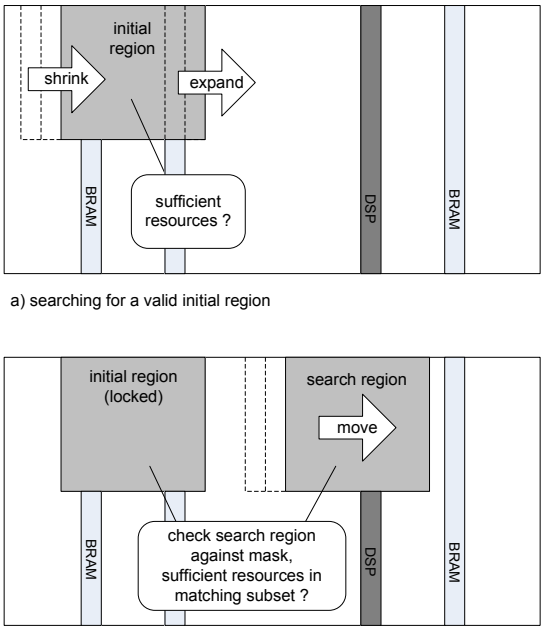

Fig. 2. The key steps of our algorithm involves finding an initial region and searching for further regions with a sufficient number of matching resources.

The algorithm considers a region list as valid if at least m regions can be found. Surplus region will later be eliminated based on additional design constraints such as the location of the static system. If, as an alternative, the number of regions m is not fixed, then the algorithm can be used to maximise the number of PR regions and hence, the number of tasks that can be executed in parallel. Valid region lists are added to a placement list, which represents list competing valid design choices.

The inner part of our algorithm as illustrated by figure 2 is repeated with various parameters in order to populate the placement list with various placement choices. It is possible to perform a full exhaustive search of the design space based on variations in the initial region placement and its aspect ratio. The search space has typically a size of several thundered PR units, with the largest Virtex-4 device containing 1248 PR units. Practically, the search space can be reduced by exploiting symmetries in the device. Furthermore, extreme region aspect ratios can be omitted as well.

The next step is to choose a suitable region list from the placement list generated by the algorithm. We can use three metrics that help selecting a region list: *number of PR regions*, *fragmentation* and *masked-out resources*. If the algorithm is used to find the maximum number of possible PR regions m, then the first criteria is to a select region list with maximum m.

While our method does not introduce any problems with run-time fragmentation, the PR regions can fragment the residual free space of the device. This can cause problems with implementing the static part of the system. We therefore use a method presented by Walder et. al. that calculates the fragmentation of the free space by using maximum free rectangles [14].

Our method will mask out non-matching resources in PR regions in order to allow module relocation. Masked-out resources cannot be used when implementing tasks in a region and hence, increase the size of the PR region. An increased area leads to larger configuration bitstreams and slower reconfiguration time. As a third criteria we therefore choose regions with the smallest percentage of masked-out resources.

We can add support for two different types of communication structures to our algorithm. For communication structures that are embedded into the reconfigurable modules as in [12], PR regions should be lined up and located directly next to each other. In order to support external communication structures as in [2], [15], we extend the region's resource requirement with dedicated routing channels. After finding an initial region and generating the region mask, a previously specified amount of extra rows or columns is added to the mask. These additional PR units are marked as prohibit since they cannot be used by the module. Routing channels are not considered to contribute to device fragmentation, because they are intentionally created. Routing channels can be added to either side of a region, and can constitute of a single row or column or an L-shape.

V. Case study

We perform a case study based on signal processing cores used in software-defined radio. Software-defined radio is an area that can benefit from using reconfiguration [16]. Table I shows a range of signal processing cores and their resource requirements with some parameter variations. All cores are generated with Xilinx CORE Generator and implemented on Xilinx Virtex-4 FPGAs. We cluster these cores into various groups to obtain a range of realistic region resource requirements $u_r(R)$. Based on these requirements, we then use our algorithm to find the maximum number of PR regions on various Virtex-4 devices, with and without module relocation. This is illustrated in table II. In this analysis we use a reconfiguration manager based on a MicroBlaze processor and the internal configuration access port (ICAP). The resource requirement of this static system is $u_r(S) = (515, 33, 3)$. As a communication constraint we require a free routing channel with a height of 16 CLBs below each region.

core	$u(T)$
FIR	(961,0,0), (235,0,16), (474,0,32), (870,0,48)
Direct Down Conversion	(488,16,0), (437,0,0)
Direct Digital Synthesis	(88,1,0), (105,1,2), (133,8,3)
Cordic	(154,0,0), (170,0,0)
FFT	(869,16,0), (200,0,3), (281,0,0), (604,7,16), (1004,17,24)
Forward Error Correction	(301,13,0)
Viterbi Decoder	(495,2,0) ,(272,2,0)
Turbo Decoder	(611,17,0), (315,7,0)

TABLE I
SIGNAL PROCESSING CORES AND THEIR RESOURCE REQUIREMENTS.

Table II shows the number of PR regions can be found for various $u_r(R)$ with and without relocation. Column two shows the resource requirement for the PR regions which is based on various clusterings of cores from table I. As a comparison, column three lists the number of possible regions with relocation when requiring fully identical regions. This scenario represents prior art in relocation. Column four shows the number of possible relocatable regions found by our algorithm that is enabled by the method of masking our non-matching resources. When searching for the maximum number of PR regions, the algorithm only finds a limited number of solutions that often bear resemblance to each other e.g. showing symmetry or minor variations. The optimisation potential for reducing fragmentation of the residual free space or the percentage of masked-out resources is therefore limited. The presented solution are, if possible, optimised

target device u_{device}	$u_r(R)$	$m^{(1)}$	$m^{(2)}$	masked PR units per region	$m^{(3)}$
FX100	(961,16,0)	2	4	10%	6
(10544,376,160)	(611,17,0)	5	6	7.7%	6
	(301,13,0)	6	9	7.7%	9
	(315,7,0)	6	12	8.3%	12
SX55	(1004,17,48)	3	3	0%	3
(6144,320,512)	(604,7,48)	5	5	0%	5
	(235,8,16)	10	10	0%	10
LX40	(495,2,0)	2	2	0%	3
(1608,96,64)	(315,7,0)	3	5	8.3%	5
LX80	(961,16,0)	2	3	5.3%	3
(8960,200,80)	(611,17,0)	2	3	7.7%	3
	(437,0,0)	4	7	23.1%	7
	(301,13,0)	3	5	7.7%	5
	(315,7,0)	5	5	0%	8
	(281,0,0)	5	10	16.7%	10

(1) identical, relocatable regions
(2) non-identical, relocatable regions using masking out of non-matching resources as shown in figure 1
(3) non-relocatable regions

TABLE II
NUMBER OF POSSIBLE PR REGIONS m ON VARIOUS TARGET DEVICES FOR RELOCATABLE AND NON-RELOCATABLE SCENARIOS.

for fragmentation first and secondly for masked-out resources. All designs provide sufficient unfragmented space to implement the static reconfiguration manager. The percentage of resources that have to be masked out is shown in column five. In 10 out of 15 cases, our algorithm finds more regions than it would be possible with identical regions. In the other five cases, fully identical regions can be found which is illustrated by a value of 0 % masked-out PR regions.

The last column lists the number of PR regions that can be found if modules do not have to be relocatable. Comparing columns four and six we see that in 12 out of 15 cases, our algorithm finds as many relocatable regions as without relocation, therefore reducing the configuration storage requirement in these cases without impacting the number of regions that can be found.

The SX55 device is particularly relocation-friendly because all its heterogeneous resources are placed in an identical pattern with equal distances. Hence, no resources have to be masked out for relocation. In both LX devices, relocatability can be hampered because of the uneven distribution of heterogeneous resources. BRAMs and DSPs are placed towards the left and the right side of the device with large CLB islands in the middle. This can lead to high percentages of masked out resources or it can reduce the number of PR regions allowing relocation. We observe that heterogeneity by itself is not an obstacle to relocation, but designing devices with a more regular distribution of heterogeneous resources simplifies the use of relocatable modules.

We implement and compare designs for the 12 cases where an equal amount of relocatable and non-relocatable regions can be found. Designs are built with Xilinx ISE 9.2 tools on a PC with a 3.2 GHz Pentium 4 processor. Comparing the configuration storage sizes when using relocatable and non-relocatable modules, we find that on average the configuration storage requirement per module is reduced from 1110.8 kB to 220.1 kB. This represents a factor of 5.2. The maximum reduction is from 1276.8 kB to 106.4 kb which is a factor of 12. The average compile time is reduced from 2 hours and 39 minutes to 43 minutes when building relocatable modules instead of non-relocatable ones. This represents a 3.7 times speed-up.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we present a model and algorithm to find PR regions for relocatable modules. These regions do not have to be fully identical and non-matching resources can be masked out. We also consider communication constraints and fragmentation of the residual free space for the static system.

Our algorithm often finds more regions than a search for fully identical regions would. In most cases, it finds as many relocatable regions as without relocation. In a case study with signal processing cores we show that PR regions allowing relocation reduce the storage size by a factor of 5.2 on average. We also find that equidistant placement of heterogeneous resources simplifies the search for regions allowing relocation. Devices without regular placement of heterogeneous resources hamper relocation or increase the number of masked out resources.

Current and future work includes adapting and evaluating the benefit of our algorithm for other applications and for other devices, such as Virtex-5 or coarse-grain architectures.

REFERENCES

[1] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson, "A reconfigurable, power-efficient adaptive Viterbi decoder," *IEEE Trans. on VLSI Systems*, vol. 13, no. 4, pp. 484–488, 2005.

[2] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration on Xilinx FPGAs," in *Field Programmable Logic and Applications*. IEEE, 2006, pp. 1–6.

[3] *Virtex-5 Family Platfrom Overview LX and LXT Platforms v2.2*, Xilinx Inc., January 2007.

[4] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Design and Test of Computers*, vol. 17, no. 1, pp. 68–83, 2000.

[5] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1393–1407, 2004.

[6] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght, "Modular dynamic reconfiguration in Virtex FPGAs," *IEE Proceedings Computers and Digital Techniques*, vol. 153, no. 3, pp. 157–164, 2006.

[7] J. Becker, M. Hübner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka, "Dynamic and partial FPGA exploitation," in *Proceedings of the IEEE*, vol. 95. IEEE, 2007, pp. 438–452.

[8] M. Majer, J. Teich, A. Ahmadinia, and C. Bobda, "The Erlangen Slot Machine: A dynamically reconfigurable FPGA-based computer," *Journal of VLSI Signal Processing*, vol. 47, no. 1, pp. 15–31, 2007.

[9] M. Ullmann, M. Hübner, B. Grimm, and J. Becker, "On-demand FPGA run-time system for dynamical reconfiguration with adaptive priorities," in *Field-Programmable Logic and Applications*, ser. Lecture Notes in Computer Science. Springer, 2004, pp. 454–463.

[10] H. Kalte, G. Lee, M. Porrmann, and U. Rückert, "REPLICA: A bitstream manipulation filter for module relocation in partial reconfigurable systems," in *19th International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2005, p. 151b.

[11] N. P. Sedcole, P. Y. K. Cheung, G. A. Constantinides, and W. Luk, "A reconfigurable platform for real-time embedded video image processing," in *Field-Programmable Logic and Applications*, ser. LNCS 2778. Springer, 2003, pp. 606 – 615.

[12] J. Hagemeyer, B. Keltelhoit, M. Koester, and M. Porrmann, "A design methodology for communication infrastructures on partially reconfigurable FPGAs," in *Field Programmable Logic and Applications*. IEEE, 2007, pp. 331–338.

[13] T. Becker, W. Luk, and P. Y. Cheung, "Enhancing relocatability of partial bitstreams for run-time reconfiguration," in *Field-Programmable Custom Computing Machines*. IEEE Computer Society, 2007, pp. 35–44.

[14] H. Walder and M. Platzner, "Non-preemptive multitasking on FPGA: Task placement and footprint transform," in *International Conference on Engineering of Reconfigurable Systems and Algorithms*. CSREA Press, 2002, pp. 24–30.

[15] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. P. Fekete, and J. van der Veen, "DyNoC: A dynamic infrastructure for communication in dynamically reconfigurable devices," in *Field Programmable Logic and Applications*. IEEE, 2005, pp. 153–158.

[16] T. Becker, W. Luk, and P. Y. K. Cheung, "Parametric design for reconfigurable software-defined radio," in *ARC '09: Proceedings of the 5th international workshop on Reconfigurable Computing*. Springer, 2009, pp. 15–26.