A Hybrid Design Space Exploration Approach for a Coarse-Grained Reconfigurable Accelerator

Mehdipour, Farhad
Research Institute for Information Technology, Kyushu University

Noori, Hamid
Institute of Systems and Information Technologies/KYUSHU

Honda, Hiroaki
Research Institute for Information Technology, Kyushu University

Inoue, Koji
Department of Informatics, Kyushu University

他

出版情報：システムLSI設計技術研究発表会. 133, 2008-01-17
バージョン：
権利関係：

A Hybrid Design Space Exploration Approach for a Coarse-Grained Reconfigurable Accelerator

Farhad Mehdipour[†], Hamid Noori[††], Hiroaki Honda[†], Koji Inoue[†††], and Kazuaki Murakami[†, †††]

[†]Research Institute for Information Technology, Kyushu University, Fukuoka 812-8581, Japan

[††]Institute of Systems and Information Technologies/KYUSHU, Fukuoka 814-0001, Japan

[†††]Department of Informatics, Kyushu University, Fukuoka 819-0395, Japan

E-mail: [†]{farhad,dahon}@c.csce.kyushu-u.ac.jp, [††]noori@c.csce.kyushu-u.ac.jp, [†††]{inoue,murakami}@i.kyushu-u.ac.jp

Abstract Multitude parameters involved in the design process of a reconfigurable accelerator which is exploited in embedded systems brings about a remarkable complexity and large design space. One effective technique is design space exploration which is capable to find a right balance between the different design parameters. Quantitative design approach is an alternative which uses the data collected from applications; however it is time consuming and highly depends on designer observations and analyses and might not conclude to an optimal design. In this paper, a hybrid approach is introduced which uses an analytical approach to explore the design space for a reconfigurable accelerator and determine a wise design point based on the quantitative data collected from the targeted applications. It also provides flexibility for applying new design constraints as well as new applications characteristics. Furthermore, this approach is a methodological approach which reduces the design time and results in a design which satisfies the design goals. Experimental results show the efficacy of the hybrid approach.

Keyword Design Space Exploration, Extensible Processor, Data Flow Graph, Coarse Grain Accelerator

1. Introduction

Tight coupling of a reconfigurable unit to a processor core in a System-on-Chip is a popular way for accelerating application execution. This approach is very domain specific that can boost the performance and adapt itself to different characteristics of each application. The design of an extensible processor entails a multitude of design parameters. Variety of design parameters indicate the high complexity of reconfigurable processor design and prove the requirements for a methodological approach. Aside meeting the tight constraints on cost, performance and power, the application specific optimization of embedded systems is inevitable. On the other hand, the flexibility is also important due to accommodating rapid changes in consumer domain. A major challenge in the design procedure is finding the right balance between the different quality requirements that a system has to meet.

Regarding involvement of multitude parameters in the design procedure, the designer needs a means to efficiently explore the design space. A common approach in traditional design techniques is to use detailed simulation for design space exploration. Design space exploration (DSE) is the process of analyzing several "functionally equivalent" implementation alternatives to identify an optimal solution. The design space consists of many alternative design implementations which vary in area, performance and power dissipation. Considering different design specifications too many number of design alternatives for a task might be generated. Therefore, exploring the large design space can become too computationally expensive; then, candidate design points must be obtained by effective design space pruning technique.

Extensible processors offer a high flexibility in identifying custom instructions from the applications and executing them on a tightly coupled reconfigurable accelerator. The reconfigurable accelerator is used as a co-processor to execute the most critical segments or custom instructions generated from embedded applications. Execution of the hot portions of applications brings about enhancement in performance.

Designing an appropriate reconfigurable accelerator satisfying demanded goals in the aspects of speedup, area and energy consumption has been known as a challenging issue. A balanced architecture with smaller area providing the required performance is the most desirable. In [3] a method for the application driven design of a hybrid reconfigurable processors has been presented. In this method, design decisions are based on quantitative data gathered from simulation of realistically-sized workloads on cycle accurate processor models. In [13] several different design parameters are examined ranging from the number of data flow graphs, inputs/outputs supported, the number of addition/subtraction supported to the types of shifts allowed. Evaluation of these designs was done with simulation as well as synthesis to fully evaluate the hardware trade offs in the context of an ARM processor. Karthikeya et al. in [4] introduce a design space exploration flow of reconfigurable architecture including two optimization techniques. Fist one is the reduction in

hardware cost by sharing critical functional resources that occupy large areas in the processing elements and the second one is critical path minimization by pipelining the critical resources.

Clark et al. [2] propose a design methodology based on a quantitative approach. A matrix of functional units is considered as an initial architecture of their accelerator. They perform a set of experiments to determine the width and height of the accelerator. The characteristics of extracted data flow graphs from 29 applications are used to determine the configuration of the accelerator. Moreover, different models of the accelerator are synthesized comprising various height and row configurations (different types of FUs are attempted). Consequently, making decision on basic parameters of the target accelerator is made by analyzing statistics which are obtained from quantitative analysis of the data flow graphs. Noori et al. in [11] utilize the similar quantitative approach for determining the design parameters including the number of FUs, the number of inputs/outputs, height and width of their RFU. Decisions are still made based on their observations and analyses on all data and statistics collected for a set of applications. This approach is based on observations and quantitative analysis obtained from experiments. Therefore, it is not an accurate method and much design time and effort are devoted to acquire a design which might not be a best design point.

Our DSE approach is a hybrid of analytical and quantitative approaches because, in essence, it is based on an analytical approach which utilizes data gathered quantitatively from the target applications. This approach serves two design goals including performance improvement and area reduction. Since, research shows that performance is strongly application-dependent; our approach emphasizes the application aspect. In our approach like ones in [2][11][13], the characteristic of DFGs are also used, but these data are just exploited in computing the analytical expressions presented for accelerator design. In fact, the proposed approach in this paper is a hybrid method for accelerator deign which tries to get benefits of both quantitative and analytical approach to find a wise design point and curtail the time cost and efforts of designer toward a systematic design method. Following are the summary of advantages of our hybrid approach compared to others:

The approach is suitable where the new applications are applied to the reconfigurable processor design. A shorter design procedure can easily apply the effect of the new applications to the design process.

2. Using DSE for Designing RAC

2.1. Design Methodology

A general overview of the stages which are followed to explore design space for an accelerator has been shown in Fig. 1. Firstly, custom instructions (CIs) are pulled out from hot portions of the applications. For each CI its corresponding data flow graph (DFG) is generated (hereafter, in our terminology CI and DFG terms are used correspondingly). In the next stage, basic information required for DSE are obtained quantitatively using the generated CIs. Design space exploration to determine the main architectural specifications of the accelerator is performed. DSE concentrates on choosing the best design analytically aiming the performance enhancement and area reduction. At the final stage, detailed specifications of the designed accelerator are determined and various parameters are tuned for more satisfaction of the design optimization goals. Parameters tuning procedure is similar to one introduced in [11] thus, the main distinction between our approach and one presented in [11] is on using analytical approach to acquire a initial design point more accurately in a shorter time. In addition, it might need a less extra effort to parameter tuning due to obtaining a wise initial design point in ours.

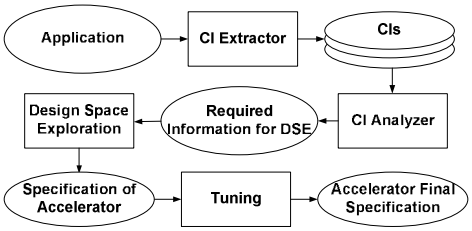


Fig. 1. Overview of the design flow for an accelerator

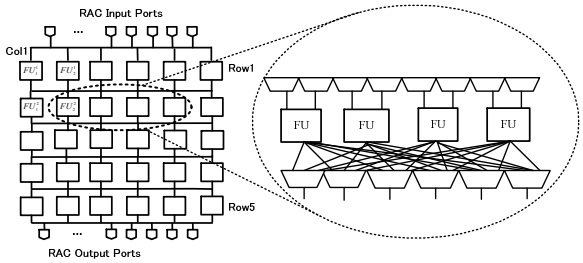


Fig. 2. A piece of RAC architecture [11]

We assume an initial structure for a reconfigurable accelerator (RAC). Following are our assumptions on the initial architecture:

RAC includes a matrix of FUs with width equal to *w* and height equal to *h* and basically has the combinational architecture. A matrix of functional units is a natural way of arranging a RAC, since it allows for both the exploration of parallelism and also for the sequential propagation of data between FUs. DSE determines the suitable dimensions of RAC including *w*, *h* and total number of FUs in RAC.

FUs in RAC are fully connected except than the lack of connections from FUs located in lower rows to FUs in upper rows. All routing resources from each FU in a row to FUs in lower rows and also to adjacent FUs at the same row are available through multiplexers. A piece of connection scheme has been depicted in Fig. 2.

Data flow graphs corresponding to generated CIs should be mapped onto the RAC and executed during

run-time. A simple algorithm is used for mapping the DFGs on RAC. Moreover, routing resources in initial structure of RAC have not been confined and the routing process can be successfully done due to routing resource availability in initial architecture. RAC [7].

2.2. Definitions and Basic Concepts

In following section, our analytical design exploration approach is introduced. First, some definitions:

n_i= no. of FUs in ith row of RAC

MUX_j^i = jth mux between rows i and i+1

$m_j^i = \sum_{k=1}^{i-1} n_k + (n_i - 1)$: the number of inputs for MUX_j^i.

$s_j^i = \left\lceil \log_2^{m_j^i} \right\rceil$: the number of selectors for MUX_j^i

Total number of muxes in row i is equal to the number of FUs in $(i+1)$th row (which is n_{i+1}) multiply by 2 due to existing two input sources for each FU.

$D(FU_j^i)$: latency of FU located in row i and column j.

We assume that all FUs in initial RAC architecture implement similar operations and have the same functionality.

$D(MUX_j^i)$: latency of a mux including m_j^i input bits and s_j^i selector bits. Delays of FUs and muxes can be achieved by synthesizing them or using the pre-synthesized library information.

Here, we present our approach for DSE formulation. It is intended to determine optimal or sub-optimal dimensionality including the RAC's width, height and the number of FUs. Overall speedup obtained from executing CIs on RAC with width of w and height equal to h (RAC_h^w) is the ratio of the number of clock cycles required for executing CI on the base processor to the number of clock cycles spending for execution of CIs on RAC (Eq. 1).

$$Speedup = \frac{TotalClockCyclesOnBase\,\mathrm{Pr}ocessor}{TotalClockCyclesOnRAC} \quad (1)$$

In embedded systems, one issue RISC processor is a conventional choice as the base processor. Therefore, assuming one clock cycle for executing each instruction of DFG on the base processor, the total number of clock cycles required for executing all DFGs on the base processor is easily obtained irrespective of the RAC's specification.

$$CC(CPU) = \sum_{i=1}^{W}\sum_{j=1}^{H} CC_j^i(CPU) \times \alpha_j^i \quad (2)$$

$CC_j^i(CPU)$ is the number of clock cycles required for executing a DFG including width w and height h (DFG_h^w) on a one-issue RISC processor. α_j^i shows the fraction of DFGs (CIs) in applications which have the width equal to w and height equal to h. For example, $\alpha_3^4 = 7\%$ represents that the percentage of application execution time concerns to all DFGs with width equal to 4 and height equal to 3 is 7%. W and H are the maximum width and height for DFGs, respectively. In out terminology, DFG_h^w means a DFG including at most w parallel node and at most h consequent node (maximum depth or critical path of DFG). We assume that the nodes of such DFG can be mapped directly one by one to FUs on RAC, thus it needs a RAC including at most w and h FUs in width and height, respectively.

A key observation is that the number of instructions in DFG_h^w is not exactly equal to $w \times h$. For example, Fig. 3 shows two DFG_3^3 s which include 5 and 6 nodes (instructions), respectively. As our analysis is done based on two parameters w and h, therefore, we define a factor γ_j^i which denotes the average ratio of the number of instructions in DFG to the product of w and h (which means the area or the total number of FUs occupied by DFG) for all DFG_h^w. This factor can be attained through quantitative data gathering from applications.

$$CC_j^i(CPU) = \gamma_j^i \times i \times j \quad (3)$$

On the other hand, total number of clock cycles elapsed for execution DFGs on RAC_h^w is calculated according to following formula. This calculation certainly depends on RAC's dimensions.

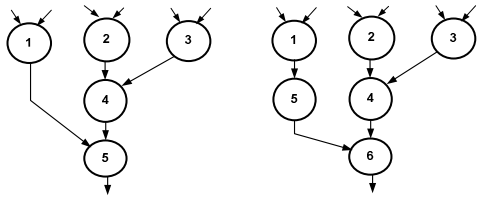


Fig. 3. Two DFG_3^3 s with different number of nodes

$$CC(RAC_h^w) = \sum_{i=1}^{W}\sum_{j=1}^{H} CC_j^i(RAC_h^w) \times \alpha_j^i \quad (4)$$

$CC_j^i(RAC_h^w)$ is the number of CCs for executing DFG_j^i on a RAC_h^w and it is calculated based on the values of w, h, i and j. Four different cases may take place as follows. In cases where one or both dimensions of DFGs are greater that RAC dimensions, we use temporal partitioning techniques. Temporal partitioning divides a DFG into time exclusive smaller DFGs which are mappable on RAC [4][7].

1) $i \le w$ and $j \le h$, in this case, DFG_j^i is mappable on RAC_h^w and no need for temporal partitioning.

$$CC_j^i(RAC_h^w) = D(RAC_h^w) \quad (5)$$

$D(RAC_h^w)$ is the critical path delay of RAC in terms of the cock cycles spent for executing a DFG on RAC_h^w. Delay measured for RAC_h^w is represented based on the number of equivalent clock cycles of the base processor.

2) $i > w$ and $j \le h$,

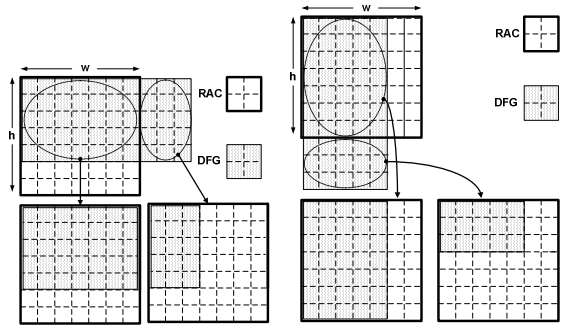
$$CC_j^i(RAC_h^w) = \left\lfloor i/w \right\rfloor \times (D(RAC_h^w) + \lambda) + (1:0) \times D(RAC_h^w) \quad (6)$$

In Fig. 4(a) width of DFG_j^i is greater than RAC_h^w's width. In this case, DFG is partitioned to $\lfloor i/w \rfloor$ smaller DFGs each of them has the width less than or equal to the RAC's width. λ is reconfiguration overhead time due to partitioning DFG to a number of smaller DFGs which should be loaded and executed on RAC subsequently. An extra partition (second term) should be considered and coefficient of the second term is 1 if i is divisible by w, otherwise it would be 0.

3) $i \le w$ and $j > h$,

$$CC_j^i(RAC_h^w) = \left\lfloor j/h \right\rfloor \times (D(RAC_h^w) + \lambda) + (1:0) \times D(RAC_h^w) \quad (7)$$

Fig. 5(b) shows the case where the height of DFG_j^i is greater than RAC_h^w's height.



(a) (b)

Fig. 4. (a) Width of DFG is greater than RAC's width (b) DFG's height is greater than RAC's height

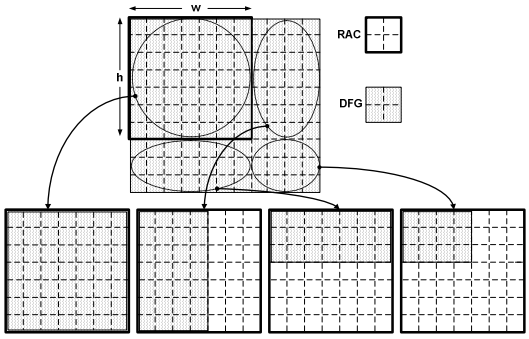


Fig. 5. Both DFG's width and height are greater than RAC's width and height

4) $i > w$ and $j > h$,

$$CC_j^i(RAC_h^w) = \left\lfloor i/w \right\rfloor \times \left\lfloor j/h \right\rfloor \times (D(RAC_h^w) + \lambda) + (1:0) \times i/w \times D(RAC_h^w + \lambda) + (1:0) \times j/h \times D(RAC_h^w + \lambda) + (1:0) \times D(RAC_h^w) \quad (8)$$

In Fig. 5 both dimensions of DFG are larger than RAC's dimensions and a number of smaller partitions are generated utilizing the temporal partitioning algorithm.

2.3. Calculating the Delay of RAC

Measuring $D(RAC_h^w)$ for different values of w and h is an important issue in calculating above expressions. One way for doing that job is synthesizing various sizes of RAC which is too time consuming and maybe not reasonable. We introduce another approach which estimates the latency of critical path for $D(RAC_h^w)$ by analyzing the RAC's structure and does not necessitate synthesizing various RACs' architectures. We use the delay of two basic components including FUs and muxes which have been obtained from their synthesis. Moreover, it should be noted that the RAC realizes a combinational architecture.

As mentioned before, all FUs have the same functionality, so, the delay for all FUs is similar. But, different sizes of muxes are synthesized to achieve their latencies due to need to various sizes of them between different rows of RAC. Therefore, critical path delay of RAC_h^w can be calculated using Eq. 9.

$$D(RAC_h^w) = \sum_{i=1}^{h} D(FU_i^k) + \sum_{i=1}^{h-1} D(MUX_i^k), k \in \{0,1,...,w\} \quad (9)$$

Increasing values of h and w can affect the critical path delay of RAC, due to their influence on the number of FUs and muxes locating in critical path and also the size of the muxes. As mentioned before, each mux located between rows i and $i+1$ receives its inputs from all FUs in upper rows and also from adjacent FUs at the same row. We assume that all muxes including mux(2^n to 1) are available and other mux sizes should be replaced with nearest greater size mux. For example all muxes comprising mux(5 to 1), mux(6 to 1) and mux(7 to 1) are replaced with mux(8 to 1).

$$D(MUX_j^k) = \text{delay of mux}(2^{s_j^k} \text{ to } 1)$$

$$s_j^k = \left\lceil \log_2^{m_j^k} \right\rceil, \quad m_j^k = (j-1) \times w + (w-1) \quad (10)$$

2.4. Optimization problem

The main objective is determining the basic parameters for RAC's architecture including its height, width and the number of FUs. The design space exploration problem here is the finding values of w and h which gives the maximum value of speedup and minimum area for RAC.

$S(RAC_h^w)$ is defined as speedup gained by executing all DFGs extracted from applications on the RAC_h^w (Eq. 11). Therefore, all speedup (and area) values are calculated for $w \in \{0,1,...,W\}$ and $h \in \{0,1,...,H\}$.

$$S(RAC_h^w) = \frac{CC(CPU)}{CC(RAC_h^w)} = \frac{\sum_{i=1}^{W}\sum_{j=1}^{H} CC_j^i(CPU) \times \alpha_j^i}{\sum_{i=1}^{W}\sum_{j=1}^{H} CC_j^i(RAC_h^w) \times \alpha_j^i} \quad (11)$$

The object function is as follows:

Objective: Maximize ($S(RAC_h^w)$) (12)

After finding suitable values for w and h, the number of FUs is calculated using those values and γ (the ratio of the number of nodes/instructions in DFG to the product of w and h).

The number of RAC's FUs= $\gamma_h^w \times w \times h$ (13)

The number of FUs in RAC can also be used for determining the shape of RAC. The initial architecture is a rectangle shape, but according the inherent characteristics of DFGs which are mapped on RAC and have the tri-angular shapes, it can conclude to a non-rectangular shape.

3. Case Study: Designing RAC for an Extensible Processor

In this section, we use the proposed approach to design a RAC for AMBER which is an extensible processor targeted for embedded systems. First, a brief explanation of AMBER and what was going on in [11] to design a RAC for AMBER is presented.

AMBER has been developed by integrating a base processor with two other main components. The base processor is a general RISC processor and the other two components are: sequencer and a coarse grain reconfigurable functional unit (RFU). The *base processor* is a 1-issue in-order RISC processor supporting MIPS instruction set. The *sequencer* mainly determines the microcode execution sequence by selecting between the RFU and the processor functional unit. *RFU* is a RAC including a matrix of functional units (FUs) plus a configuration memory.

3.1. Quantitative Approach

A quantitative approach was used for designing RFU (hereafter the RAC presented in AMBER is referred as RFU), using 22 applications of Mibench [9] and Simplescalar [12] as the simulator tool. RFU was developed in two phases. In the first phase, some primary constraints were considered. DFGs were generated and mapped on RFU considering these constraints. Using the feedbacks of mappings, a proper architecture for RFU was presented. After finalizing the RFU architecture, new constraints of RFU were added to CI generator and mapping tool constraints [11].

First, mapping rate for different number of inputs and outputs was obtained and according to the results, eight and six were chosen as the number of inputs and outputs, respectively. Mapping rate has been defined as the percentage of extracted CIs from 22 applications of Mibench [9] that are successfully mappable and executable on RFU. The mapping rate was similarly measured for various numbers of FUs and the number of FUs was chosen to be 16. Similar procedure is done to specify the width and depth of RFU and also the appropriate number of FUs in each row. Design process finally led to a triangular shape RFU. Experiment results indicate that 6 and 5 are appropriate numbers for width and depth, respectively and 6, 4, 3, 2, 1 FUs for rows 1 to 5, respectively.

3.2. Hybrid Approach

The proposed hybrid approach was also used to design a RFU for AMBER. First, required data including γ and α (Section 3.2) were obtained. Then the analytical approach was followed to determine the design specifications of the RFU. Like the quantitative approach, 22 applications of Mibench were attempted to acquire required information. In our experiments, the base processor is a 1-issue RISC processor and the reconfiguration penalty (λ) is assumed to be one clock cycle which is reasonable in a tightly coupled coarse grain hardware due to small size of the configuration bit-stream and possibility for parallel loading of the RFU configuration.

First the optimal RAC's dimensions for different frequencies ranging from 100MHz to 500MHz are obtained. Table 1 shows the results achieved. According to this table, similar result (width= 6 and height= 5) to quantitative approach is obtained in frequency of 166MHz. This experiment indicates the capability of analytical approach for applying the frequency while this factor could not be considered during the quantitative design procedure. According to experiments, increasing frequency of the base processor reduces RFU dimensions and decreasing its clock frequency has the inverse affect. Because, the number of clock cycles needed for executing DFGs on RFU increases due to reduction in processor clock period and furthermore, enlarging RFU dimensions is less promising in accelerating DFGs execution.

Table 1. RAC's dimensions considering various frequencies and targeting speedup

Frequency(MHz)	**500**	**333**	**250**	**200**	**166**	**100**
Width	5	5	5	5	6	7
Row	2	3	3	4	5	5

Object function (Eq. 12) is computed for different dimensions of RFU in the frequency of 166MHz. Fig. 6 shows how the object function affected by various RFU's width and heights. The peak of graph is where *width= 5* and *height=4*. In this point, maximum speedup and smallest area are gained. For the resulted dimensions using analytical approach, the number of FUs is also calculated according to Eq. 13 which is equal to 15. This indicates that the RFU's shape should be triangular.

As another feature of the analytical approach, the effect of reconfiguration penalty (overhead time) was examined. For large DFGs which have to be partitioned and mapped subsequently on the RFU, reconfiguration penalty should be taken into account. Reconfiguration overhead time may affect the performance of DFG execution on RFU because, loading subsequent DFGs

takes one or more clock cycles. Since RFU is coarse grain hardware with tight integration to the base processor and reconfiguration memory (which is used for storing DFGs' configuration bit-streams) therefore, its reconfiguration overhead time may take one or more clock cycles depending on the bit-stream size and topology of connection between RFU and configuration memory. Table 2 shows the effect of reconfiguration penalty on RFU's dimensions in frequency of 166MHz. By increasing reconfiguration overhead time the size and height of RFU is increased to cover larger DFGs and reduce the number of partitions hence, preventing the speedup amortization.

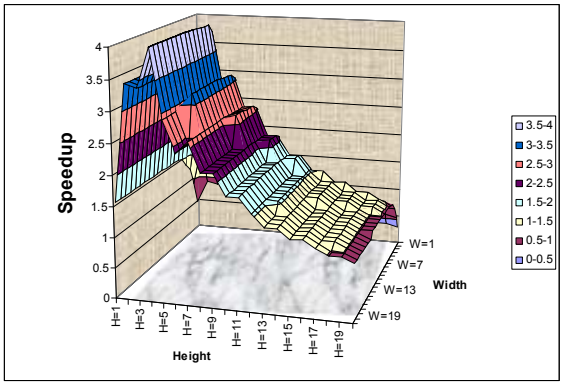


Fig. 6. Object function is maximized in $w=5$ and $h=4$

Table 2. RAC's dimensions considering various reconfiguration penalties

Reconfiguration Penalty (clock cycles)	1	2-6	7-9	10-15
Width	5	5	6	7
Row	2	4	6	8

4. Conclusion

Several parameters involving in the design process of a reconfigurable accelerator integrating to a processor in an extensible processor result in a large design space. Design space exploration using analytical method in combination with a quantitative analysis is one way to find a right balance between the various design parameters. This hybrid approach unlike the quantitative approach targets the design goals directly and gives a wise starting design point based on quantitative data collected from applications. This approach is susceptible for applying new design parameters as well as new applications and therefore, it substantially reduces the design time and effort. This approach was used for obtaining dimensions of an accelerator which is exploited in an extensible processor called AMBER. Various frequencies were examined and a number of constraints and parameters like area and reconfiguration penalty were applied to DSE procedure. Experiment results show that the analytical approach can be used to obtain a similar design point to the quantitative approach in a noticeably shorter time and less design effort.

Acknowledgment

This research was supported in part by Core Research for Evolutional Science and Technology (CREST) of Japan Science and Technology Corporation (JST) and Grant-in-Aid for Encouragement of Young Scientists (A), 17680005.

References

[1] J. Becker, M. Vorbach, "Architecture, memory and interface technology integration of an industrial/academic configurable system-on-chip (CSoC)," ISVLSI 2003, pp. 107-112, 2003.

[2] N. Clark, M. Kudlur, H. Park, S. Mahlke and K. Flautner, "Application-specific processing on a general-purpose core via transparent instruction set customization," The 37th Annual IEEE/ACM Intl. Symp. on Microarchitecture, pp. 30-40, 2004.

[3] R. Enzler, M. Platzner, "Application-driven design of dynamically reconfigurable processors," http://e-collection.ethbib.ethz.ch/browse/sg/092_e.html, 2001.

[4] M. Karthikeya, P. Gajjala, B. Dinesh, "Temporal partitioning and scheduling data flow graphs for reconfigurable computer," IEEE Transactions on Computers, vol. 48, no. 6, pp.579-590, 1999.

[5] Y. Kim, M. Kiemb, K. Choi, "Efficient design space exploration for domain-specific optimization of coarse-grained reconfigurable architecture," SoC Design Conference, pp.12-17, 2005.

[6] M. H. Lee, H. Singh, G. Lu, N. Bagherzadeh, and F. J. Kurdahi, "Design and implementation of the MorphoSys reconfigurable computing processor," Journal of VLSI Signal Processing Systems, vol. 24, no. 2-3, pp. 147-164, 2000.

[7] F. Mehdipour, H. Noori, M. Saheb Zamani, K. Murakami, M. Sedighi, K. Inoue, "Custom instruction generation using temporal partitioning techniques for a reconfigurable functional unit," Int. Conference on Embedded and Ubiquitous Computing, pp. 722-731, 2006.

[8] B. Mei, S. Vernalde, D. Verkest, R. Lauwereins, "Design methodology for a tightly coupled VLIW/reconfigurable matrix architecture: A Case Study", Design, Automation and Test in Europe, vol. 2, pp. 21-24, 2004.

[9] Mibench, www.eecs.umich.edu/mibench.

[10] S. Mohanty, V.K. Prasanna, S. Neema, J. Davis, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation", LCTES'02-SCOPES'02, pp. 18-27, 2002.

[11] H. Noori, F. Mehdipour, K. Murakami, K. Inoue, M. Saheb Zamani, "An Architecture Framework for an Adaptive Extensible Processor," Journal of Supercomputing, in press.

[12] Simplescalar, www.simplescalar.com

[13] S. Yehia et al., "Exploring the design space of LUT-based transparent accelerators," International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, pp. 11-21, 2005.