An efficient Heterogeneous Reconfigurable Functional Unit for an Adaptive Dynamic Extensible Processor

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

Pedram, Hossein
Computer Engineering Department, Amirkabir University of Technology

他

出版情報：VLSI-SoC 2007, 2007-10-16
バージョン：
権利関係：

An efficient Heterogeneous Reconfigurable Functional Unit for an Adaptive Dynamic Extensible Processor

Arash Mehdizadeh Behnam Ghavami Morteza Saheb Zamani Hossein Pedram Farhad Mehdipour*

{ a_mehdizadeh, ghavami, szamani, pedram}@ aut.ac.ir, *farhad@c.csce.kyushu-u.ac.jp

Computer Engineering Department, Amirkabir University of Technology (Tehran Polytechnic)

424 Hafez Ave, Tehran 15785, Iran

*Computing and Communication Center, Kyushu University

3-8-33-309 Momochihama, Sawara-ku, Fukuoka 814-0001, Japan

ABSTRACT

Replacing functional units of an extensible processor with reconfigurable functional units enhances performance and flexibility of processors to execute custom instructions. That is due to the ability of reconfigurable functional units to perform computations in hardware to increase performance, while retaining much of the flexibility of a software solution. 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. Our experiments show that compared to the similar architectures, ours supports a wide range of custom instructions. In addition, use of the new architecture improves execution time of custom instructions by 20% to 30% on average. Moreover, compared with the previous architecture, area is reduced by 15%.

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

1. Introduction

Embedded systems, having proven their abilities in a wide range of applications, are extensively used in communications and consumer products. Regarding prevalence of these systems, different methods, such as general purpose processors (GPPs) and application specific integrated circuits (ASIC), have been adapted to implement them. Although the use of GPPs, as a usual approach of implementing embedded systems results in high flexibility, because of their inefficiency in performance and power consumption, they are not widely applicable. As a result, ASICs have been proposed. Deep submicron issues of interconnect delay and signal integrity have significantly increased design costs of ASICs both due to the higher engineering costs resulting from longer design cycles and increasing cost of design tools [1]. Another recent approach of embedded systems implementation, and in fact a way to fill the gap between GPP and ASIC era, 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. Compared to GPPs, ASIPs have more potential to meet high-performance demands of embedded applications. However, synthesis of ASIPs traditionally involved the generation of a complete instruction set architecture for the target application. On the other hand, GPPs are very flexible but may not offer the necessary performance. Hence, as a complement to the approach of ASIPs, processors with extensible instruction sets have been introduced. The important motivation toward specialization of existing processors versus the design of complete ASIPs is to avoid the complexity of a complete processor and toolset development. In these systems, a core collaborates with a reconfigurable functional unit (which can be implemented either coarsely or finely). Even after the design and 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 which 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 fast-interconnected reconfigurable functional unit (RFU) is presented for the previously introduced Adaptive dynaMic extensiBlE processoR (AMBER) [2]. 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 paper is organized as follows: In Section 2 a background of systems with reconfigurable functional units will be given. In Section 3 and Section 4 AMBER processor, which is used as a basis for our implementations, is introduced continued with a proposed structure for the RFU in Section 5. The mechanics of DFG clustering and their mapping on the RFU are presented in Section 6. In Section 7, binding of the RFU to the main processor is discussed. The issue of configuration memory is discussed in Section 8. In Section 9, experimental results are reported and finally, the paper is concluded with some proposals for the future works.

2. Related Work

Recently design and implementation of extensible processors FUs has been much of concern in numerous papers. Programmable accelerators augmenting to a base processor fall in two categories based on the granularity of their structure, fine grain and coarse grain. Fine grain accelerators are suitable for very flexible computations. However, long latency and slow reconfiguration time are two of the most important drawbacks associated with these systems. They also need a large amount of memory for storing the configuration bits. To compensate the computational inefficiency and configuration latency most of them deal with very large sub-graphs. Some of the fine grained hardware accelerators are introduced in [3][12].

Chimaera [4], OneChip [5] and XiRisc [6] are some instances of fine grain programmable hardware integrated with GPPs. ADRES [7] is a counterpart of the formers with a coarse grain structure.

The number of inputs/outputs and integration method of accelerator and base processor differ for each design. For example, PRISC uses an RFU with two inputs and one output, while RFU of Chimaera has nine inputs and one output.

Accelerators are divided into two general categories as loosely coupled and tightly coupled. A loosely coupled accelerator plays the role of a co-processor which helps balancing of the load on the main processor and itself. Use of these accelerators calls for exclusive compilers and refinement of portions of the opcode [10][11]. In loosely coupled systems like MorphoSys [8] and Garp [9] there is an overhead for transferring data between the base processor and the coprocessor. In contrast, use of tightly coupled accelerators does not require any overhead in information transfer. Further more, there is no need to worry about an individual compiler or refinement of opcode.

In [2][3] 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 [2][3] 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 architecture is introduced in the following section.

3. Overview of AMBER

AMBER is an extensible processor which can be utilized in different applications of embedded systems. It consists of a microprocessor, profiler, RFU, and a scheduler. The base processor is a 4-issue in-order RISC processor that supports MIPS instruction set. Figure 1 demonstrates AMBER components that will be elaborated in the following sections.

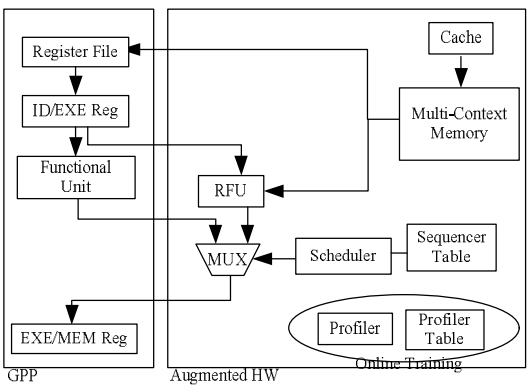

Figure 1. Components of AMBER

3.1 Profiler

AMBER has two modes of operation: Training and Normal. In the training mode, applications are profiled to extract HBBs. Then the object code is used to extract the configuration bits. Training can be done either dynamically or statically. In the former case, there is a sheer need for extra hardware to perform profiling (profiler). In addition, all elaboration functions such as HBB recognition and CI generation are done on the main processor. On the other hand, in the latter case, a host computer simulates and profiles the programs prior to their execution on AMBER. The host computer works independently from AMBER. However, dynamic profiling can be done during intermittent idle periods of the main processor. In this case, generation of CIs does not interfere with the main tasks of the processor.

Profiler contains the following components: 1) two registers one for the previous program counter (PC) and the other for current PC, 2) a comparator to compare values of the two registers; and 3) a table to store the start addresses of HBBs and their execution frequency. In every clock cycle, the profiler compares values of the two aforementioned registers. If the difference of these two values is not equal to the instruction length, a taken branch or jump has occurred. The profiler has a table with a counter for each entry that keeps the execution frequency of basic blocks. In the case of a taken branch/jump, the profiler's table is checked. If the target address (the current PC) is in the table, the corresponding counter is incremented, otherwise current PC is added as a new entry and its counter is initialized to one. Using the profiler's table and a predefined threshold value, the start addresses of HBBs are detected according to their frequency of occurrence [3].

3.2 Reconfigurable Functional Unit (RFU)

Portions of applications suitable for acceleration are the ones which are executed frequently. These portions 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 [15], 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. Exploiting such core can increase the execution speed dramatically [2]. Hence, promisingly, increasing the number of CIs (mappable on the RFU) can decrease the application runtime.

3.3 Scheduler

Scheduler is responsible to decide whether operations to be executed on the microprocessor or on the RFU. This unit contains a table in which the starting address of each CI (in the reconfigurable memory unit) along with the required clock cycles for execution of the instruction is stored. This table is given the values based on the starting address of each CI in the object code. During the execution, as soon as the scheduler observes the PC equal to one of the entries in this table, FU of the microprocessor halts and the RFU takes the responsibility to execute the CI. In this situation, the scheduler waits for completion of the CI, and then it sets value of the PC according to the length of the recently executed CI.

4. An Architecture for the RFU: A Quantitative Approach

The quantitative flow in Figure 2 was applied to 20 applications of Mibench [13] to identify suitable CIs. To do it, Simplescalar [14] is used as the simulation tool and it is modified to keep track of taken branches and jumps. The trace file is employed as input by the profiler to detect beginning of HBBs [3]. Then a DFG is generated for each HBB and passed to the CI generator tool. The CI generator makes CIs. Mapping tool receives CIs and maps them on the RFU. Results of the mapping tool lead to the RFU architecture. To reduce

implementation overhead and increase efficiency, two primary constraints are considered for CIs: a) supporting only fixed-point instructions excluding multiply, divide and load, b) including at most one store and at most one control instruction in a CI. Multiply and divide were excluded due to their low execution frequency and large area occupancy.

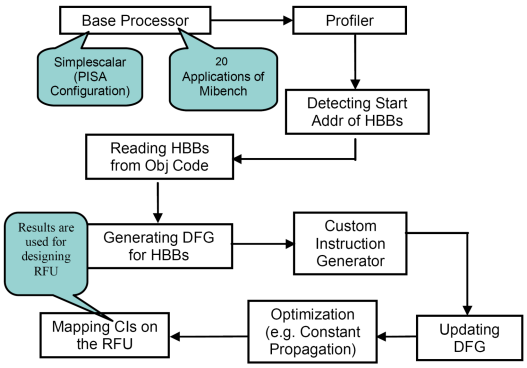

Figure 2. Quantitative approach flow

In order to justify the use of heterogeneous functional units, in this section an overview of the previous work in [3] is presented firstly then our proposed architecture is introduced.

4.1 Homogeneous Architecture of RFU

To determine the proper number of inputs and outputs, first it is assumed that all DFGs corresponding to the extracted CIs can be mapped on the architecture. Based on the analyses of CIs in 20 test applications of Mibench, concerning higher mapping rate as well as less consumption of resources, proper numbers of inputs and outputs were proved to be 8 and 6 respectively [3] (Figure 3). This means that having an RFU with 8 inputs and 6 outputs, nearly 100% of CIs with 8 inputs and 6 outputs are mappable on such structure (mapping rate is almost equal to 100%).

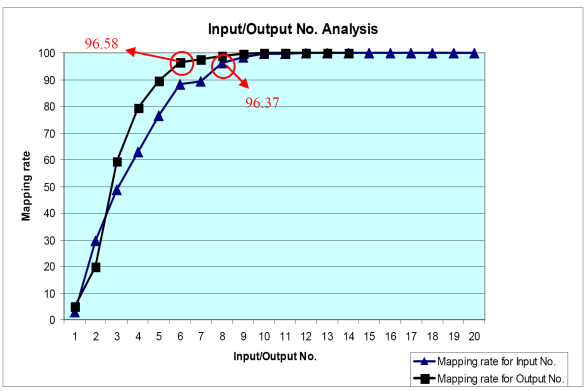

Figure 3. Mapping rate for different numbers of I/O

In the next step there should be an appraisal on the total number of Functional Units (FUs) in the RFU. This is to acquire a high mapping rate using as few FUs as possible. Based on the observations (Figure 4), provided that the limitation put on the number of inputs and outputs is considered, mapping rate curve levels out around the number 16. Hence the minimum proper number of FUs is 16.

Number of inputs, outputs and FUs being determined, to preserve the high mapping rate it is assumed that all the FUs are identical and each is able to implement an individual function per configuration. Based on this assumption other analyses were performed to determine the dept and number of FUs for every row.

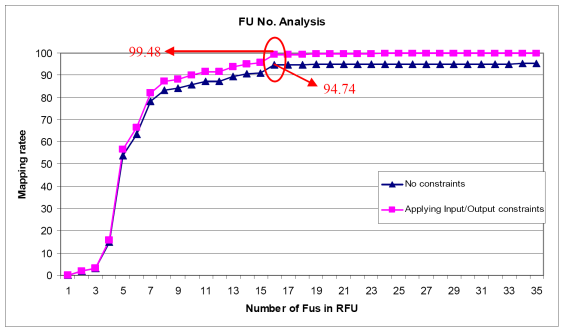

Figure 4. Mapping rate for different numbers of FUs

Aforementioned observations resulted in the architecture depicted in Figure 5. 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 the ones associated with the configuration of FUs.

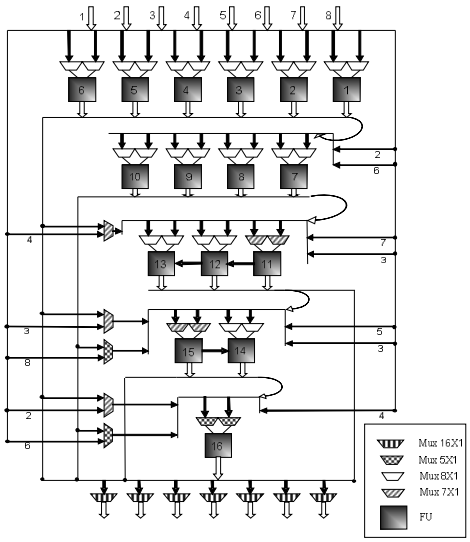

Figure 5. Proposed architecture of RFU using identical FUs

5. New Heterogeneous Architecture of RFU

One of the most important aspects of RFU construction which did not receive much of concern in the 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 occurrence of CIs inherently calls for reduction of CIs execution time. Consider a mapped CI which requires two clock cycles to be accomplished on the RFU. CI execution time is equal to the integer number of clock cycles multiplied by the length of a cycle. This time directly depends on the critical path delay. If the specified CI is to be repeated for 2000 times in the course of the whole program execution, reducing its run time to one clock cycle will make the whole program execution time 2000 cycles shorter which is a considerable gain in comparison to the former case. Shortening CIs execution time calls for reduction of critical path delay.

Based on these premises, experiments have been conducted on the relativity of delay of different components of RFU to the overall RFU critical path delay. Results show that delay associated with the multiplexers being used for interconnection of FUs constitutes a grate portion of the overall delay. This is

shown in Table 1 (RFU's circuit is synthesized with TSMC 0.18u technology). Intuitively, reducing number of multiplexers affiliated with the structure of a mapped CI on the RFU will reduce the critical path dramatically.

Table 1. Critical path delays of different component of RFU in TSMC 0.18u technology.

Unit	delay of critical path (ns)	Unit	delay of critical path (ns)
mux3-1	1.16	mux7-1	1.7
mux 4-1	1.24	mux8-1	1.86
mux5-1	1.47	FU	6.94
mux6-1	1.52		

Based on this assumption, in order to reduce the number of multiplexers in the critical path without affecting the mapping rate, we proposed an RFU comprising non-identical FUs. This means, 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 output of every node is not connected to more than one node. Analyses over 20 applications of Mibench show that almost 97% of DFGs have this attribute. Considering these regular DFGs structures, we proposed FUs which can implement every consecutive two and three-instruction sets representing sub-data flow graphs (sub-DFG) depicted in Figure 6. We referred these FUs as bi- and tri-instruction FUs respectively.

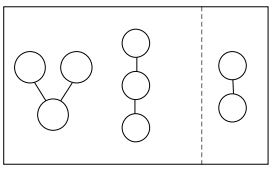

Figure 6. Regular executable sub-DFGs on tri and bi-instruction FUs (left to right)

By replacing a number of previous uni-FUs and multiplexers of RFU with bi/tri-FUs a considerable reduction in critical path delay of mapped CIs can be resulted as is shown in Table 2.

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

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 indicate the proper numbers for RFU inputs and outputs. We inspected mapping rate of generated CIs of 20 applications of Mibench on the RFU. Then, we mapped our generated CIs 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 depicted in Figure 3, eight and six are good candidates for input and output numbers, 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. Hence, to have a more reliable analysis, we observed precisely all the regular DFGs consisting 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 we used mapping rate frequency by which we mean the percentage of generated CIs for 20 applications of Mibench that can be mapped on the RFU.

Table 3. Possible numbers of regular DFGs with different numbers 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 Figure7 is introduced.

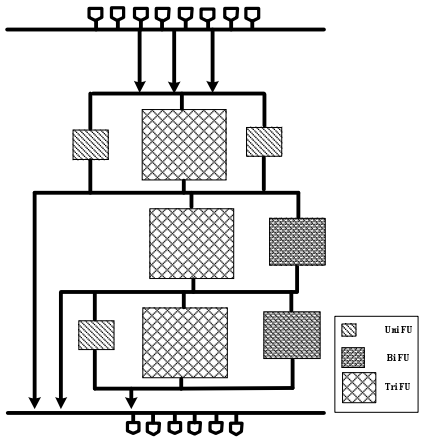

Figure 7. The first proposed heterogeneous architecture of RFU

In this architecture when an input data is needed by FUs located in rows other than first row or when the output of one row is used by FUs placed in a non-subsequent row, move instruction 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 considering the mapping rate results were examined. We reached to the architecture illustrated in Figure 8.

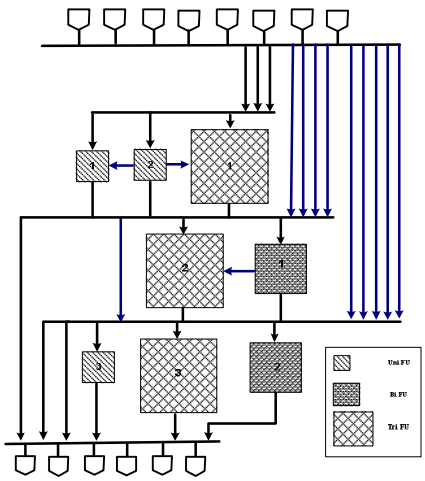

Figure 8. The proposed fast interconnected heterogeneous architecture of RFU

To facilitate data accesses for FUs and reduce the inserted move instructions (which occupy FUs), besides the connections that exist from outputs of each row to the inputs of subsequent row, ten other longer connections were added. One longer connection is: from outputs of row 1 to inputs of row 3 and other connections are: from main inputs of RFU to the inputs of row 2 and 3. 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 CIs with lengths less than 13 nodes can be supported by the RFU.

This RFU is able to map all CIs consisting of less than 9 nodes and 78% of CIs 9 to 16 nodes long. As a result the mapping rate becomes 95.31%. Experiments show that each FU of the RFU does not need to support all of 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 constraints, the mapping rate becomes 94.86% which is almost 6% better than the previous architecture [2]. In addition regarding the homogeneous architecture of the RFU proposed in [2] (Figure 5), DFGs longer than 8 nodes are not mappable. This limitation is improved to DFGs 12 nodes long in our proposed architecture. Each CI configuration needs 287 bits for storing control signals and 201 bits for immediate values. Therefore, configuration of each CI on the RFU requires 488 bits totally compared to 512 bits for the RFU proposed in [3].

6. DFG Clustering

Regarding difference of FUs and their varying ability to implement certain sub-DFGs, clustering of DFGs requires certain considerations to be mappable on the RFU. For optimum utilization we used a greedy algorithm to cluster CIs, thus first the biggest branch of the graph is clustered, then a new sub-graph is established. The same step will be repeated on the sub-graph based on the remaining resources in the RFU.

Optimum clustering is the most of concern in this algorithm. As efficiency of clustering is affected by the limit on the number of resources, architecture of the RFU is designed based on the mostly repeated patterns in DFGs. In other words, for establishment of interconnections, the structures which are more frequent are noticed. As a result our structure will inherently be optimum for our way of clustering. On the other hand, for less frequent structures, clustering will be affected by the limitations of interconnection network which will deviate the result from optimality. Figure 9 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 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 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.

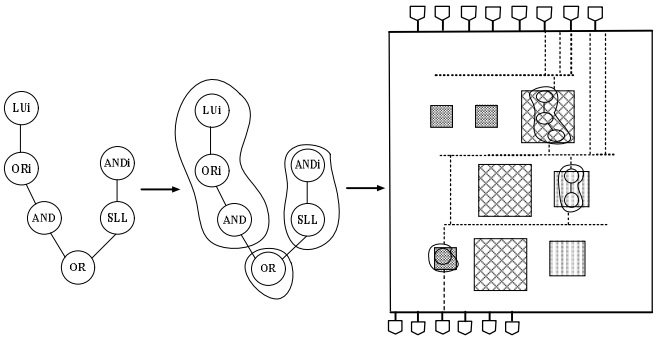

Figure 9. An example of DFG clustering and CI mapping on the RFU

7. RFU and Main Processor Connection

The core processor of AMBER is a 4-issue in-order RISC microprocessor. Connection of this processor to the RFU is shown in Figure 10.

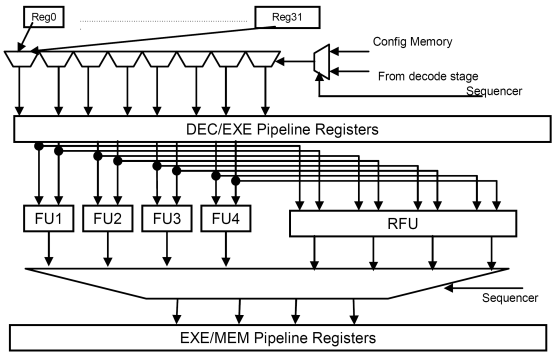

Figure 10. Integrating the RFU and the core processor

As it is shown, in/output ports of the microprocessor and the RFU are common. This way of connection eliminates the need for extra lines to separate intputs/outputs while eliminating parallelism of these two units as well. As a large portion of application code is executed on the RFU, inherently there is no opportunity for parallelism which in turn reduces the need for accommodating exclusive in/output lines. As stated before, the RFU has 6 outputs; hence, two registers are added to the RFU structure which will store extra outputs (more than 4). Contents of these registers are written to the register file in the next cycle.

8. Configuration Memory

As computed in Section 6, each CI configuration needs 488 bits. As a result for an application such as rijndael with 117 CIs we need around 6.96 KBs to keep 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 we receive generated CIs as inputs and look for similar CIs and their subsets and then merge their configuration data into one. 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 will 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 switching will be decreased due to fewer configurations.

9. Experimental Results

Average execution times of CIs derived from 8 applications of Mibench on both homogeneous and heterogeneous architectures are given in Figure 11. 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. Moreover as in the new architecture the number of multiplexers is reduced, configuration bits of multiplexers are reduces which in turn results in less memory and power consumption. All these improvements are besides the reduction in the area which proved to be reduced by 15%, according to the reports of MAGMA's layout synthesis tool.

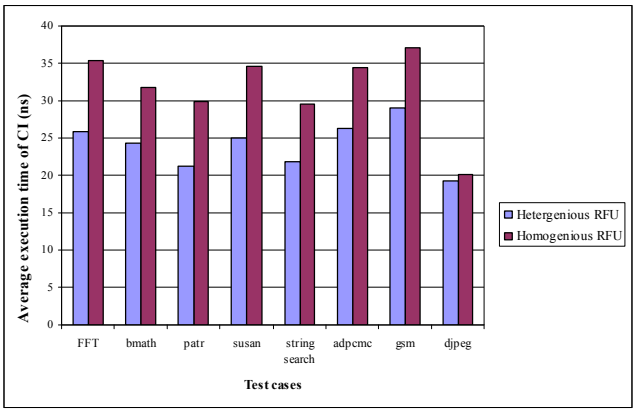

Figure 11. Comparing average execution time of CIs on previous and new architecture

10. 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 we proposed a heterogeneous architecture for 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%. In addition, the CI execution speed on this architecture was improved drastically compared to the previous one. Our RFU consists of 8 inputs, 6 outpus, 3 tri-instruction FUs, 2 bi-instruction FUs and 3 uni-instruction FUs.

As a continuation of this work and to improve the overall runtime of applications, one can work on the architecture and mapping mechanisms to support floating point operations. In addition, it would be worth trying to work on some methods to reduce overall power consumption of the unit.

Reference

[1] Keutzer. K., S.Malik, A.R.Newton, J.M.Rabaey and A.Sangiovanni-Vincentelli. 2000. "System-Level Design: Orthogonalization of Concerns & Platform-based Design". In *IEEE Transactions on CAD of Integrated Circuits and Systems*, 19, No.12, 1523-1543.

[2] Hamid Noori, Farhad Mehdipour, Kazuaki Murakami, Koji Inoue and Morteza 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.

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

[4] 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, Apr.1997.

[5] J. E. Carrillo, and P. Chow, "The effect of reconfigurable units in superscalar processors," in Proc.of the 2001 ACM/SIGDA FPGA, pp. 141–150, 2002

[6] 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.

[7] 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.

[8] 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, Mar. 2000.

[9] J. R. Hauser, and J. Wawrzynek, "GARP: A MIPS processor with a reconfigurable processor," in IEEE Symp. On FPGAs for Custom Computing Machines, Apr. 1997.

[10] K. Compton, and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," ACM Computing Surveys, vol. 34, no. 2, pp. 171–210, 2002.

[11] F. Barat, and R. Lauwereins, "Reconfigurable Instruction Set Processors: A Survey," International Workshop on Rapid System Prototyping, 2000.

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

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

[14] www.Simplescalar.com

[15] 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.