Design Space Exploration for a Coarse Grain Accelerator

Farhad Mehdipour[1], Hamid Noori[2], Morteza Saheb Zamani[3], Koji Inoue[2], Kazuaki Murakami[2]

[1] Research Institute for Information Technology, Computing and Communication Center,
Kyushu University, Fukuoka, Japan, e-mail: farhad@c.csce.kyushu-u.ac.jp
[2] Department of Informatics, Graduate School of Information Science and Electrical Engineering,
Kyushu University, Fukuoka, Japan
e-mail: noori@c.csce.kyushu-u.ac.jp, {inoue, murakami}@i.kyushu-u.ac.jp
[3] Department of IT and Computer Engineering, Amirkabir University of Technology,
Tehran, Iran, e-mail: szamani@aut.ac.ir

Abstract - In the design process of a reconfigurable accelerator employing in an embedded system, multitude parameters may result in remarkable complexity and a large design space. Design space exploration as an alternative to the quantitative approach can be employed to find a right balance between the different design parameters. In this paper, a hybrid approach is introduced to analytically explore the design space for a coarse grain accelerator and determine a wise design point exploiting data extracted from applications, quantitatively. It also provides flexibility for taking into account new design constraints as well as new characteristics of applications. Furthermore, this approach is a methodological approach which reduces the design time and results in a point which satisfies the design goals.

I. Introduction

Nowadays, tight coupling of a reconfigurable unit to a processor core in a System-on-Chip is a popular way for accelerating application execution. The design of such a reconfigurable 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. A major challenge in the design procedure is finding the right balance between the different quality requirements that a system has to meet. One approach is design space exploration (DSE) which 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. For example, a design with four tasks on an architecture with three processing units and each have four possible configurations results in 500 design alternatives (can be computed using Stirling's number [9]) which is a large design space even in case of simulation. Therefore, exploring the large design space can become too computationally expensive; then, candidate design points must be obtained by effective design space pruning techniques.

The main contribution of this paper is to explore the design space for a reconfigurable accelerator which is augmenting to a base processor in a reconfigurable processor. 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. In the design procedure of a reconfigurable accelerator many trade-offs should be considered in which the speedup and area are two important factors. Therefore, a balanced architecture with smaller area providing the required speedup is the most desirable. We outline an analytical approach for the design space exploration utilizing data obtained quantitatively from target applications serving two design goals including speedup and area reduction.

II. Related Work

A coarse-grain reconfigurable hardware can provide higher performance than software implementation and more flexibility than custom hardware implementation. Many coarse-grain reconfigurable architectures have been suggested [1][2][5][6][7][10]. Among them, two dimensional mesh architectures have the advantage of rich communication resources for parallelism. Morphosys [5] consists 8x8 array of reconfigurable cell coupled with TINY_RISC processor. It shows good performance for regular code segment in computation intensive domains but requires high hardware cost. XPP configurable system-on-chip architecture [1] is another example. XPP has 4x4 or 8x8 reconfigurable array and LEON processor with AMBA bus architecture.

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 the quantitative data gathered from simulation of realistically-sized workloads on cycle accurate processor models. In [11] several different design parameters are examined ranging from the number of data flow graphs, inputs/outputs, the number of addition/subtraction to the types of shifts allowed. Evaluation of these designs is done with simulation as well as synthesis to fully evaluate the hardware trade offs in the context of an ARM processor. In [4] a design space exploration flow of reconfigurable architecture including two optimization techniques is introduced. 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

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

quantitative approach. A matrix of functional units (FUs) 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 22 applications are used to determine the configuration of the accelerator. Moreover, different models of the accelerator are synthesized comprising various width and height 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 the quantitative analysis of the data flow graphs. In [10], authors use the similar quantitative approach for determining the design parameters. Decisions are still made based on their observations and analyses on all data and statistics collected for a set of applications. Approaches proposed in [2] and [10] uses mapping rate as a factor representing speedup which has been defined as the percentage of DFGs that are successfully mappable and executable on the accelerator. Therefore, these approaches are based on observations and quantitative analysis obtained from experiments and they target the design goals implicitly. Moreover, much design time and effort are devoted to acquire a design which is not necessarily a good design point.

Since, research shows that performance is strongly application-dependent; our approach emphasizes on the application aspect. Our proposed approach is hybrid because it is an analytical approach on the statistics data gathered from analyzing of DFGs to find a reasonable design point. In this approach like ones in [2] and [10], the characteristic of DFGs are also used, but these data are just exploited in computing the analytical expressions presented for accelerator design. In fact, this hybrid approach 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. Here is the summary of advantages of our analytical approach compared to others:

1. This approach uses realistic data collected for attempted applications in demanded domains to accomplish a hybrid analytical and quantitative approach.

2. The proposed approach reduces design time and designer efforts substantially by giving a design point which is obtained considering the speedup and area parameters.

3. It is promising to extend this approach such that additional design parameters could be applied.

4. The approach is suitable for cases 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 accelerator design. Items 3 and 4 reflect the flexibility of the proposed method.

III. RAC Design Using DSE Approach

A. Overall Design Approach

Fig. 1 shows a general overview of the stages which are followed to explore design space for an accelerator. At the first stage, 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).

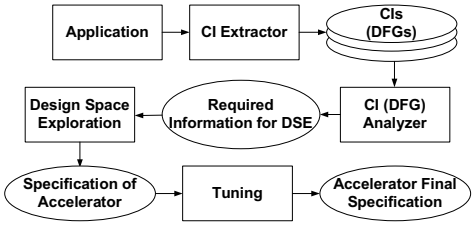


Fig. 1. overview of procedure for accelerator design

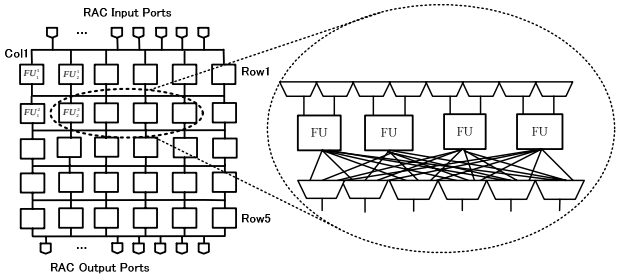


Fig. 2. A piece of RAC architecture

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 [10], thus, the main distinction between our approach and one presented in [10] is on using analytical approach to acquire a initial design point more accurately and in a shorter time.

We consider an initial structure for a reconfigurable accelerator (RAC). FUs implementing instruction level operations and muxes for establishing connections between FUs are basic elements of RAC. The following are our assumptions on the initial architecture:

1. RAC includes a matrix of FUs with the width equal to *w* and the height equal to *h*. A matrix of FUs 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 [2].

2. 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 (except the horizontal connections) has been depicted in Fig. 2.

3. Data flow graphs corresponding to the generated CIs should be mapped onto the RAC and executed during run-time. An algorithm from [6] is utilized 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 the initial architecture. After fixing final architecture and pruning redundant FUs and connections, a designer may replace a dedicated mapping algorithm.

B. Problem Definition and Basic Concepts

The problem is finding suitable dimensions comprising width, height and the number of FUs for a coarse grain RAC which is tightly integrated to a base processor. The inputs are the DFGs corresponding to critical segments of applications. The main goal is designing a minimum size RAC to get maximum enhancement for demanded applications. Here, our analytical design exploration approach is introduced. First, we present some definitions:

n_i= no. of FUs in *i*th row of RAC

MUX_j^i = *j*th 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 (n_{i+1}) multiplied 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 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.

Our formulation is based on speedup and area factors. Overall speedup obtained from executing DFGs 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 DFG on the base processor to the number of clock cycles spending for execution of DFGs on RAC.

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

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 in applications which have the width equal to *w* and height equal to *h*. For example, $\alpha_3^4 = 7\%$ indicates that the percentage of application execution time pertaining 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 our 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 [6], thus it needs a RAC including at most *w* and *h* FUs in width and height, respectively.

A key observation we have made 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 (atomic instructions), respectively. As our analysis is done based on two parameters *w* and *h*, therefore, we define a factor (γ_h^w) 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 s. 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, the total number of clock cycles elapsed for executing DFGs on RAC_h^w is calculated according to the following formula. This calculation certainly depends on RAC's dimensions.

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

$CC_j^i(RAC_h^w)$ is the number of clock cycles 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 are as follows. In cases where one or both dimensions of DFGs are greater than RAC dimensions, we use temporal partitioning techniques for partitioning DFGs to smaller and mappable ones. Temporal partitioning divides a DFG into time exclusive smaller DFGs that can not or do not need to run concurrently [6].

1) $i \le w$ and $j \le h$, in this case, DFG_j^i is mappable on RAC_h^w. $CC_j^i(RAC_h^w) = D(RAC_h^w)$ (5)

$D(RAC_h^w)$ is the critical path delay of RAC in terms of the clock cycles spent for executing a DFG on RAC_h^w .

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

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

Fig. 4.a shows the case where the width of DFG_j^i is greater than the RAC's width, 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 the reconfiguration overhead time due to partitioning DFG to a number of smaller DFGs which should be loaded and executed on RAC subsequently. Also, the coefficient of the second term is 1 if *i* is divisible by *w*, otherwise it would be 0.

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

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

Fig. 4.b shows the case where the height of DFG_j^i is greater than RAC_h^w 's height.

4) $i \succ w$ and $j \succ h$,

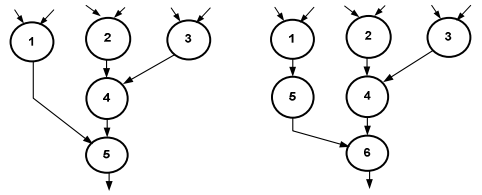
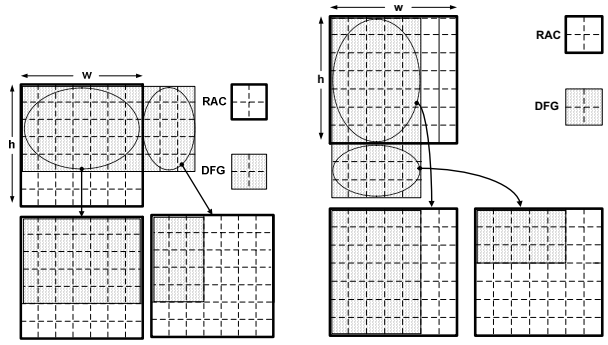


Fig. 3. Two DFG_3^3 s including five and six nodes



(a) (b)

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

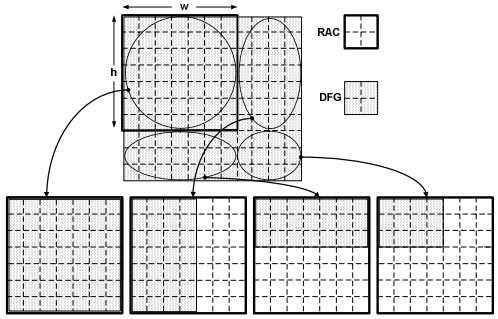


Fig. 5. DFG's width/height are greater than RAC's width/height

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

Fig. 5 shows the case where the both dimensions of a DFG are larger than RAC's dimensions.

C. 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 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 of RAC. We use the delay of two basic components including FUs and muxes which have been obtained from their synthesis.

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 including mux(5 to 1), mux(6 to 1) and mux(7 to 1) are replaced with mux(8 to 1).

$$D(MUX_j^k) = \text{delay of mux}(2^{s_j^k} \text{ to} 1)$$
$$s_j^k = \left\lceil \log_2^{m_j^k} \right\rceil, \quad m_j^k = (j-1) \times w + (w-1) \quad (10)$$

D. Area of RAC

Area is the second parameter which impacts the RAC design. Area of the RAC depends on the area of FUs and also the area of muxes that are used for establishing the connections between FUs. Total area of RAC can be estimated by the following expression:

$$A(RAC_h^w) = \sum_{i=1}^{w}\sum_{j=1}^{h} A(FU_j^i) + 2 * \sum_{i=1}^{w}\sum_{j=1}^{h-1} A(MUX_j^i) \quad (11)$$

$A(FU_j^i)$ is the area of FU_j^i which is identical for all FUs due to their similarity. $A(MUX_j^i)$ is the area of MUX_j^i which varies because of the different number of inputs and selectors. $A(MUX_j^i)$ is calculated according to following expression.

$$A(MUX_j^i) = \text{Area of mux}(2^{s_j^i} \text{ to} 1)$$
$$s_j^i = \left\lceil \log_2^{m_j^i} \right\rceil, \quad m_j^i = (j-1) \times w + (w-1) \quad (12)$$

E. 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 give the maximum value of speedup and minimum area for RAC.

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

$$S(RAC_h^w) = \frac{CC(CPU)}{CC(RAC_h^w)} = \frac{\sum_{i=1}^{W}\sum_{j=1}^{H} CC_j^i(CPU) \times \alpha_j^i}{\sum_{i=1}^{W}\sum_{j=1}^{H} CC_j^i(RAC_h^w) \times \alpha_j^i} \quad (13)$$

The object function is as follows:

Objective: Maximize ($S(RAC_h^w)$) (14)

For the designs which give the similar speedup the smaller

design is chosen. If speedup for a design is greater than the other one (second design) and the ratio of speedups is more than the r_1(>1) then the first design is preferred if the ratio of the areas for two designs are less than r_2 (e.g. r_2=1.25 means that the first design's area is 1.25 area of the second design).

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

$$\text{The number of RAC's FUs} = \gamma_h^w \times w \times h \quad (15)$$

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 triangular or trapezium shape.

IV. 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 [10] 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). Fig. 6 depicts the integration of different components in AMBER. 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 matrix of functional units (FUs) plus a configuration memory. Each FU implements fixed-point instructions of the base processor except *multiply, divide* and *load*.

A. Quantitative Approach

A quantitative approach was used for designing AMBER's RFU (RAC) using 22 applications of Mibench [8]. In the RFU's initial architecture there is no limitation on the number of inputs, outputs (I/O), FUs and RFU's width and height. DFGs extracted from applications are mapped on the initial architecture of RFU. Using the feedbacks of mapping, a proper architecture for RFU was presented. The design procedure in [10] is mainly based on obtaining mapping rate (Section II) according to each design parameter and determining a suitable parameter value which gives the reasonable mapping rate. Mapping rate implicitly represents the achievable speedup. More mapping rate, more portions of applications are executable on RFU to obtain higher speedup. Design procedure has led to 16, 6 and 5 as the number of FUs, width and height of RFU, respectively. Moreover, 6, 4, 3, 2, 1 FUs have been chosen for rows 1 to 5 which represents the triangular shape for RFU (more details can be found in [10]).

B. Analytical Approach

The proposed approach was used to design a RFU for AMBER. First, required data including γ and α (Section III.*B*) were obtained. Like the quantitative approach in [10], 22 applications of Mibench were attempted to acquire the required information. Also, delay and area of basic elements comprising FUs and muxes are achieved by synthesizing them using Hitachi 0.18µm. Then the analytical approach was followed to determine the design specifications of RFU. In our experiments, the base processor is a 1-issue RISC processor and reconfiguration penalty (λ) is assumed to be one clock cycle which is reasonable in a tightly coupled coarse grain hardware. First the optimal RFU's dimensions for different frequencies ranging from 100MHz to 500MHz are obtained. TABLE I shows the results achieved. According to this table, similar result (*width= 6* and *height= 5*) to the 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 quantitative design procedure. Moreover, increasing frequency of the base processor reduces RFU dimensions and decreasing its clock frequency has inverse affect. Because, the number of clock cycles needed for executing DFGs on RFU increases due to reduction in processor clock period therefore, enlarging RFU dimensions is not still promising in accelerating DFGs execution.

We also applied the parameter of area to determine the dimensions of RFU. TABLE II shows the results targeting both speedup and area. A larger design with more speedup is preferred if the ratio of its area to the design with lower speedup is not more than a threshold value (r_2 in Section III.*E*). In our experiments, r_1 and r_2 were empirically set to be 1.1 and 1.2, respectively. A similar speedup is achieved in frequency of 166MHz by choosing a smaller design (*width=5* and *height= 4*). Object function (Eq. 14) is computed for different dimensions of RFU in the frequency of 166MHz. Fig. 7 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 *width=5*, maximum speedup and smallest area are attained. For the resulted dimensions using analytical approach, the number of FUs is also calculated according to Eq. 15 which is equal to 15. This indicates the RFU's shape should be triangular or trapeze.

To show another feature of the analytical approach, effect of reconfiguration penalty (overhead time) in the design procedure 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 because, loading subsequent DFGs' configurations for execution on RFU takes one or more clock cycles.

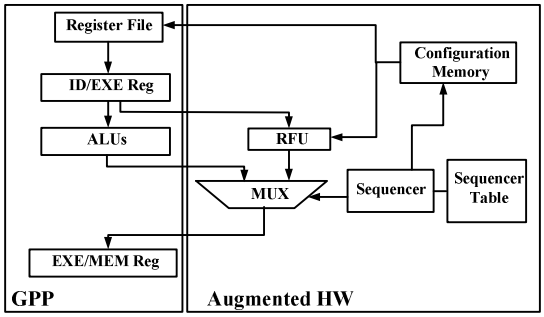


Fig. 6. Integration of main components in AMBER

Since RFU is coarse grain hardware with tight integration to the base processor and reconfiguration memory (is used as storage for 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 connections between RFU and configuration memory. TABLE III shows the effect of reconfiguration penalty on RFU's dimensions in the 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 degradation. Finally, we compared our hybrid approach by three other DSE approaches. TABLE IV reflects the features of ours.

V. Summary and Conclusions

Multitude parameters involving in the design process of a reconfigurable accelerator (RAC) integrating to a processor result in a large design space. Design space exploration using analytical method is one way to find a right balance between the various design parameters.

TABLE I. RAC (RFU)'s dimensions considering various frequencies and targeting speedup

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

TABLE II. RFU's dimensions considering various frequencies and targeting speedup and area

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

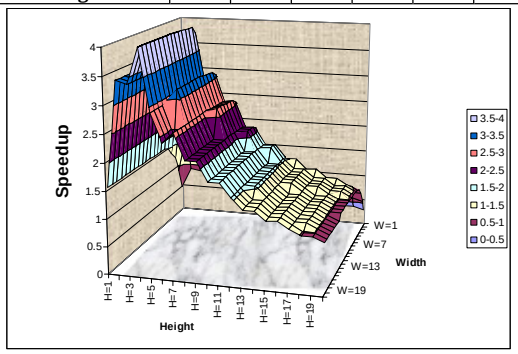


Fig. 7. Object function is maximum in *width=5, height= 4*

TABLE III. RFU's dimensions considering various reconfiguration penalties

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

TABLE IV. Comparing ours with different design approaches

Design Approaches	Design Method	Design Time &Effort	Design Basic Parameters	Flexibility
Clark et al. [2]	Quantitative & Synthesis	High	Mapping rate	Low
Yehia et al. [11]	Synthesis & Simulation	Very High	No. of operations, inp/outputs	Low
Noori et al. [10]	Quantitative	Median	Mapping rate	Low
Ours	Hybrid	Low	Speedup	High

Our approach unlike the quantitative approach targets the design goals explicitly and gives a wise starting design point based on the quantitative data collected from applications. Calculating delay and area of RAC without need to synthesizing and simulation has the basic role in shortening the design time. This approach is susceptible for applying new design parameters as well as new applications and substantially reduces the design time and effort. This approach was used for obtaining dimensions of an accelerator employing in an extensible processor called AMBER. Various frequencies were examined and constraints and parameters like area and reconfiguration penalty were applied to design space exploration procedure. Experiment results show that the analytical approach can be used to obtain a similar design point to quantitative approach in a noticeable shorter time.

Acknowledgment

This research has been 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 International Symposium on Microarchitecture, 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] Y. Kim, M. Kiemb, K. Choi, "Efficient design space exploration for domain-specific optimization of coarse-grained reconfigurable architecture," SoC Design Conference, 2005.

[5] 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 and Signal Processing Systems for Signal, Image and Video Technology, 2000.

[6] 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, 2006.

[7] 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 (DATE04), 2004.

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

[9] 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, 2002.

[10] H. Noori, F. Mehdipour, K. Murakami, K. Inoue, M. Saheb Zamani, "An architecture framework for an adaptive extensible processor," Journal of Supercomputing, in press.

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