Performance Enhancement of an Adaptive Dynamic Extensible Processor by Using a Heterogeneous Reconfigurable Functional Unit

Mehdizadeh, Arash
Computer Engineering Department, Amirkabir University of Technology

Ghavami, Behnam
Computer Engineering Department, Amirkabir University of Technology

Zamani, Morteza Saheb
Computer Engineering Department, Amirkabir University of Technology

Mehdipour, Farhad
Computing and Communication Center, Kyushu University

出版情報：International SoC Design Conference, pp.251-254, 2007-10
バージョン：
権利関係：

Performance Enhancement of an Adaptive Dynamic Extensible Processor by Using a Heterogeneous Reconfigurable Functional Unit

Arash Mehdizadeh Behnam Ghavami Morteza Saheb Zamani
Computer Engineering Department
Amirkabir University of Technology
Tehran, Iran
{a_mehdizadeh, ghavami, szamani}@ aut.ac.ir

Farhad Mehdipour
Computing and Communication Center
Kyushu University
Fukuoka, Japan
farhad@c.csce.kyushu-u.ac.jp

ABSTRACT - *In this paper, we develop a heterogeneous architecture for the reconfigurable functional unit of an extensible processor. To verify the efficiency of our architecture, we applied it to 8 applications of Mibench. The new architecture improves execution time of custom instructions by 20% to 30% on average while supporting more custom instructions. The area and the total wire length are reduced by 15% and 20% respectively. In addition, depending on the custom instructions being run on the unit, average dynamic power consumption is reduced by 9% making this unit more suitable for embedded applications.*

Keywords : Custom Instruction, Extensible Processor, Reconfigurable Functional Unit.

1 Introduction

One recent approach of embedded systems implementation, and in fact a way to fill the gap between general purpose processors (GPP) and application specific integrated circuits (ASIC), is the use of custom hardware in special applications. In such way, even instructions could be customized. Application-specific instruction-set processors (ASIPs) have been an important design and implementation methodology for system-on-chip processors in the last decade. As a complement to the approach of ASIPs, processors with extensible instruction sets have been introduced in which a core collaborates with a reconfigurable functional unit being implemented either coarsely or finely [2][3][5]. Even after the implementation of the instruction set architecture of such systems, custom instructions (CIs) can be added to the system. These instructions are extracted regarding hot basic blocks (HBB). A basic block is a sequence of instructions that is ended with a control instruction. HBBs are referred to the basic blocks which are repeated more than a threshold number of times during the execution of a certain program. With such definition, critical sections of programs are extracted as data flow graphs (DFG), mapped and executed on a hardware accelerator or a functional unit (FU) bound to the main core. In this paper, A new tightly coupled [4] fast-interconnected reconfigurable functional unit (RFU) is presented for the previously introduced Adaptive dynaMic extensiBlE processoR (AMBER) [1]. Enhancing AMBER's functionality, we reduced critical path delay of the RFU by replacing collections of individual identical FUs with some other non-identical ones. The Rest of the paper is organized as follows: In Section 2 the first proposed architecture of AMBER's RFU is presented. In section 3 the new heterogeneous architecture is introduced. Sections 4 and 5 discuss mechanics of DFG clustering and configuration memory. In sections 6 experimental results are given. Finally, the paper is concluded in section 7.

2 Related Work

In [1][7] an extensible processor named AMBER is introduced which utilizes a tightly coupled coarse grain RFU. In AMBER, there is no need for a new programming model, compiler, opcode for new instructions, source code modification or recompilation. The user just runs the applications on the base processor then generation of custom instructions and handling their execution are done transparently and automatically. The main concern in [1][7] was to cover as much CIs as possible or in other words has a coverage percentile as close to as 100%. We further enhanced this structure by introducing a new heterogeneous architecture which reduces critical path delay and configuration bits while increasing CI coverage with no penalty in area or total wire length. AMBER is an extensible processor consisting of a 4-issue in-order RISC processor supporting MIPS instruction set, profiler, RFU, and a scheduler. Portions of applications suitable for acceleration can be executed on a reconfigurable core in AMBER that is the RFU. The RFU is a matrix of FUs. As it has been also mentioned in [8], according to the processor's size of data, a matrix of FUs seems an efficient and reasonable hardware for accelerating sub-dataflow graphs as CIs.

2.1 RFU's Homogeneous Architecture

Based on the quantitative approach in [1] regarding maximum possible mapping rate, the RFU architecture in Figure 1 was proposed. In this architecture, there are

16 identical FUs which implement a single function per configuration based on their configuration bits. Interconnection of these 16 units is established through multiplexers programmed by configuration bits other than those associated with the configuration of FUs and those storing immediate values.

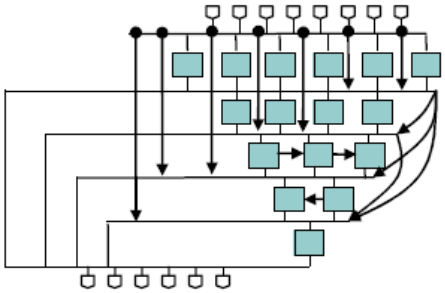

Figure 1. Architecture of RFU using identical FUs

3 The New Heterogeneous Architecture

One of the most important aspects of RFU construction which did not receive much of concern in similar works is the length of critical path and its effect on performance. Critical path is referred to the path incorporating maximum number of active sequential FUs from one input to one output. Frequent occurrences of CIs inherently call for reduction of CIs execution time on the RFU. This time directly depends on the critical path delay and therefore, its reduction leads to the reduction of critical path delay. Experimental results show that the delay associated with the multiplexers used for interconnection of FUs constitutes a great portion of the overall delay (Table 1). Intuitively, reducing the number of multiplexers affiliated with the structure of a mapped CI on the RFU can reduce the critical path delay dramatically.

Table 1. Critical path delays of components of RFU in TSMC 0.18u technology optimized for speed.

Unit	Critical path delay (ns)	Unit	Critical path delay (ns)
Mux 3-1	1.16	Mux 7-1	1.7
Mux 4-1	1.24	Mux 8-1	1.86
Mux 5-1	1.47	FU	6.94
Mux 6-1	1.52		

To reduce the number of multiplexers in the critical path without affecting the mapping rate, we propose an RFU with non-identical FUs. FUs of this new architecture are able to execute multiple instructions (one, two or three) with every regular consecution in a DFG structure (based on the FU's structure). We define a regular DFG as the one in which the output of every node is not connected to more than one node. Analyses over 20 applications of Mibench [6] show that almost 97% of DFGs have this attribute. Considering these regular DFG structures, we propose FUs which can implement every consecutive two and three-instruction sets representing sub-data flow graphs (sub-DFGs) depicted in Figure 2. We named these FUs as bi- and tri-instruction FUs respectively. By replacing a number of previous uni-FUs and multiplexers of RFU with bi/tri-FUs, a considerable reduction in the critical path delays of mapped CIs can be resulted as shown in Table 2.

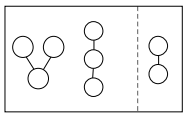

Figure 2. Regular executable sub-DFGs on tri and bi-instruction FUs (left to right)

Table 2. Replacing a number of uni-FUs and multiplexers with bi/tri-FUs (synthesized in TSMC 0.18u technology optimized for speed)

Units	Critical path delay (ns)	**Replaced Unit**	Critical path delay (ns)
2 uni-FU and 1 Mux	14.47	**bi-FU**	10.7
3 uni-FU and 2 Mux	23.8	**tri-FU**	15.9

To determine the structure of RFU, first we must find proper values for the number of RFU inputs and outputs. Therefore, the mapping rate of the generated CIs were inspected in 20 applications of Mibench on the RFU. Then, our generated CIs were mapped on the RFU without considering any constraints. By examining the mapping rate for different numbers of inputs and outputs, we tried to choose proper numbers. According to the results, eight and six are good candidates for the number of inputs and outputs, respectively.

We also conducted an analysis to determine the number of uni-, bi- and tri-instruction FUs. According to Figure 3, putting constraints on the number of inputs and outputs, 85% of CIs contain less than 9 nodes.

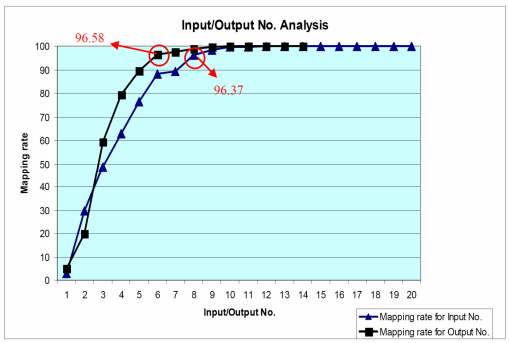

Figure 3. Mapping rate for different numbers of I/O

Therefore, to have a more reliable analysis, we observed precisely all the regular DFGs being composed of less than 9 nodes with different topologies (multiplicities of these DFGs are shown in Table 3). Moreover, for the analysis of CIs with more than 8 instructions, mapping rate frequency was used by which we mean the percentage of generated CIs for the applications of Mibench that can be mapped on the RFU.

Table 3. Possible number of regular DFGs with different number of nodes

Num of DFG nodes	Num of regular DFGs	Num of DFG nodes	Num of regular DFGs
1	1	5	6
2	1	6	11
3	2	7	23
4	3	8	46

Many different architectures and configurations considering the mapping rate results were examined. Based on the conducted analyses, the RFU architecture depicted in Figure 4 (a) is introduced. In this architecture, when an input data is needed by the FUs located in rows other than the first row or when the output of one row is used by the FUs placed in a non-subsequent row, move instructions are mapped on the intermediate FUs to pass over the data. Assuming these limitations, the mapping rate decreases to 84.72%. To improve the mapping rate, many different architectures and configurations were examined. Finally, we reached to the architecture illustrated in Figure 4 (b).

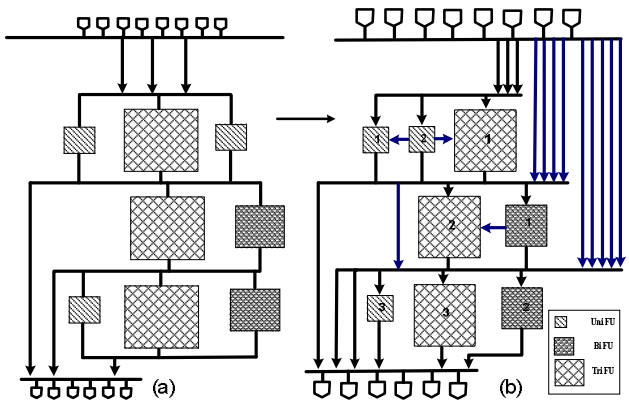

Figure 4. The proposed fast interconnected heterogeneous RFU architectures

To facilitate data accesses for the FUs and reduce the inserted move instructions, beside the connections that exist from outputs of each row to the inputs of the subsequent row, ten other longer connections were added as shown in Figure 4(b). In rows 1 and 2 unidirectional connections to the neighbor FUs were added. These three connections support those long CIs that do not have much parallelism but their operations are very dependent. Using these connections, the CIs shorter than 13 nodes can be supported by the RFU. This RFU is able to map all CIs comprising less than 9 nodes and 78% of CIs 9 to 16 nodes long. In this way, the mapping rate becomes 95.31%. Experiments show that each FU of the RFU does not need to support all of the microprocessor supported instructions. We defined three types of instructions: logical (type 1), add/sub/compare (type 2) and shift operations (type 3). Distribution and multiplicity of each type for each row are given in Table 4.

Table 4. Type of functions for each FU

Row Number	uni-FU	uni or bi-FU	tri-FU
1	Type 1,2	Type 3	Type 1,2,3
2	-	Type 2	Type 1,2,3
3	Type 1,2	Type 1,2,3	Type1,2

Enforcing all of the constraints, the mapping rate becomes 94.86% which is almost 6% better than the previous architecture [7]. In addition, in the previous RFU (Figure 1), DFGs longer than 8 nodes are not mappable whilst it is improved to 12 nodes long in our proposed architecture. Each CI configuration needs 287 bits for storing control signals and 201 bits for immediate values.

4 DFG Clustering

Due to the different FUs and their varying ability to implement certain sub-DFGs, DFG clustering requires certain considerations to be mappable on the RFU. For optimum utilization, we used a greedy algorithm to cluster CIs, It clusters the biggest branch of the graph and then establishes a new sub-graph. The same step is repeated on the sub-graph based on the remaining resources in the RFU. Figure 5 illustrates an example of DFG clustering and its corresponding CI mapping on the RFU. In order to obtain better results, we applied a new mechanism called "Structure Profiling" to map CIs less than 9 nodes long. In our survey, all regular structures of DFGs with less than 9 nodes were identified and then traversed bottom up. So, for all of these DFGs we generated a configuration profile stored in the configuration memory. During CI mapping, every new DFG is traversed and then compared with the previously profiled ones. If one of the stored profiles is identical to the corresponding DFG, it will be used to map the CI on the RFU. Every configuration profile is stored in the mapping tool.

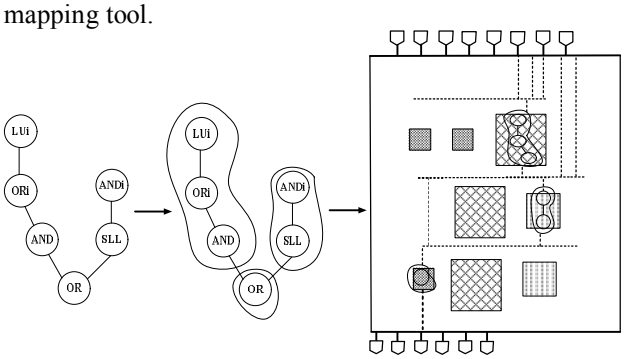

Figure 5. DFG clustering and CI mapping on the RFU

5 Configuration Memory

As computed in Section 3, each CI configuration needs 488 bits- this is already better than the 512 bits of [1]. As a result, in an application such as rijndael with 117 CIs, we need about 6.96 KBs for the configuration data. However, experiments show that similar CIs

provide good opportunities to reduce the configuration memory by merging their configuration data to one. The problem is that in most cases, the configuration bits which are related to functions and connections are the same but the control bits for inputs, outputs or immediate values differ. In order to reduce the size of configuration memory, CI configuration data is divided to four parts. In other words, RFU is provided with partial reconfiguration; 138 bits for configuring intermediate connections and selecting functions of FUs (P1), 90 bits for selecting inputs (P2), 60 bits for outputs (P3) and 196 bits for immediate values (P4). We added another stage to our tool flow in which generated CIs are received as inputs and similar CIs and their subsets are searched and then their configuration data are merged into one. The two CIs are similar if their P1 are the same and a CI is a subset of another CI if its P1 is a subset of the bigger one. Although we generate one P1 for a CI and its subsets, as their P2, P3 and P4 are different they generate the desired results. For inputs (P2), outputs (P3) and immediate values (P4), we just look for equal configuration bits and generate one configuration data for them. In the next step, to make the configuration memory smaller, we try to merge P1 of small CIs into one configuration. Using these two techniques and using less intermediate multiplexers we were able to reduce the configuration memory. The other advantage of using these two techniques is that the number of context switchings is decreased due to the less number of configurations.

6 Experimental Results

Average execution times of CIs derived from 8 applications of Mibench on both homogeneous and heterogeneous architectures are given in Figure 6. As it is shown, owing the fact that the critical path is reduced in our newly proposed architecture, CI execution times are 20-30% shorter than the previous architecture. Average dynamic power consumption of both architectures was measured during several simulation runs and as it is shown in Figure 7 the new architecture consumes less energy. All these are beside the reduction in the area and total wire length which proved to be reduced by 15% and 20% respectively, according to the reports of the logic and layout synthesis tools.

7 Conclusion

Exploiting non-identical FUs in a heterogeneous RFU can improve the runtime and mapping rate of CIs compared to their homogeneous counterparts. In this paper, a heterogeneous architecture was proposed for the RFU of an extensible processor named AMBER, based on a quantitative and analytical approach. The mapping rate of CIs on this architecture was 94.86% and the CI execution speed on our structure was improved drastically compared to the previous architecture.

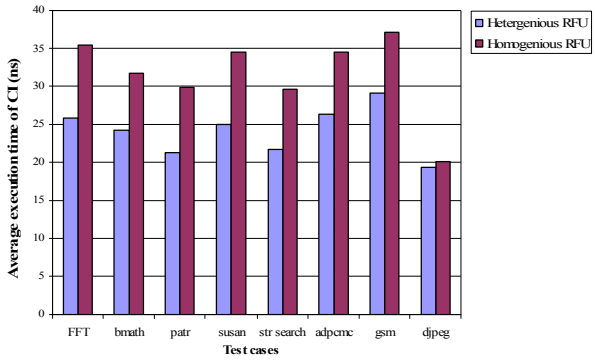

Figure 6. Comparing average execution time of CIs on previous and new architecture

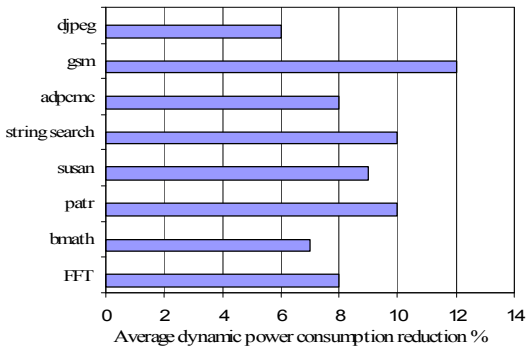

Figure 7. Average dynamic power reduction in the new architecture. The reduction is dependant on the structure and frequency of CI's in different applications.

Reference

[1] H. Noori, F. Mehdipour, K. Murakami, K. Inoue and M. Saheb Zamani, "A reconfigurable functional unit for an adaptive dynamic extensible processor," IEEE International Conference on Field Programmable Logic and Applications (FPL'06), Spain, 2006.

[2] S. Hauck, T. Fry, M. Hosler, and J. Kao, "The Chimaera reconfigurable functional unit," in Proc. IEEE Symp. FPGAs for Custom Computing Machines, pp. 87–96, 1997.

[3] A. Lodi, M. Toma, F. Campi, A. Cappelli, R. Canegallo, and R. Guerrieri , "A VLIW processor with reconfigurable instruction set for embedded applications," IEEE Journal of Solid-State Circuits, vol. 38, no. 11, pp. 1876–1886, 2003.

[4] B. Mei, S. Vernalde, D. Verkest, and R. Lauwereinsg, "Design methodolody for a tightly coupled VLIW/reconfigurable matrix architecture: A case study," in Proc. Design, Automation and Test in Europe, 2004.

[5] S. Vassiliadis, and et al., "The MOLEN polymorphic processor," IEEE Transactions on Computers, vol. 53, no. 11, Nov. 2004, pp. 1363–1375.

[6] www.eecs.umich.edu/mibench

[7] H. Noori, K. Murakami, and K. Inoue, "A General Overview of an Adaptive Dynamic Extensible Procesor", in Proc. Workshop on Introspective Architecture, 2006.

[8] N. Clark, M. Kudlur, H. Park, S. Mahlke and K. Flautner, "Application-Specific Processing on a General-Purpose Core via Transparent Instruction Set Customization", in Proc. MICRO-37, 2004.