INTEGRATION, the VLSI journal 47 (2014) 232–241

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Design of a coarse-grained reconfigurable architecture with floating-point support and comparative study

Manhwee Jo [a], Dongwook Lee [b], Kyuseung Han [a], Kiyoung Choi [a,*]

[a] Department of Electrical Engineering and Computer Science, Seoul National University, 151-744 Seoul, Korea
[b] Department of Electrical and Computer Engineering, University of Texas at Austin, TX 78712, United States

ARTICLE INFO

Article history:
Received 17 August 2012
Received in revised form
11 August 2013
Accepted 16 August 2013
Available online 28 August 2013

Keywords:
Coarse-grained reconfigurable architecture
Floating-point operations

ABSTRACT

With a huge increase in demand for various kinds of compute-intensive applications in electronic systems, researchers have focused on coarse-grained reconfigurable architectures because of their advantages: high performance and flexibility. This paper presents FloRA, a coarse-grained reconfigurable architecture with floating-point support. A two-dimensional array of integer processing elements in FloRA is configured at run-time to perform floating-point operations as well as integer operations. Fabricated using 130 nm process, the total area overhead due to additional hardware for floating-point operations is about 7.4% compared to the previous architecture which does not support floating-point operations. The fabricated chip runs at 125 MHz clock frequency and 1.2 V power supply. Experiments show 11.6× speedup on average compared to ARM9 with a vector-floating-point unit for integer-only benchmark programs as well as programs containing floating-point operations. Compared with other similar approaches including XPP and Butter, the proposed architecture shows much higher performance for integer applications, while maintaining about half the performance of Butter for floating-point applications.

1. Introduction

With a huge increase of various high-performance multi-media applications running on a portable device, great attention to reconfigurable array architectures has been built up since such architectures can be a key to performance and flexibility [1–6]. According to the granularity, we can classify such architectures into fine-grained reconfigurable array (FGRA) and coarse-grained reconfigurable array (CGRA). A representative example of FGRA is FPGA, which has an array of gates. On the other hand, CGRAs typically have an array of arithmetic and logic units (ALUs) or processing elements (PEs). CGRAs have an advantage over FGRAs in that they can quickly adapt to a new application through dynamic reconfiguration. It is mainly due to the coarser granularity that renders less configuration overheads.

In spite of the advantages of CGRAs, most of the existing architectures are limited to integer-based applications such as audio-visual data codec [7,8], wireless communication [9], cryptography [10], and so on. Thus they are not able to meet the demands for floating-point-based applications effectively. Physics engines in 3-dimensional (3D) graphics are representative examples that cannot be handled efficiently by a conventional CGRA. There have been researches on implementing 3D rendering [11,12] and ray tracing [13] with reconfigurable architectures. However, their approaches have limitations in generating high quality results due to the lack of floating-point computation.

Adding floating-point units (FPUs) to the integer-only reconfigurable architecture can be a solution to the above-mentioned problem. There have been several related researches on FPGAs. [14] introduces an FPGA including FPUs. [15] shows a design-space exploration for efficient implementation of floating-point operations on an FPGA by adding extra modules such as multipliers and FPUs or by modifying look-up tables (LUTs) for efficient binding. However, the area cost due to those approaches is significantly high, especially when multiple floating-point units are added. Moreover, the added units are not utilized at all when integer-based applications are running on the system. By the same token, the integer PEs will be useless when floating-point-based applications are running.

Another solution to the problem is reconfiguring existing units such that they can perform the floating-point operations [16–22]. [16] suggests a novel FPGA architecture and efficient implementation methods of floating-point operations on that architecture. However, implementing floating-point units using LUTs in FPGAs requires much more time to reconfigure the circuit than coarse-grained reconfigurable architecture. Thus it is hard to accelerate applications mixed with integer operations and floating-point operations. Coarse-grained reconfigurable architectures support

* Corresponding author. Tel.: +82 2 880 6768; fax: +82 2 882 4656.
E-mail address: kchoi@snu.ac.kr (K. Choi).

0167-9260/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.vlsi.2013.08.003

reconfiguration with much less reconfiguration time (one cycle in our architecture). Thus the processing elements (PEs) in the architecture can be reconfigured to execute floating-point operations right after executing integer operations. The approaches in [17–20] combine a pair of integer PEs to perform a floating-point operation. Since there are many PEs in an array, it is possible to perform multiple floating-point operations in parallel. For an efficient implementation of the floating-point operations, they use separate FSMs in addition to the configuration of the architecture. In this paper, we present details of the chip implementation and experimental results of a coarse-grained reconfigurable architecture called FloRA (Floating-point-capable Reconfigurable Array), which supports floating-point operations as well as integer operations. Since the floating-point operations are performed with multiple integer PEs, the architecture does not have any separate floating-point units. This allows the architecture to have extended applicability with less hardware overhead.

There are other approaches to implementing floating-point operations exploiting the existing integer functional units in a CGRA. One of them is PACT XPP [21], a commercial coarse-grained reconfigurable architecture [22]. Their approach relies only on configurations of the existing architecture without any additional hardware support for floating-point operations and thus results in an inefficient implementation in terms of performance-to-area ratio. Another approach uses Butter architecture [23–25] where floating-point operations are implemented using its integer addition/subtraction units and multiplication units. Those architectures will be compared with our architecture in the later sections.

The organization of the paper is as follows. Section 2 presents the base hardware architecture of FloRA. Section 3 explains the design for floating-point operations in detail. Section 4 presents the characteristics of the chip and other experimental results obtained from chip test. Section 5 compares the proposed approach to other similar approaches. Finally, Section 6 concludes the paper with some future work.

2. Architecture

2.1. Overall hardware architecture

Fig. 1 shows the overall architecture of FloRA. It consists of a RISC processor, a main external memory block, a DMA controller, and a reconfigurable computing module (RCM). All the components are connected through a data bus. Before executing an application on the architecture, the RISC processor initializes all other components in the architecture. It also controls them during the execution of the application. In addition, it executes control-intensive and irregular code blocks of the application while the RCM accelerates data-intensive and repetitive code blocks such as DSP kernels or matrix-vector calculations, which can be easily parallelized. The DMA controller is used for efficient communications between the RCM and the main memory.

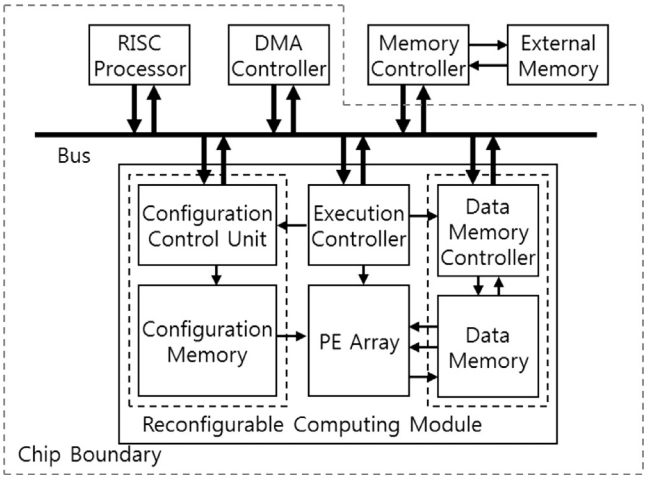

Fig. 1. Overall architecture of FloRA, a coarse-grained reconfigurable array architecture with floating-point capability.

2.2. Reconfigurable computing module

The RCM is in charge of accelerating data-intensive code blocks using an array of PEs. A PE is an ALU-like functional unit that can handle 16-bit integer values. The PE array is designed for accelerating data-intensive kernel code blocks by parallelizing independent operations in a code block on the array of PEs. As shown in Fig. 2, each PE in the array has interconnections to its neighbor PEs (top, bottom, left, and right). Each PE also has interconnections to the PEs in two-hop distance in vertical and horizontal direction, and so on, so that it can communicate directly with other PEs in a cycle via those interconnections without having to pass through neighbor PEs one-by-one along the paths. Such abundant interconnection resources make it easy to map data-flow graphs onto the array [26,27].

Each PE can perform arithmetic operations and logical operations including shift operations and compare and select operations. Thus a PE can be considered as a small processor without instruction fetch unit and branch unit. Some operations (critical operations) such as multiplication and division require functional units that require much larger area and delay than other operations. Each of the critical functional units such as multipliers and dividers is shared by a set of PEs. For this, the executions of those functional units by the PEs are scheduled ahead of time [2]. Since the critical functional units typically have longer delays, they are pipelined so that they do not degrade the overall system throughput. The number of critical functional units integrated into the array is much less than the number of PEs, thereby saving much area and power consumption at the cost of ignorable performance degradation. In case of division, it is a common practice to change divisions into shift operations for applications that are not very sensitive to accuracy. Thus the number of dividers can be further reduced compared to the number of multipliers.

The PEs are configured by configuration control unit (CCU) and a set of configuration memory elements (CEs). Configuration Memory in Fig. 1 is basically an array of CEs. Each CE provides the configuration

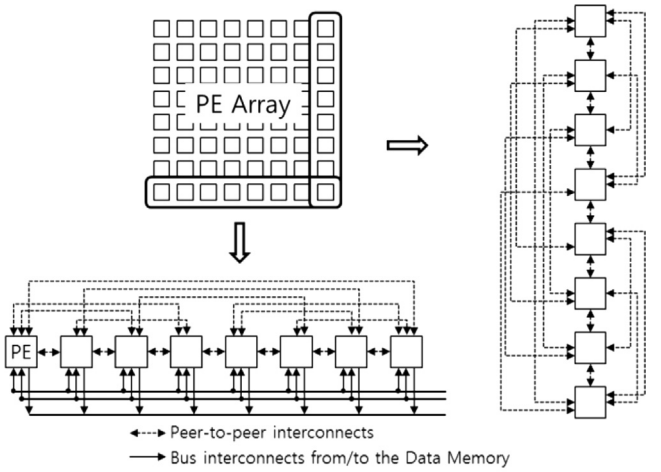

Fig. 2. Interconnection topology in PE Array. Solid line means one-way bus interconnects from/to data memory while dotted line means peer-to-peer interconnects. Each dotted line is physically implemented as two one-way interconnects.