Dynamically Reconfigurable Register File for a Softcore VLIW Processor

Stephan Wong, Fakhar Anjam, and Faisal Nadeem
Computer Engineering Laboratory
Delft University of Technology, Delft, The Netherlands
E-mail: {J.S.S.M.Wong, F.Anjam, M.F.Nadeem}@tudelft.nl

***Abstract*—This paper presents dynamic reconfiguration of a register file of a Very Long Instruction Word (VLIW) processor implemented on an FPGA. We developed an open-source reconfigurable and parameterizable VLIW processor core based on the VLIW Example (VEX) Instruction Set Architecture (ISA), capable of supporting reconfigurable operations as well. The VEX architecture supports up to 64 multiported shared registers in a register file for a single cluster VLIW processor. This register file accounts for a considerable amount of area in terms of slices when the VLIW processor is implemented on an FPGA. Our processor design supports dynamic partial reconfiguration allowing the creation of dedicated register file sizes for different applications. Therefore, valuable area can be freed and utilized for other implementations running on the same FPGA when not the full register file size is needed. Our design requires 924 slices on a Xilinx Virtex-II Pro device for dynamically placing a chunk of 8 registers, and places registers in multiples of 8 registers to simplify the design. Consequently, when 64 registers is not needed at all times, the area utilization can be reduced during run-time.**

I. Introduction

Very Long Instruction Word (VLIW) processors exploit Instruction Level Parallelism (ILP) found in applications by means of a compiler. By issuing multiple operations in one instruction, a VLIW processor is able to accelerate applications compared to a Reduced Instruction Set Computer (RISC) system [1][2]. In [2], we presented the design and implementation of an open-source reconfigurable softcore VLIW processor called ρ-VEX, accompanied by a development framework. A 4-issue ρ-VEX processor is depicted in Figure 1. This processor architecture is based on the VLIW Example (VEX) Instruction Set Architecture (ISA), as introduced in [3]. VEX offers a scalable platform that allows variation in many aspects, including instruction issue width, organization of functional units, and instruction set. A software development toolchain for VEX is made freely available by Hewlett-Packard [4].

The VEX architecture supports up to 64 multiported registers in a register file for a single cluster VLIW processor [3]. The number of read and write ports on the multiported registers increases with the increase in issue-slots. Implementing such a processor on an FPGA consumes considerable amount of area in terms of slices. Fixing the number of registers means occupying a fixed area on the FPGA. For example, for our 4-issue ρ-VEX VLIW processor, creating 64 registers requires 8594 Xilinx Virtex-II Pro slices (62% of the *XC2VP30* device slices). Because not all applications require 64 registers, implementing 64 registers would mean a lot of wasted slices as well as wasted power. On the other hand, implementing a smaller number of registers, for example, 8 registers would consume a smaller number of slices (909 slices - without the area overhead for partial reconfiguration), but may degrade the performance of an application, as memory swapping (through loads/stores) may then be needed more frequently.

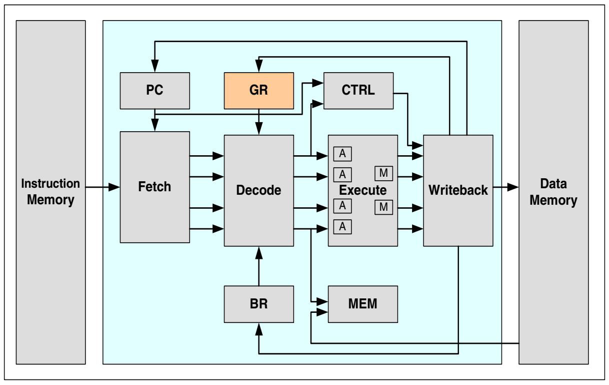

Figure 1. 4-issue ρ-VEX VLIW Processor

We modified our processor ρ-VEX to support dynamic partial reconfiguration. The processor can dynamically create its own register file composed of the actual number of registers the application needs, potentially saving considerable FPGA area. In our current implementation, it takes 1.29 milliseconds to configure 8 registers. A matrix multiplication example shows that it takes the same number of cycles (335 cycles) to complete, whether the VLIW processor has 24 or 32 or even 64 registers. Therefore, configuring 24 registers for this application would save 5626 slices compared to the case in which we have 64 permanent registers. Moreover, this can potentially be translated into considerable power savings. To optimally utilize the pre-placed reconfigurable area, we investigated the possibility of dynamic sharing of this saved area between multiple clusters of our processor. We designed a multiprocessor system with two processor cores sharing a single register file to optimally utilize the resources.

The remainder of this paper is organized as follows. Section II discusses work related to softcore VLIW processors and multiported register file. Section III presents our ρ-VEX VLIW processor. Section IV presents the design and implementation of a reconfigurable register file for our ρ-VEX processor. Results are discussed in Section V. Finally, conclusions are presented in Section VI.

II. Related Work

The first VLIW softcore processor found in literature is Spyder [5]. Because the designer had to put efforts in both directions - the processor architecture and the compiler - therefore, the Spyder did not evolve extensively. Instance-specific VLIW processors are presented in [6][7]. A VLIW processor with reconfigurable instruction set is presented in [8]. An FPGA based design of a VLIW softcore processor is presented in [9]. Due to the licensed Altera NIOS-II softcore, this VLIW design is not much flexible as well as not open source. Modular and configurable VLIW processor designs are presented in [10][11] and [12]. The lack of a good software toolchain in each of these cases limited their use.

In [2], we presented the design of a reconfigurable and open source softcore VLIW processor called ρ-VEX, along with a development framework. Different parameters such as the number and types of functional units, supported instructions, degree of pipelining, memory bandwidth, size of register file can be chosen based on the application requirements and available resources on the FPGA.

The multiported register file of a VLIW processor implemented on FPGA is one of the most resource consuming module. Table I presents the synthesis results for our 4-issue, 32-bit ρ-VEX processor for Xilinx Virtex-II Pro *XC2VP30* FPGA. Table II presents the synthesis results for the register file only. From Tables I and II, we can observe that when the number of registers is 64, the total number of slices occupied by the register file exceeds the total slices taken by the all other processor modules. Similarly, in [12], the design and implementation of a 3-issue VLIW microprocessor is presented. The processor datapath is 64-bit and it supports only 16 operations. The multiported register file for the processor containing 16, 64-bit registers each having 6-read and 3-write ports require 7172 Logic Cells of the Altera Stratix *EP1S25F1020C* FPGA, which is more than the area taken by the rest of the design.

The data in [9] shows that increasing the number of read and write ports or in other words increasing the issue-width of the VLIW processor results in an exponential increase in

Table I
Synthesis Results for 4-issue ρ-VEX VLIW Processor

Processor with Register File size	Slices	Percent Slices of XC2VP30 FPGA	Max. Frequency (MHz)
8	6647	48%	74.369
16	7523	54%	74.614
24	8616	62%	74.559
32	9432	68%	74.520
64	14469	105%	64.255

Table II
Synthesis Results for Register File of ρ-VEX VLIW Processor

Register File size	Slices	Percent Slices of ρ-VEX Processor	Max. Frequency (MHz)
8	909	14%	495.123
16	1865	24%	433.332
24	2968	34%	341.041
32	3778	40%	385.253
64	8594	59%	303.012

resources. In [13], the architecture of a configurable, multiported register file for soft processor cores is presented. The register file is designed using the low-latency Block-RAMs found in Xilinx FPGAs, and avoids consuming the configurable resources on the FPGA (slices). In this design, since registers are distributed across several banks associated with different write ports, registers must be allocated, and instructions scheduled in a manner that avoids contention for the write ports. Therefore, instructions cannot be scheduled to execute in parallel if they produce results in registers that belong to the same register bank. Because this requirement puts a constraint on the scheduling of instructions, we cannot use this design for our processor because the Hewlett-Packard (HP) provides only the VEX compiler executable [4], not the source code, and hence we cannot modify the compiler.

We designed our VLIW processor in a manner that supports partial dynamic reconfiguration using Xilinx Early Access Partial Reconfiguration (EAPR) methodology [14] and PlanAhead tools. The processor can dynamically create its own register file composed of the actual number of registers the application needs or as given by the user before each application runs, potentially saving considerable FPGA area (slices) in case the running application requires less number of registers than the maximum number of could-be available registers. Our results for running different applications proves the idea.

III. The ρ-VEX VLIW Processor

We implemented a single-cluster standard configuration of VEX machine for our VLIW processor called ρ-VEX [2]. The ISA is based on the VEX ISA [3], which is loosely modeled on the ISA of the HP/ST Lx [1] family of VLIW embedded cores. Figure 1 depicts the organization of our 32-bit, 4-issue ρ-VEX VLIW processor implemented on an FPGA. It additionally supports reconfigurable operations, as the VEX compiler supports the use of custom instructions via pragmas inside the application code. To optimally exploit the processor utilization, a development framework is provided, which consists of compiling a piece of C code with VEX compiler and then generating a VHDL instruction ROM file by assembling the assembly file with our assembler [2].

IV. The Reconfigurable Register File for ρ-VEX Processor

The default register file size for a single-cluster VEX machine is 64. For our 4-issue processor, the total number of input/output (I/O) ports of the register file is 460. Each register is 32-bit, having 8-read and 4-write ports.

A. Design and Implementation

We used the Xilinx EAPR methodology [14] for designing our partial reconfigurable processor. Because we only want the registers to be partially reconfigured, we split our processor into two regions: *static* and *reconfigurable*. Figure 2 depicts the static and partially reconfigurable regions of our processor. For this particular design, we can create a maximum of 32 registers instead of 64. This choice simplified the design to allow for a quick verification of our idea of dynamically reconfigurable registers. To further simplify the design, we divided the large register file containing 32 registers into 4 small register files each having 8 registers. The number of I/O ports for each smaller register file is now 424 that would cross the boundary between static and reconfigurable portions on the FPGA, and would require *bus macros*.

Figure 2 depicts the overall system design. Four reconfigurable regions for register banks are connected to the other part of the design called static region using asynchronous bus macros. The static region contains all of our processor modules namely, *fetch*, *decode*, *execute*, *writeback*, *memory unit*, *control unit*, *branch registers*, *instruction* and *data memories*, except the general-purpose register file, which is implemented in the reconfigurable region. The granularity level is 8 for our reconfigurable register design, i.e., registers can be added in the increments of 8 up to maximum of 32 registers.

Using the EAPR design methodology, partial bitstreams for the register banks are generated and can be downloaded to the FPGA using the Xilinx iMPACT tool. The actual number of registers used in an application code can be determined when assembling the code. The assembler for the processor calculates the number of required registers, and this information is placed in the executable code or ROM storage. This information can be used to direct the reconfiguration controller to reconfigure the required number of registers before activating the run signal for the processor. A Universal Asynchronous Receiver Transmitter (UART) controller is implemented in the static region of the FPGA and is used to download the application executable file into the instruction memory. The information about the number of registers to be configured can be transmitted to the PC and then using batch files, partial bitstreams for register banks can be downloaded to the FPGA using iMPACT commands. When the application execution is finished, the results are transmitted to the PC using the UART.

B. Dynamic Sharing of Registers in Multicore Systems

One of the main limitations of the current tools is that the reconfigurable regions have to be pre-placed at design time. Current tools and technology are not mature enough for partial reconfiguration and these do not allow arbitrary placement of reconfigurable modules at run-time. Therefore, for the current design, we investigated the possibility of dynamic sharing of the registers between multiple instances of our processor. We designed a two-core processor system sharing the placed registers to make the optimal use of the resources. Figure 3 depicts a two-core processor system sharing the register file.

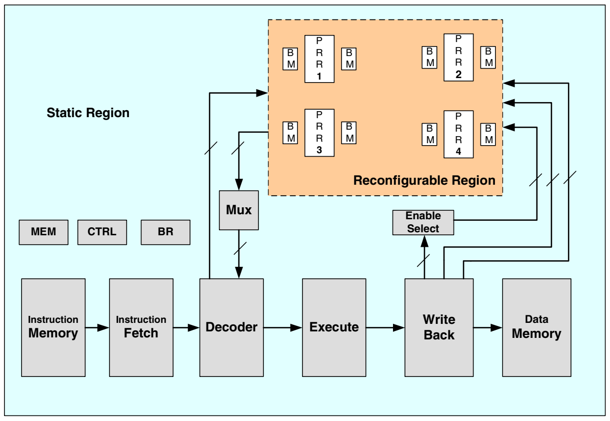

Figure 2. Dynamically and Partially Reconfigurable Processor Design

We developed a *register controller* module that is used to share the registers between two processor cores. It provides connectivity between the two cores and the reconfigurable registers. The register controller is placed in the static region of the design.

V. Results and Discussions

Each reconfigurable register file in our design has 424 I/O ports and is composed of 8 32-bit registers having 8 read and 4 write ports and requires 909 slices on the Xilinx Virtex-II Pro *XC2VP30* FPGA. For placement, we constrained this register file to a bank of 924 slices. Table III presents the slices utilization for our design.

The partial bitstream size for a reconfigurable register bank of 8 registers in our design is 85 KBytes, and is 16.65 times smaller than the full bitstream size which is about 1415 KBytes. The time needed to configure a bank of 8 registers in a Virtex-II Pro device is 1.29 milliseconds. This value does not include the time needed for accessing the memory in which the bitstreams are placed. It is the time needed for the SelectMAP

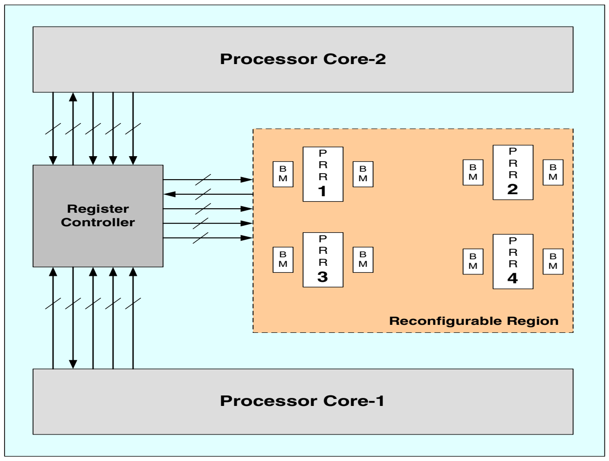

Figure 3. Dynamic Sharing of Register File in Multicore System

Table III
SLICE UTILIZATION FOR THE RECONFIGURABLE DESIGN

Region	Slices
Reconfigurable Bank of 8 Registers	924
Static Region	1437
Bus Macros	848

and Internal Configuration Access Port (ICAP) to configure the device.

To measure the effectiveness of our design, we used three applications: a *matrix multiplication* program, a program to calculate an element of the *Fibonacci series* and the *CCITT ADPCM encoder/decoder*. The number of cycles to complete the multiplication (3-by-3 matrices) and to calculate the 45^{th} element of the Fibonacci series against different number of registers are presented in Table IV.

From Table IV, it can be observed that when the number of registers is 24 or 32, the processor takes the least number of cycles to complete the multiplication. When the number of registers is reduced to 16 or 8, the cycle count increases. Therefore, keeping 32 or 64 registers for such a program would mean a waste of a large number of resources (3778 - 2968 = 810 slices or 8594 - 2968 = 5626 slices). The saved resources can be allocated to another processor core and another instance of the same program or different program can be run on it improving the overall performance of the system. We ran two instances of the same matrix multiplication program on two cores each with 24 registers effectively doubling the performance compared to one permanently placed 64 registers on a single core processor. Similarly, the number of cycles to calculate an element of the Fibonacci series remains the same with 8, 16, 24, 32, or 64 registers, hence we can configure only 8 registers to run this program and save a large FPGA area for other applications. Finally, Table IV presents the cycle count results against different number of registers for running the CCITT ADPCM encoder and decoder applications. It can be observed that increasing the number of registers has almost no effect on the cycle count, hence the minimum number of registers (e.g., 8) can be configured for these programs and extra area can be saved.

VI. CONCLUSIONS

In this paper, we presented the design and implementation of a dynamically and partially reconfigurable mutliported register file for a VLIW processor. Because the register file consumes a considerable amount of resources when the VLIW processor is implemented on an FPGA, and its area (slices) requirement increases with increasing the issue-width of the processor, permanently creating a large number of registers would consume a large number of slices. Similarly, creating a small number of registers may degrade the performance of the VLIW processor. We developed the register file for our VLIW processor ρ-VEX that supports dynamic and partial reconfigurability. Using the EAPR methodology, we designed our processor in a manner that its registers can be dynamically

Table IV
CYCLE COUNT FOR BENCHMARK APPLICATIONS

No. of Registers	Cycle count			
	Matrix Multiplication	Fibonacci Series	ADPCM Encoder	ADPCM Decoder
8	495	561	211622696	229277148
16	395	561	211556705	229277140
24	335	561	211556705	229277140
32	335	561	211556705	229277140

configured as per the requirement of the application before it is run. The design can save a large number of slices as well as power without affecting the cycle count for the selected applications. We additionally investigated dynamic sharing of the registers between multiple cores of our processor in order to optimally utilize the pre-placed area in the current design and improve the performance of the system.

REFERENCES

[1] P. Faraboschi, G. Brown, J.A. Fisher, G. Desoli, and F. Homewood, "Lx: A Technology Platform for Customizable VLIW Embedded Processing", in *Proceedings of the 27th annual International Symposium of Computer Architecture (ISCA 00)*, pp. 203 - 213, 2000.

[2] S. Wong, T.V. As, and G. Brown, "ρ-VEX: A Reconfigurable and Extensible Softcore VLIW Processor", in *IEEE International Conference on Field-Programmable Technologies (ICFPT 08)*, 2008.

[3] J.A. Fisher, P. Faraboschi, and C. Young, *Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools*. Morgan Kaufmann, 2004.

[4] Hewlett-Packard Laboratories. VEX Toolchain. [Online]. Available: http://www.hpl.hp.com/downloads/vex/.

[5] C. Iseli and E. Sanchez, "Spyder: A Reconfigurable VLIW Processor using FPGAs", in *FPGAs for Custom Computing Machines*, pp. 17-24, 1993.

[6] C. Grabbe, M. Bednara, J.V.Z. Gathen, J. Shokrollahi, and J. Teich, " A High Performance VLIW Processor for Finite Field Arithmetic", in *Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS 03)*, 2003.

[7] M. Koester, W. Luk, and G. Brown, "A Hardware Compilation Flow For Instance-Specific VLIW Cores", in *Proceedings of the 18th International Conference on Field Programmable Logic and Applications (FPL 08)*, 2008.

[8] A. Lodi, M. Toma, F. Campi, A. Cappelli, and R. Canegallo, "A VLIW Processor with Reconfigurable Instruction Set for Embedded Applications", in *IEEE Journal on Solid-State Circuits*, vol. 38, no. 11, pp. 1876 - 1886, 2003.

[9] A.K. Jones, R. Hoare, D. Kusic, J. Fazekas, and J. Foster, "An FPGA-based VLIW Processor with Custom Hardware Execution", in *Proceedings of the 2005 ACM/SIGDA 13th Internal Symposium on Field Programmable Gate Arrays (FPGA 05)*, pp. 107 - 117, 2005.

[10] V. Brost, F. Yang, and M. Paindavoine, "A Modular VLIW Processor", in *IEEE International Symposium on Circuits and Systems (ISCAS 2007)*, pp. 3968 - 3971, 2007.

[11] M.A.R. Saghir, M. El-Majzoub, and P. Akl, "Customizing the Datapath and ISA of Soft VLIW Processors", in *High Performance Embedded Architectures and Compilers (HiPEAC 07)*, LNCS 4367, pp. 276-290, 2007.

[12] W.F. Lee, *VLIW Microprocessor Hardware Design For ASICs and FPGA*. McGraw-Hill, 2008.

[13] M.A.R. Saghir, and R. Naous, "A Configurable Multi-ported Register File Architecture for Soft Processor Cores", *International Symposium on Applied Reconfigurable Computing (ARC 07)*, LNCS 4419, pp. 14 - 25, 2007.

[14] Xilinx, Inc. 2006. User Guide UG208: Early Access Partial Reconfiguration User Guide, http://www.xilinx.com.