A Shared Reconfigurable VLIW Multiprocessor System

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

***Abstract*—In this paper, we present the design and implementation of an open-source reconfigurable very long instruction word (VLIW) multiprocessor system. This processor is implemented as a softcore on a field-programmable gate arrays (FPGA) and its instruction set architecture (ISA) is based on the Lx/ST200 ISA. This multiprocessor design is based on our earlier ρ-VEX processor design. Since the ρ-VEX processor is a parameterized processor, our multiprocessor design is also parameterized. By utilizing a freely available compiler and simulator in our development framework, we are able to optimize our design and map any application written in C to our multiprocessor system. This VLIW multiprocessor can exploit data level as well as instruction level parallelism inherent in an application and make its execution faster. More importantly, we achieve our results by saving expensive FPGA area through the sharing of resources. The results show that we can achieve two times better performance for our dual-processor system (with shared resources) compared to a uni-processor system or a 2-cluster processor system for applications having data level and instruction level parallelism.**

I. Introduction

Embedded systems have become commonplace nowadays and they are being utilize for many different applications such as image processing, computer vision, networking, wireless communication, etc. Because these applications offer a good amount of functional and data level parallelism, they can achieve better performance when run on multiprocessor systems rather than on uni-processor systems. Further, to exploit instruction level parallelism, very long instruction word (VLIW) processors can be utilized to increase the performance beyond the single issue or reduced instruction set computer (RISC) architectures [1]. While RISC architectures only take advantage of temporal parallelism (by using pipelining), VLIW architectures can additionally take advantage of the spatial parallelism by using multiple functional units (FUs) to execute several operations simultaneously. VLIW multiprocessor systems (where each processor is a VLIW processor) can support both fine-grain (instruction level) as well as coarse-grain (data level) parallelism.

Field-programmable gate arrays (FPGAs) have become a widely used tool for rapid prototyping, providing software-like flexibility and hardware-like performance. For an application to take advantage of performance improvement from FPGA implementations, it must possess inherent parallelism. Applications in different domains such as multimedia, bioinformatics, wireless communication, numerical analysis, etc. contain a high level of instruction level parallelism (ILP), as they have many independent repetitive calculations. VLIW processors such as the Lx/ST200 [1] from HP and STMicroelectronics and the TriMedia [2] from NXP exploit the ILP found in an application by means of a compiler. By issuing multiple operations in one instruction, a VLIW processor is able to accelerate an application many times compared to a RISC system [1][3]. This is further enhanced by the fact that VLIW processors are simpler in design when compared to their more complex (out-of-order) RISC counterparts.

However, the higher performance of the VLIW processor does not come for free as other resources do not scale well. For example, the number of read ports on the shared multi-ported register file in a VLIW processor is twice the issue-width, and the number of write ports is equal to the issue-width (assuming that each FU requires two input operands and writes one output as a result). It means that the resource/area requirements for a multi-ported register file are directly proportional to the product of the number of read and write ports, therefore, these parameters are not scalable to a large extent. To reduce the pressure on the number of read and write ports of the shared register file, a *clustered* architecture is used. A cluster is a collection of register files and a tightly coupled set of FUs. A multi-cluster processor has multiple clusters, but a single execution thread, while a multiprocessor has multiple processors, but may have multiple execution threads. Clustered VLIW does not scale well in terms of performance. Clustering costs about one fourth of the machine functional units in terms of performance. For example, a 16-unit/2-cluster machine performs roughly like a 12-unit/1-cluster and an 8-unit/2-cluster machine like a 6-unit/1-cluster machine [1]. The reason for this performance penalty is the inter-cluster communication. Additionally, in some cases, the compiler is not able to extract ILP for larger issue-width processors. This situation is called as the ILP saturation.

To avoid the performance penalty due to inter-cluster communication in a multi-cluster VLIW processor, and to avoid the ILP saturation problem in a very-wide issue single-cluster VLIW processor, we present the design and implementation

of a reconfigurable VLIW multiprocessor system. By utilizing multiple independent processors in our multiprocessor system, we are able to avoid these problems. The base processor of this multiprocessor system is an open-source, extensible and reconfigurable softcore VLIW processor called as the *Delft Reconfigurable VLIW processor ρ-VEX* [3][4]. The processor architecture is based on the VEX (VLIW Example) instruction set architecture (ISA), as introduced in [5], and is implemented as a softcore on an FPGA. Parameters of the multiprocessor such as the number and type of processors can be chosen by the designer. All processors of the multiprocessor system have the same ISA but parameters of the individual VLIW processors such as the number and type of functional units, supported instructions, memory-bandwidth, and register file size can be chosen based on the application and the available resources on the FPGA. A software development toolchain including a highly optimizing C compiler and a simulator for VEX is made freely available by Hewlett-Packard (HP) [6]. We present a development framework to optimally utilize the processor system. Any application written in C can be executed on the processor implemented on the FPGA. The ISA can be extended with custom operations and the compiler is able to generate code for the custom hardware units, further enhancing the performance.

The remainder of the paper is organized as follows. Section II explains the rationale behind the project. In Section III, some previous work related to softcore processors and multiprocessors is discussed. The VEX VLIW processor architecture and the available software toolchain are briefly discussed in Section IV. Section V presents our softcore VLIW processor ρ-VEX. The design and implementation of our multiprocessor system are presented in Section VI. Experimental results are discussed in Section VII. Finally, conclusions are presented in Section VIII.

II. The Rationale

In [1] and [3], it has been shown that the performance of a VLIW processor is high compared to a RISC processor for applications which have inherent ILP. The most important factor in the design of a VLIW processor is its compiler. Engineering a production-quality, high-performance optimizing compiler is a work requiring person-decades and a great deal of sophistication [5]. Therefore, it is always recommended that VLIW designers start with the available compiler, not the available hardware, when considering the space of possible designs. Based on the above fact, we have chosen to implement a softcore VLIW processor based on the Lx architecture [1], for which the Hewlett-Packard (HP) provides a free software toolchain (C compiler and simulator).

VLIW and superscalar are the two main architectures which can exploit ILP in a single core processor. Both architectures exploit ILP by issuing multiple operations per issue-slot to additional functional units (FUs). The main difference between a superscalar and VLIW processor is that in a superscalar processor, a special control hardware enables the dynamic scheduling of operations while in a VLIW processor, operations are scheduled statically by the compiler. Therefore, the hardware of a VLIW processor is very simple compared to the hardware of a superscalar processor at the expense of a complex compiler. Because a superscalar processor has extra hardware logic for dynamic scheduling, its area and power requirements on a die are higher compared to a VLIW processor, therefore it is less attractive for embedded applications that require small and energy-efficient devices. Therefore, we chose to implement a VLIW processor instead of a superscalar processor as a good (open-source) compiler already exists.

Soft multiprocessor systems as can be found in literature are normally based on some proprietary intellectual properties (IPs), such as the Xilinx MicroBlaze and the Altera NIOS-II softcore processors. The MicroBlaze and the NIOS-II are single issue/RISC type processors, require a license, and are not open-source. VLIW processors are used to increase the performance of applications having ILP beyond a normal RISC processor. It has been shown in [1] that increasing the issue-width beyond a certain number no longer increases the performance. Additionally, increasing the number of clusters in a VLIW processor does not add too much to the performance, rather reduces the performance in some cases. This performance penalty is due to the inter-cluster communication. In order to avoid this penalty, we designed a multiprocessor system, where each processor is a reconfigurable VLIW processor based on the VEX ISA. Each processor is parameterized, and therefore, a different mix of these processors can be combined to form a multiprocessor system. This VLIW multiprocessor system is able to exploit both instruction level as well as data level parallelism. The design is open-source as well as the development toolchain is freely available.

III. Related Work

The first VLIW softcore processor found in literature is Spyder [7]. A compiler toolchain was also made available. Because the designer had to put efforts in both directions, the processor architecture and the compiler, therefore, the processor did not evolve extensively. Instance-specific VLIW processors are presented in [8][9]. These architectures are specific implementations for some applications, and do not represent a more general VLIW processor. A VLIW processor with reconfigurable instruction set is presented in [10]. An FPGA-based design of a VLIW softcore processor is presented in [11]. Additionally, this processor is able to execute custom hardware. It has an ISA that is binary-code compatible with the Altera NIOS-II soft processor. To support this architecture, a compilation and design automation flow are described for programs written in C. The compilation scheme consists of a Trimaran [12] as the front-end and the extended NIOS-II as the back-end. Due to the licensed Altera NIOS-II, this VLIW design is not much flexible and not open-source. In [13], a modular design of a VLIW processor is reported. Certain parameters of the processor architecture could be altered in a modular fashion. The lack of a good software toolchain and the absence of parametric extensibility limited

the use of this architecture. In [14], the architecture and micro-architecture of a customizable soft VLIW processor are presented. Additionally, tools are discussed to customize, generate and program this processor. Performance and area trade-offs achieved by customizing the processor's datapath and ISA are evaluated. The limitation is the absence of a compiler. In [15], the design and architecture of a VLIW microprocessor is presented without having any toolchain, which restricts the processor usability.

In [3] and [4], we presented the rationale and the design and implementation of an open-source softcore reconfigurable VLIW processor called as the Delft Reconfigurable VLIW processor ρ-VEX, accompanied by a development framework. The processor architecture is based on the VEX ISA [5].

Softcore multiprocessor systems as found in literature are mostly based on either the Xilinx MicroBlaze or the Altera NIOS-II softcore processors. Altera provides a tutorial [16] for creating a multiprocessor system using their softcore NIOS-II processor. The tutorial provides a complete design flow from hardware building to software programming. [17] gives a design of a symmetric multiprocessing on programmable chips using Altera NIOS-II softcore as the basic building block. Similarly, there are many design and implementations of multiprocessor systems using the Xilinx MicroBlaze softcore processor [18][19][20][21][22]. The main drawback of all these designs is that they are using proprietory softcores which are not open-source. In addition, the NIOS-II and MicroBlaze are single issue processor cores and can not exploit ILP as can be achieved by a VLIW processor core [3].

In this paper, we present the design and implementation of a softcore VLIW multiprocessor system. The base processor of this system is the open-source *Delft Reconfigurable VLIW processor* ρ-VEX previously developed at our group [3][4]. This processor is parameterized in many aspects as discussed above. Parameters of the multiprocessor system such as the number and type of the base processor can be changed. All individual processors of the multiprocessor system have the same ISA, but can have different parameters as discussed above. This VLIW multiprocessor system can exploit both instruction level as well as data level parallelism, while a multiprocessor system based on MicroBlaze or NIOS-II can only exploit data level parallelism. This multiprocessor system can save the performance penalty as is in the case of a multi-clustered VLIW processor system, as now the communication is done at processor level. Additionally, we reduced the area/power overhead of the multi-ported register file by utilizing a smaller issue-width VLIW processor (such as 4-issue) as the basic building block for our multiprocessor system instead of using a larger issue-width VLIW uni-processor alone for the application.

IV. The VEX VLIW Processor: ISA and the Toolchain

Compared to superscalar and RISC processors, the VLIW architectures require a more powerful compiler, because of operation scheduling. [5] gives the definition of the VLIW design philosophy as: *"The VLIW design philosophy is to design processors that offer ILP in ways completely visible in the machine-level program and to the compiler"*.

A. The VEX System

The VEX is a system developed according to the VLIW philosophy by Hewlett-Packard (HP) and STMicroelectronics. The VEX includes three basic components [5]:

1) *The VEX ISA*: VEX instruction set architecture (ISA) is a 32-bit clustered VLIW ISA that is scalable and customizable to individual application domains. The VEX ISA is loosely modeled on the ISA of HP/ST Lx/ST200 family of VLIW embedded cores [1]. VEX ISA is scalable because different parameters of the processor such as the number of clusters, FUs, registers, and latencies can be changed. VEX ISA is customizable because special-purpose instructions can be defined in a structured way.
2) *The VEX C Compiler*: Based on *trace scheduling*, the VEX C compiler is an ISO/C89 compiler. It is derived from the Lx/ST200 C compiler, which itself is a descendant of the *Multiflow* C compiler. A very flexible programmable machine model determines the target architecture, which is provided as input to the compiler. This means that without the need to recompile the compiler, architecture exploration of the VEX ISA is possible with this compiler.
3) *The VEX Simulation System*: The VEX simulator is an architectural-level simulator that uses *compiled simulator* technology to achieve faster execution. It provides a set of POSIX-like *libc* and *libm* libraries (based on the GNU *newlib* libraries), a simple built-in cache simulator (level-1 cache only) and an application program interface (API) that enables other plug-ins used for modeling the memory system.

A VEX software toolchain including the VEX C compiler and the VEX simulator is made freely available by the Hewlett-Packard laboratories [6]. The reason behind choosing the VEX architecture for our project is the scalability and customizability of the VEX ISA and the availability of the free C compiler and simulator, which can be used for architecture exploration.

V. The ρ-VEX VLIW Softcore Processor

[3] presents the design and implementation of a reconfigurable and extensible softcore VLIW processor developed at our group. We implemented a 1-cluster standard configuration of the VEX machine for our processor called ρ-VEX. Figure 1 depicts the organization of our 32-bit, 4-issue VLIW processor implemented on an FPGA. The ρ-VEX processor consists of *fetch*, *decode*, *execute* and *writeback* stages. The fetch unit fetches a VLIW instruction from the attached instruction memory, and splits it into syllables that are passed on to the decode unit. In this stage, register contents used as operands are fetched from the register file. The actual operations take place in either the execute unit, or in one of the parallel

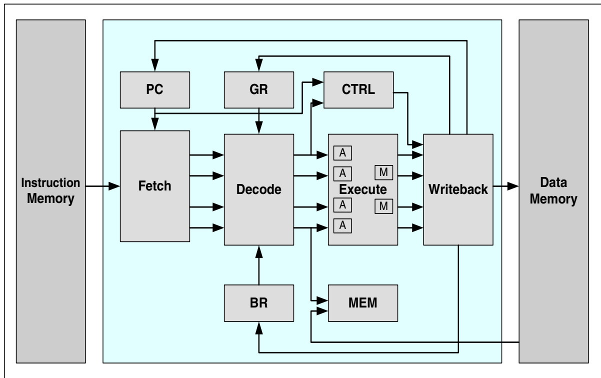

Figure 1. The ρ-VEX VLIW Processor

control (CTRL) or *memory* (MEM) units. *Arithmetic logic unit* (ALU) and *multiplier* (MUL) operations are performed in the execute stage. This stage is implemented in a parameterized manner, allowing the adaptation of the number of ALU and MUL functional units. All jump and branch operations are handled by the CTRL unit, and all data memory load and store operations are handled by the MEM unit. All write activities are performed in the writeback unit to ensure that all targets are written back at the same time. The different write targets could be the *general register* (GR) file, *branch registe*r (BR) file, *data memory* or the *program counter* (PC).

The ρ-VEX processor implements most of the operations of the VEX operations set. Additionally, it supports reconfigurable operations, as the VEX compiler supports the use of custom instructions via pragmas inside the application code. In the current ρ-VEX prototype, it takes only a few lines of VHDL code to add a custom operation to the architecture. One of the 24 available reserved opcodes should be chosen, and a template VHDL function should be extended with the custom functionality. Currently, the following properties of ρ-VEX are parameterized:

- Syllable issue-width
- Number of ALU units
- Number of MUL units
- Number of GR registers (up to 64)
- Number of BR registers (up to 8)
- Types of accessible FUs per syllable
- Width of memory busses

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 [3]. The ROM file is then synthesized with the rest of the processor VHDL design files.

As the target reconfigurable fabric, the Xilinx Virtex-II PRO *XC2VP30* FPGA was chosen, embedded on the XUP V2P development board by Digilent. All experiments were performed on a non-pipelined ρ-VEX system with 32, and 64 general-purpose registers (GR). A data memory implemented using Block RAM (BRAM) was connected to ρ-VEX to store results. The application code was loaded in the instruction memory before synthesis. Table I presents the resource utilization for the ρ-VEX processor.

VI. The VLIW Multiprocessor System

This section presents the design and implementation of our reconfigurable VLIW multiprocessor system. The base processor of this multiprocessor system is the open-source, extensible and reconfigurable softcore VLIW processor ρ-VEX [3]. We designed the ρ-VEX as a non-pipelined processor because of the following reasons:

- a non-pipelined design is very simple compared to a pipelined design
- a non-pipelined design does not require register bypass and pipeline flush logic
- a non-pipelined design requires smaller number of resources and less power due to reduced logic

A. Design and Implementation

We designed our VLIW multiprocessor utilizing the non-pipelined ρ-VEX processor. A dual-processor system is depicted in Figure 2. The multiprocessor has two independent ρ-VEX processors. Each processor has its own instruction and data memory. BRAMs are used as data memory and for instruction memory, we use ROM. This is a simple design and it works well for applications with large data sets. For example, if we need to encrypt 100 kbytes of data according to advanced encryption standard (AES) algorithm, we run the application code on both processors each encrypting 50 kbytes of data and then combine the result. Therefore, we can achieve almost twice the performance as compared to a uni-processor system.

Table II presents the resource utilization for our dual-processor system, where each processor is a 4-issue standard ρ-VEX processor. A universal asynchronous receiver transmitter (UART) module is implemented in the design which is used to transmit the results from the data memories, as well the value of the internal cycle counter from each processor core. A multiprocessor system with any given number of processors can be implemented depending upon the application and the available resources. Parameters of the individual VLIW processors may differ, but the ISA shall remain the same. The ISA can be extended with custom operations and the compiler is able to generate code for the custom hardware units, further enhancing the performance.

B. Resource Sharing in a Multiprocessor System

Table III presents the resource utilization for the individual modules of our base ρ-VEX processor. From Tables I and III, we can observe that the *execute* unit for a 4-issue 64-registers

Table I
Resource Utilization for the ρ-VEX Processor

ρ-VEX	No. of Registers	Slices	Max. Frequency (MHz)
4-issue	32	9894	65.601
4-issue	64	14740	62.293

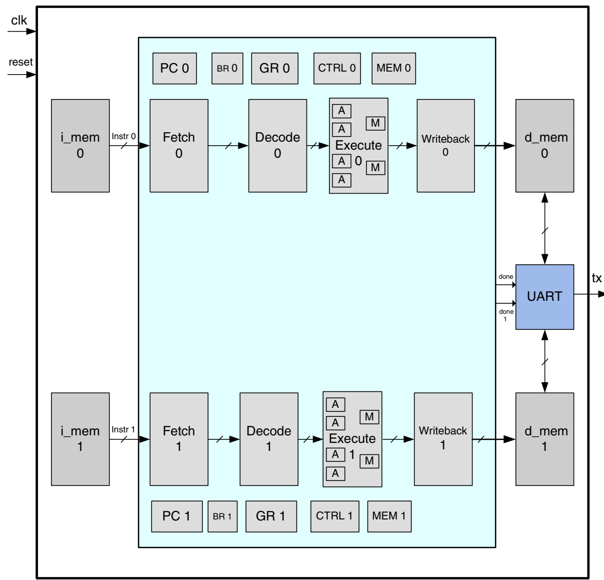

Figure 2. The Dual-Processor System

ρ-VEX consumes 34% slices of the total design. Instead of putting the execute unit in each processor of the multiprocessor system, we developed a scheme to share it.

Because our base processor ρ-VEX is a non-pipelined design, we can share the execute unit between two processors. In a non-pipelined processor a new instruction is only fetched when the older one gets executed and results written back. Therefore, there is a possible time when the execute unit is free. We use that time for another processor, and therefore two processors execute their own threads but they have only one execute unit. This saves resources and power for the extra execute unit. In the current design of ρ-VEX only two cores can share a single execute unit. Therefore, if we need a quad-processor system, we combine two dual-processor systems each with a single execute unit. Figure 3 depicts the design of our resource-shared (execute-unit-shared) dual-processor system.

Table IV presents the resource utilization for our resource-shared (execute-unit-shared) dual-processor system. The last column in the table shows the slices saved due to the execute unit sharing.

C. The Resource Controller

The *Resource Controller* unit is added to the design. This unit takes care of the single *execute* unit. It takes inputs from the *decode* units of both the processor cores and provides these inputs to the execute unit in turn to optimally utilize it. Output from the execute unit is supplied to the *writeback* units of both the cores at the same time. The correct writeback unit writes the results based on the input signals from its corresponding decode unit. Figure 4 depicts the Resource Controller that controls and time shares the single execute unit between the two processors. The Resource Controller utilizes 1606 slices and runs at a maximum frequency of 168.87 MHz. It resolves some conflicts and then multiplexes the signals from the decode units of the two processors. The addition of the Resource Controller improves the critical path of the multiprocessor system by avoiding the logic and connections of the extra execute unit, thereby, increasing the clock frequency of the multiprocessor system.

Table II
RESOURCE UTILIZATION FOR THE DUAL-PROCESSOR SYSTEM

No. of Registers	Slices	Max. Frequency (MHz)
8	13612	61.001
16	15856	62.293
24	17816	61.212
32	19566	62.531
64	28923	62.431

Table III
RESOURCE UTILIZATION FOR THE 4-ISSUE ρ-VEX MODULES

Module	Slices	% slices of the ρ-VEX Processor	Max. Frequency (MHz)
Fetch	97	0.65%	288.37
Decode	296	2.00%	752.30
Execute	5024	34.08%	79.35
Writeback	172	1.16%	528.05
CTRL	68	0.46%	800.48
MEM	146	0.99%	741.86
BR 8	62	0.42%	511.48
GR 64	8594	58.30%	303.01

D. Alternate Design of the ρ-VEX Processor

As can be seen from Table III, the execute unit and the multi-ported register file (GR 64) consumes about 92% of the total ρ-VEX processor slices. In order to reduce the resources for the execute unit, we developed a scheme to share it between two cores. To reduce the slices utilized by the register file, we redesigned our register file. For our 4-issue

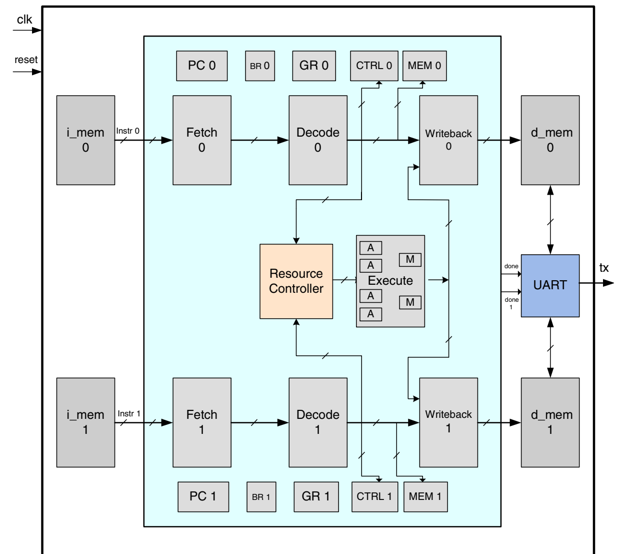

Figure 3. The Resource-Shared Dual-Processor System

Table IV
RESOURCE UTILIZATION FOR THE RESOURCE-SHARED DUAL-PROCESSOR SYSTEM

No. of Registers	Slices	Max. Frequency (MHz)	Slices saved
8	8958	80.500	4654
16	11276	80.266	4580
24	13051	81.177	4765
32	14918	80.491	4648
64	24212	80.324	4711

ρ-VEX processor, we implemented a register file with 4 write and 8 read ports using BRAMs based on [23]. In order to support multiple register ports, the BRAMs are organized into register banks and data is duplicated across various BRAMs within each bank. One of the main limitations of the BRAM-based design of the multi-ported register file is associated with the write ports. This design influences the way registers and instructions are allocated and scheduled [23]. 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. In order to avoid the conflicts for write ports in this design, we developed a *register renaming* technique. This technique is applied at the assembler level after an application source code is compiled with the VEX compiler. The register renaming tool renames registers in the assembly language program generated by the VEX compiler to avoid conflicts for the write ports and generate assembly language program. This final assembly language program is assembled with the ρ-VEX assembler to generate the executable code for the ρ-VEX processor. Since our registers are 32-bit wide, each BRAM can provide up to 512 such registers. Out of these 512 registers, we are using 256, which is a four times *over-dimensioning* for the original 64 registers. This over-dimensioning of the registers does not affect the number of BRAMs in our design and ensures that our register renaming tool always have free registers for renaming. This register renaming tool enables the BRAM-based register file to utilize saving a considerable amount of slices without affecting the overall clock frequency of the processor and cycle count for any application. Table V presents the resource utilization for the BRAM-based register file, and the ρ-VEX processor with the BRAM-based register file.

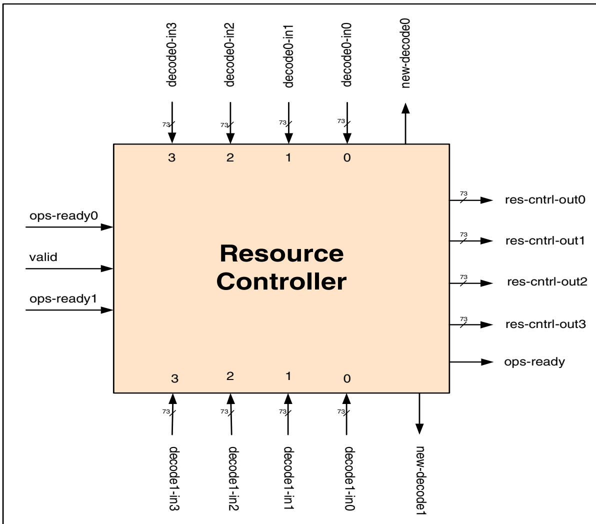

Figure 4. The Resource Controller

Based on our new design of the ρ-VEX processor, we implemented another version of our dual-processor system. Table VI presents the resource utilization for the new multiprocessor designs, both with individual execute units and a shared execute unit. Consequently, we have two alternate designs for the multiprocessor system. If the designer has extra slices, he/she can instantiate the slice-based design. If there are no free slices, the designer can instantiate the BRAM-based design. In terms of performance, there is no difference between these two designs.

E. Application Development Framework

To optimally utilize our multiprocessor system, we present an application development framework. The framework is depicted in Figure 5. An application program written in C is first partitioned based on the application data. It means that half of the application data is placed in the data memory (BRAM) of one processor and the other half in the data memory of the second processor. Therefore, the application data is split into data1 and data2. Based on data1 and data2, the application code is compiled with the VEX compiler and then assembled with the ρ-VEX assembler to generate binaries for the two processors of the system. Custom operation definitions (if any) and the target machine model in the form of *machine model format* (.fmm) [5] are provided to both the compiler and the assembler. Machine model specifies the characteristics of the individual ρ-VEX processor in the multiprocessor system.

VII. EXPERIMENTAL RESULTS

We compared the results on three different VLIW processing systems. The first system was a 4-issue ρ-VEX processor, with 64 registers called as the uni-processor system. The second system was a dual-processor system, using two 4-issue ρ-VEX processors. A single execute unit was shared between the two processors of the dual-processor system. Each processor in the dual-processor system had 64 registers. We implemented these two systems with BRAM-based register files on the Xilinx *XC2VP30* FPGA. The third system was

Table V
RESOURCE UTILIZATION FOR THE BRAM-BASED 4-ISSUE ρ-VEX PROCESSOR

BRAM-based Design	No. of Registers	Slices	BRAMs	Max. Frequency
Register File	up to 512	256	32	822.707 MHz
ρ-VEX Processor	up to 512	6219	32	64.645 MHZ

Table VI
RESOURCE UTILIZATION FOR THE BRAM-BASED DUAL-PROCESSOR SYSTEM

Multiprocessor Design	No. of Registers	Slices	BRAMs	Max. Frequency
No-shared Execute	up to 512	12427	64	64.651 MHz
Shared Execute	up to 512	7759	64	80.500 MHZ

a 2-cluster VLIW system, where each cluster had 4 issue slots and 64 registers. This system was simulated using the VEX simulator. We utilized three applications to test the effectiveness of our dual-processor system. These applications are discussed below.

A. AES Encryption/Decryption Algorithm

Cryptographic applications are becoming a crucial part for modern electronic systems. For example, secure data communication and user authentication are implemented in many systems such as online transactions, bank cash machines, etc. Advanced encryption standard (AES) algorithm is one of the most used encryption algorithms in cryptography. The AES algorithm takes an input data of 128 bits and a key of 128, 196 or 256 bits and produces an encrypted output data of 128 bits. For decryption the same key is used as was used in the encryption process. We implemented a 128 bit key version of the AES algorithm.

We encrypted and decrypted a text of 2048 bytes. For the uni-processor system and the 2-cluster system, the C programs for the encryption and decryption are compiled and assembled and the data memory is initialized with the input data of 2048 bytes. For the dual-processor system, the input data is split into two sets each of 1024 bytes. Each processor is provided its own data set and the same application for encryption/decryption runs on it. Each processor reads 16 bytes (*128* bits) out of 1024 bytes from its local memory and encrypt/decrypt them. Therefore, each processor needed 64 iterations to complete its job. The clock cycles needed to complete the whole application for our three different processing systems are presented in Figure 6. It can be observed from Figure 6 that the dual-processor system completed the execution of the whole application in almost half number of the clock cycles compared to the other two systems.

B. Matrix Multiplication

The second application to test our design is a matrix multiplication program. We created a C program that does a 100-by-100 matrix multiplication. The input matrices are in the data memory of the system. We run the application on our uni-processor system and the 2-cluster system and noted the clock cycles required to carry out the whole multiplication. We then modified the application program for our dual-processor system. We sent one matrix to both the processors and half of the rows of the other matrix to one processor and rest of the half rows to the second processor. The clock cycles required to compute the multiplication for our three different processing systems are presented in Figure 6. It can be observed from Figure 6 that the dual-processor system completed the execution of the matrix multiplication in almost half number of the clock cycles compared to the other two systems.

C. ADPCM Encode/Decode

The third application is the CCITT ADPCM encode/decode process. The application code is provided by the Sun Microsystems. The ADPCM audio encoding/decoding technique is used for the encoding/decoding of audio (voice) in voice over IP (VoIP) telephony applications. For this benchmark, an input audio signal of 175940 bytes is encoded to a 24 kbps G.723 signal using 8-bit, A-law scheme by one processor of the dual-processor system. The other processor of the dual-processor system does the decoding process. It decodes a 24 kbps G.723 encoded signal to 175940 bytes audio signal using 8-bit, A-law scheme. Because normally both the encoder and the decoder are implemented in a single module, this application was a good candidate to test our shared-execute unit multiprocessor system. Instead of employing two individual processors for encoder and decoder, we can save valuable resources by using a resource-shared multiprocessor system. The clock cycles required to compute the encode and decode processes on uni-processor, dual-processor and 2-cluster VEX

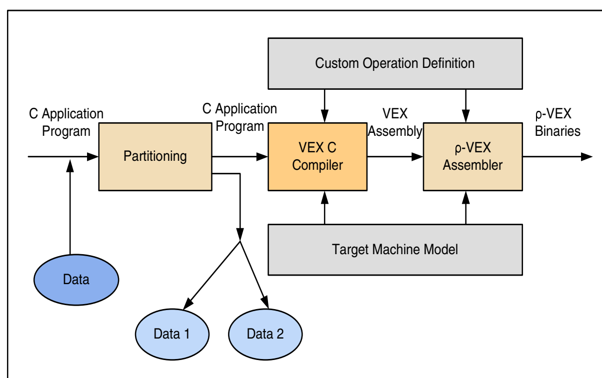

Figure 5. Application Development Framework

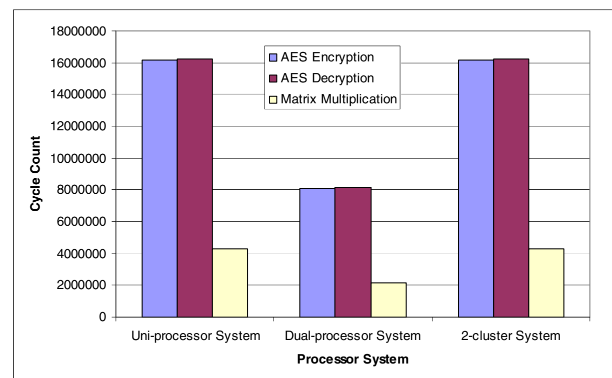

Figure 6. Cycle Count for AES and Matrix Multiplication

systems are presented in Figure 7. The bar *combined* in the figure shows the combined clock cycles needed by each of these systems to run both the encode and the decode processes.

VIII. Conclusions

In this paper, we presented the design and implementation of a reconfigurable softcore very long instruction word (VLIW) multiprocessor system. This multiprocessor design is based on the ρ-VEX processor we designed earlier, and is implemented on an FPGA. Since the ρ-VEX processor is parameterized in different properties, such as the number and type of functional units, supported instructions, memory-bandwidth, and register file size, our multiprocessor is also parameterized in these properties. We evaluated different versions of the ρ-VEX processor based on the slice-based and the BRAM-based register files. We implemented a resource controller unit to time-share a single execute unit between two processors of the multiprocessor system reducing a considerable amount of resources without affecting the performance. A toolchain including a C compiler and a simulator is freely available. Additionally, we presented an application development framework to optimally utilize our design. Any application written in C can be mapped to the multiprocessor system in a simple manner. This VLIW multiprocessor is able to exploit data level and instruction level parallelism inherent in an application and make its execution faster. Both the multiprocessor design and the application development framework are open-source. We utilized AES encryption/decryption, matrix multiplication and ADPCM encode/decode applications to test our dual-processor system. These applications have both data level and instruction level parallelism. The results show that we can achieve two times better performance for our dual-processor system compared to a uni-processor system or a 2-cluster processor system for applications having data level and instruction level parallelism.

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] TriMedia Processor Series. http://www.nxp.com/.

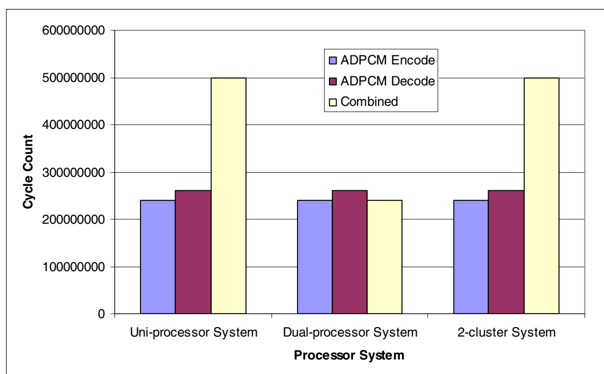

Figure 7. Cycle Count for ADPCM Encode/Decode

[3] 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)*, pp. 369 - 372, 2008.

[4] S. Wong and F. Anjam, "The Delft Reconfigurable VLIW Processor", *In Proceedings of the 17th International Conference on Advanced Computing and Communications (ADCOM 09)*, pp. 244 - 251, 2009.

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

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

[7] C. Iseli and E. Sanchez, "Spyder: A Reconfigurable VLIW Processor using FPGAs", in *Proceedings of the FPGAs for Custom Computing Machines (FCCM 93)*, pp. 17 - 24, 1993.

[8] 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 IEEE International Parallel and Distributed Processing Symposium (IPDPS 03)*, 2003.

[9] 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)*, pp. 619 - 622, 2008.

[10] 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.

[11] 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.

[12] http://www.trimaran.org/.

[13] V. Brost, F. Yang, and M. Paindavoine, "A Modular VLIW Processor", in *IEEE International Symposium on Circuits and Systems (ISCAS 07)*, pp. 3968 - 3971, 2007.

[14] 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.

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

[16] Altera, Inc. December 2007. Tutorial: Creating Multiprocessor Nios II Systems. http://www.altera.com.

[17] A. Hung, W. Bishop and A. Kennings,"Symmetric Multiprocessing on Programmable Chips Made Easy", *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE 05)*, vol. 1, pp. 240 - 245, 2005.

[18] G.G. Mplemenos and I. Papaefstathiou, "MPLEM: An 80-Processor FPGA Based Multiprocessor System", *16th International Symposium on Field-Programmable Custom Computing Machines (FCCM 08)*, pp. 273 - 274, 2008.

[19] S. Xu and H.P. Smith, "A Multi-MicroBlaze Based SOC System: From SystemC Modeling to FPGA Prototyping", *The 19th IEEE/IFIP International Symposium on Rapid System Prototyping (RSP 08)*, pp. 121 - 127, 2008.

[20] K. Ravindran, N. Satish, Y. Jin, and K. Keutzer, "An FPGA-Based Soft Multiprocessor System for IPv4 Packet Forwarding", *Proceedings of the 15th International Conference on Field Programmable Logic and Applications (FPL 05)*, pp. 487 - 492, 2005.

[21] M. Hubner, K. Paulsson, and J. Becker, "Parallel and Flexible Multiprocessor System-on-Chip for Adaptive Automotive Applications Based on Xilinx MicroBlaze Soft-Cores", *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 05)*, pp. 149a - 149a, 2005.

[22] P. Huerta, J. Castillo, J.I. Martinez, and V. Lopez, "A MicroBlaze Based Multiprocessor SoC", *WSEAS Transactions on Circuits and Systems*, vol. 4, no. 5, pp. 423 - 430, 2005.

[23] 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.