Microprocessors and Microsystems 33 (2009) 441–452

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

Scalable register bypassing for FPGA-based processors

Nikolaos Kavvadias *, Spiridon Nikolaidis

Section of Electronics and Computers, Department of Physics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

ARTICLE INFO

Article history:
Available online 29 July 2009

Keywords:
Microprocessors
Register bypassing
Field-programmable gate arrays
Embedded systems
Hardware description languages

ABSTRACT

In this paper, a scalable scheme, configurable via register-transfer level parameters, for full register bypassing in a modern embedded processor architecture, termed ByoRISC, is presented. The register bypassing specification is parameterized regarding the number of homogeneous register file read and write ports and the number of pipeline stages of the processor. The performance characteristics (cycle time, chip area) of the proposed technique have been evaluated for FPGA target implementations of the synthesizable ByoRISC model. It is proved that, a full bypassing network is a viable solution for the elimination of data hazards when servicing instructions with multiple read and write operands. While the maximum clock frequency is reduced by 17.9% in average, when using partial versus full forwarding, the positive effect of custom computation eliminates this effect by providing cycle speedups of 3.9× to 5.5× and corresponding execution time speedups for a ByoRISC testbed processor of 3.6×. Individual application speedups of up to 9.4× have also been obtained.

1. Introduction

A recent approach to embedded System-on-Chip design involves the use of configurable and extensible processors [1–5], usually in the form of synthesizable cores, offering architecture customization possibilities. Configurability lies in tuning architectural parameters, while extensibility of a processor usually refers either to tightly-coupled modifications obtained by adding single-, multi-cycle or pipelined versions of custom instructions or to loosely-coupled accelerators not directly integrated within the processor pipeline.

These architectural frameworks are regularly updated with enhancements targeting at the improvement of diverse and often conflicting requirements such as low power consumption, performance for the general-purpose or specific application domains, code size and overall system cost. During the development of such processors and especially regarding non-legacy architectures, the designers ought to consider the entire space of architectural solutions regarding the instruction set and underlying microarchitecture. However, it is often that designers limit themselves to solutions that are empirically derived from past practices in order to seemingly reduce complexity without negatively affecting performance. An interesting example is the domination of the three-address instructions limitation which is closely associated to a general-purpose register file with a small number of read and write ports, typically two and one, respectively. While a multi-port register file could provide significant performance boost, it is regarded as unnecessary complexity, dramatically degrading the timing characteristics of the processor, especially if its implementation is combined with data forwarding mechanisms across several pipeline stages.

In this work, we focus on the design and evaluation of data forwarding (register bypassing) architectures, which is a technique for eliminating data hazards in pipelined processors. The function of the bypassing hardware is to resolve data hazards that arise when an instruction needs the results of previous instructions in the pipeline that have not been written to the register file by the time the current instruction reads its source operands from the register file. Generally, it is expected that extensive bypassing comes with a significant impact on cycle time, area and power consumption of the processor.

In this paper, a scalable and parameterized register bypassing scheme is presented that can be utilized in current embedded processors. The specification of the bypassing architecture can be configured for the desired number of register file read/write ports and pipeline stages of the processor in mind. The main contributions of this paper can be summarized as follows:

- Development of a clear and concise register bypassing specification that is fully parameterized and can be easily applied to different processors. Especially, it can be of particular interest to developers of new/emerging processor architectures for providing more architectural options to their end users.
- The effect of the bypass circuitry on the timing and area of a representative processor are carefully examined. Most previous works only model either a partial processor or solely the bypassing mechanisms.

* Corresponding author. Tel.: +30 2310 998079; fax: +30 2310 998018.
E-mail addresses: nkavv@physics.auth.gr (N. Kavvadias), snikolaid@physics.auth.gr (S. Nikolaidis).

0141-9331/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2009.07.002

- Specific issues regarding targeting the bypass specification to recent FPGA devices are highlighted. FPGAs have been neglected as an implementation medium even in recent works on the subject.

The rest of this paper is organized as follows. Related work is summarized in Section 2. The processor pipeline model is briefed in Section 3. Section 4 discusses the details of the scalable register bypassing (SRB) specification, and in Section 5 its performance is evaluated in terms of timing characteristics and area requirements as well as in context of an image processing application set running on an embedded RISC processor. Finally, Section 6 summarizes the paper.

2. Related work

In related work, a number of approaches have been proposed for the evaluation of register bypassing networks [6–10]. Most of them deal with exploring the design space of partial bypassing for an application set, representative of a particular domain, in order to drive the customization and reduction of a full bypass network. In the aforementioned works, neither a concise formalism nor a reusable model of bypassing, applicable to FPGAs, is presented that can provide sufficient assistance to the processor designer. Further, it is common that the bypass network is evaluated for timing and area characteristics apart from the processor, while a processor model taking account only the cycle behavior of using the bypassing mechanisms is used separately for obtaining execution cycles measurements.

In an early work in this field, Abnous and Bagherzadeh [6] analyzed partial bypassing between VLIW functional units in their 4-integer-unit VIPER processor. They argued that complete bypassing is too costly in VLIW processors even though significant performance benefits can be achieved. The pipeline model of VIPER is rather inflexible and cannot be used for exploration purposes: it is restricted to four stages, deduced from the classic 5-stage pipeline of early RISCs by removing the memory access stage. In order to achieve this, the processor model is limited to a single addressing mode (register indirect).

Further, in [7] the architecture of a detailed bypassing execution unit model is described and applied for a multiple instruction issue processor. Similar to [6] the processor model features a four-stage pipeline, but in this case with configurable multiplicity of execution datapaths. A design space exploration approach for eliminating infrequently used routes in register bypass networks has been presented in [8] applied to the case of a 5-issue custom VLIW processor. In a similar architectural context, low power optimizations that exploit the forwarding paths of a fixed register bypass network, for the purpose of minimizing power-costly accesses to/from the register file have been also examined [10].

In [9] an operation table formalism was developed for capturing the bypass mechanisms in pipelined embedded processors, along with an automation tool (PBExplore) for exploring the design space, constituted of the partial bypassing solutions, in terms of achievable performance. The authors assume that full register bypassing is not a viable solution, thus partial bypassing is preferred. However, the integration of the bypass networks within a synthesizable description of their testbed architecture (Intel XScale) is not considered at all, even though this would be necessary in order to evaluate the effect of the bypassing network on the processor cycle time and aggregate area. Further, in their work, processor implementations on FPGAs have not been considered at all.

A recent technique [11] on the design of register bypasses involves a compiler-driven approach based on the fact that certain register addresses are not actually read, given that the corresponding operands are forwarded to the appropriate ports by the bypass network. In this case, the processor instructions have to be statically rewritten to free the corresponding fields in order to derive the appropriate control signals. Their architectural model is closer to our approach, incorporating a multi-port register file and a configurable number of pipeline stages. However, the main aim of this technique is the energy consumption and area reduction of the redundant bypasses for a VLIW ASIC processor model, and not the thorough investigation of the practicality of full register bypassing on FPGA-based soft processors.

An extensive design space exploration of clustered VLIW architectures, typically employing 2, 4 or 8 partitioned register files can be found in [12]. The complexity of a full bypass network is reduced due to the smaller number of read and write ports of the partitioned register files, with the tradeoff of introducing copy operations among these. For the case of the unicluster architecture which is also investigated as a reference, it is stated that its performance is significantly lower to the clustered architecture. While this is true, the exploration targets a standard cell VLSI process, and performance on modern FPGA devices is not discussed.

A common denominator of some of these works [8,9] is the consideration of compiler visibility of the partial bypasses. Here, although partial bypassed networks are possible, we focus on presenting a scalable specification for full register bypassing hardware that is fully transparent to the programmer.

3. An abstracted view of the ByoRISC processor model

The architectural model targeted in this paper, shown in Fig. 1, is the ByoRISC processor [13]. A ByoRISC processor can be extended by application-specific hardware extensions (ASHEs) in the form of either custom instruction units or locally-interfaced coprocessors. Such ASHEs can implement multi-input multi-output (MIMO) computations with local state that may have an arbitrary number and combination of load/store accesses to the data memory. The ByoRISC template employs a pipeline stage structure consisting of:

- an instruction fetch stage, IF (not shown in Fig. 1) of a possibly wide instruction, incorporating one or more micro-operations to be executed in their corresponding execution slots;
- a custom instruction operand address access stage and an instruction decode stage where *NRP* register operands are read;
- *NPIPE* execution stages with at least one of them accessing the data memory for full support of ByoRISC ASHEs;
- a register write-back stage for writing *NWP* register operands.

The basic assumption for the first execution stage is that it receives up to *NRP* read register operands from a multi-ported register file and produces a result vector of up to *NWP* write register operands. There is no limitation on the policy followed in the architecture for the incorporated functional units: the processor can present a VLIW/EPIC architecture, servicing a number of independent micro-operations in the same control step, or it can evaluate MIMO (multiple-input multiple-output) instructions [14] that are represented by data-dependence directed acyclic graphs of basic block scope at the level of compiler intermediate language. The subsequent execution stages accept the result vector from their preceding stage, which is of width $NWP \times DW$, where DW is the register word width. Further, it can be specified that they read up to *NRP* from the forwarded read operands, given that these have been stored in the corresponding pipeline registers. The final pipeline stage is responsible for committing the final result vector to the register file. Additional computations do not take place at this stage, so reading the read register operand vector and the corresponding register addresses (assumed through the figure) is not necessary. Any of the *NPIPE* execution stages can be configured