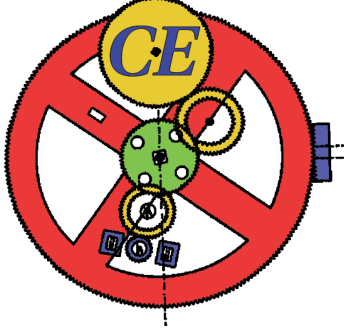


Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2014

MSc THESIS

Design-time and Run-time Reconfigurable Clustered ρ-VEX VLIW Softcore Processor

Muez Berhane Reda

Abstract

The ρ-VEX processor is a parameterized reconfigurable Very Large Instruction Word (VLIW) softcore processor. It can be reconfigured in the issue-width, number and type of functional units (FUs), width of memory buses and number of registers in the multi-ported register file. The current design of the ρ-VEX processor supports single cluster processor organization. The design also provides run-time dynamic reconfigurability between different processor architectures. As the issue-width of the processor increases, the number of read and write ports from the FUs to the register file increases which enlarges its area utilization. This increase in the number of read and write ports is not scalable with the interconnect wires available in the current IC technology. From literature, we know that clustering the FUs of the processor and splitting up the register file into a smaller subsets significantly reduces the area overhead and power consumption.

In this thesis, we have developed all the necessary hardware and software components to enable the design-time and run-time reconfigurable ρ-VEX processors to support clustered organization. Those development are the design and implementation of inter-cluster communication FUs, inter-cluster path and local register file per cluster and the adaptation of the compiler toolchain. As an inter-cluster communication model (ICC), *copy operation* and *dedicated issue slot* ICC model are implemented. The cycle count and total operations of different benchmark applications are measured and analyzed on the clustered organization of ρ-VEX processors. The cycle count for most of the benchmarks is higher in clustered organization except for applications with high instruction level parallelism such as *matrix* and *adpcm*. A speedup of 3.04× is achieved by *matrix* benchmark. On the other hand, an increase in code size of the benchmark applications is measured for the clustered processor by a maximum of 38%. The area utilization of the 4-issue and 8-issue design-time reconfigurable ρ-VEX processors are significantly reduced by up to 74% by clustering them into two clusters. In addition, a speedup of 1.55× is obtained on the clock frequency of the processor. Similarly, the run-time reconfigurable clustered ρ-VEX processor occupies 61.3% less area, consumes up to 41.6% less dynamic power than the single clustered processor and has a reduced energy delay product (EDP).

CE-MS-2014-14

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Design-time and Run-time Reconfigurable Clustered ρ-VEX VLIW Softcore Processor

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Muez Berhane Reda
born in Mekelle, Ethiopia

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Design-time and Run-time Reconfigurable Clustered ρ-VEX VLIW Softcore Processor

by Muez Berhane Reda

Abstract

The ρ-VEX processor is a parameterized reconfigurable Very Large Instruction Word (VLIW) softcore processor. It can be reconfigured in the issue-width, number and type of functional units (FUs), width of memory buses and number of registers in the multi-ported register file. The current design of the ρ-VEX processor supports single cluster processor organization. The design also provides run-time dynamic reconfigurability between different processor architectures. As the issue-width of the processor increases, the number of read and write ports from the FUs to the register file increases which enlarges its area utilization. This increase in the number of read and write ports is not scalable with the interconnect wires available in the current IC technology. From literature, we know that clustering the FUs of the processor and splitting up the register file into a smaller subsets significantly reduces the area overhead and power consumption.

In this thesis, we have developed all the necessary hardware and software components to enable the design-time and run-time reconfigurable ρ-VEX processors to support clustered organization. Those development are the design and implementation of inter-cluster communication FUs, inter-cluster path and local register file per cluster and the adaptation of the compiler toolchain. As an inter-cluster communication model (ICC), *copy operation* and *dedicated issue slot* ICC model are implemented. The cycle count and total operations of different benchmark applications are measured and analyzed on the clustered organization of ρ-VEX processors. The cycle count for most of the benchmarks is higher in clustered organization except for applications with high instruction level parallelism such as *matrix* and *adpcm*. A speedup of 3.04× is achieved by *matrix* benchmark. On the other hand, an increase in code size of the benchmark applications is measured for the clustered processor by a maximum of 38%. The area utilization of the 4-issue and 8-issue design-time reconfigurable ρ-VEX processors are significantly reduced by up to 74% by clustering them into two clusters. In addition, a speedup of 1.55× is obtained on the clock frequency of the processor. Similarly, the run-time reconfigurable clustered ρ-VEX processor occupies 61.3% less area, consumes up to 41.6% less dynamic power than the single clustered processor and has a reduced energy delay product (EDP).

Laboratory : Computer Engineering
Codenumber : CE-MS-2014-14

Committee Members :

Advisor: Stephan Wong, CE, TU Delft

Chairperson: Stephan Wong, CE, TU Delft

Member:	Zaid Al-Ars, CE, TU Delft
Member:	Rene Van Lauken, CAS, TU Delft

Dedicated to my dearest family

Contents

List of Figures viii

List of Tables ix

List of Acronyms xii

Acknowledgements xiii

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 Problem Statement and Project Goals . . . 4
- 1.3 Methodology . . . 4
- 1.4 Overview . . . 5

2 Background 7
- 2.1 The VEX System . . . 7
  - 2.1.1 The VEX Instruction Set Architecture . . . 8
  - 2.1.2 The VEX C Compiler . . . 9
- 2.2 Design-time Reconfigurable ρ-VEX VLIW Processor . . . 10
- 2.3 Run-time Reconfigurable ρ-VEX VLIW Processor . . . 12
- 2.4 Related Work . . . 15
  - 2.4.1 Copy Operation ICC model . . . 16
  - 2.4.2 Dedicated Issue Slot ICC model . . . 17
  - 2.4.3 Extended operand ICC model . . . 17
  - 2.4.4 Extended results ICC model . . . 18
  - 2.4.5 Broadcasting ICC model . . . 18
- 2.5 Conclusion . . . 19

3 Design-time Reconfigurable Clustered ρ-VEX Processor 21
- 3.1 VEX ISA and Compiler Support for Clustered VLIW Processors . . . 21
- 3.2 Compiler Toolchain Adaptation for Clustered Architectures . . . 25
  - 3.2.1 Modified GNU Binutils for Clustered ρ-VEX Architectures . . . 25
  - 3.2.2 Inter-cluster Copy Operations Syllable . . . 26
  - 3.2.3 Conditional Branch Operations Syllable . . . 27
  - 3.2.4 Borrowing Scheme for Clustered ρ-VEX Processor . . . 29
- 3.3 Design of Clustered ρ-VEX VLIW processor . . . 31
  - 3.3.1 Decoder unit . . . 31

3.3.2 Copy Operation Inter-cluster Communication model 32
3.3.3 Forwarding Logic 34
3.4 HP VEX Compiler for Clustered Architectures: BUG and its FIX 35
3.5 Clustered ρ-VEX Processor with Dedicated Issue Slot ICC Model 38
3.6 Conclusion 40

4 Run-time Adaptable Clustered ρ-VEX Processor **41**
4.1 Design of Clustered Run-time Reconfigurable ρ-VEX VLIW Processor . . 41
4.2 The (4 + 4)-way Clustered ρ-VEX VLIW Processor 44
4.3 The Dual-core (2 + 2)-way Clustered ρ-VEX VLIW Processor 46
4.4 Run-time Dynamic Reconfiguration of the 2-4-8-issue VLIW processor . . 48
4.5 Conclusion 49

5 Results **51**
5.1 Experimental Setup 51
5.1.1 Benchmark applications 51
5.1.2 Hardware Platform and Software Tools 51
5.2 Clustered ρ-VEX processor with Copy Operation ICC Model 52
5.2.1 Performance Analysis: Maximum Clock Frequency 53
5.2.2 Total Cycle Count and Speedup 53
5.2.3 Operations Count 55
5.2.4 Power Consumption and Area Overhead Analysis 58
5.2.5 Energy Delay Product (EDP) 62
5.3 Clustered ρ-VEX Processor with Dedicated Issue Slot ICC Model 63
5.3.1 Total Cycle Count and Speedup 63
5.3.2 Operations Count 64
5.3.3 Power Consumption and Area Overhead Analysis 64
5.4 Conclusion 65

6 Conclusions **67**
6.1 Summary 67
6.2 Main Contributions 69
6.3 Future Work 69

Bibliography **74**

A Inter-Cluster Communication Functional Unit **75**

B Inter-Cluster Path **79**

List of Figures

1.1 Datapath widths in an 8-issue ρ-VEX VLIW architecture. 2
1.2 Datapath widths in a two cluster, 8-issue VEX VLIW architecture. . . . 3

2.1 The ρ-VEX VLIW processor pipeline organization. 11
2.2 The four 2-issue ρ-VEX cores. I-Mem and D-Mem refer to the instruction and data memories of the processor. 12
2.3 The two 4-issue ρ-VEX cores. 13
2.4 One 8-issue ρ-VEX core. 14

3.1 High level structure of a VEX multi-cluster organization. 22
3.2 Output of disassembler for a (2 + 2)-way clustered ρ-VEX processor. . . 26
3.3 Clustered ρ-VEX architecture with inter-cluster path and local register files. 32
3.4 Copy operation ICC model in a (2 + 2)-way clustered ρ-VEX processor. 33
3.5 Dedicated Issue Slot Inter-Cluster communication Model. 39
3.6 Instruction Bundle Format for Dedicated Issue Slot Inter-Cluster Communication Model. 39
3.7 Issue slot scheduling in 4-issue ρ-VEX processors with dedicated issue slot ICC model. 40

4.1 The 2-4-8-issue run-time adaptable clustered ρ-VEX processor. 42
4.2 The 8-issue run-time adaptable clustered ρ-VEX core. 45
4.3 The address generation unit for GR register file in (4 + 4)-way clustered run-time adaptable ρ-VEX core. 46
4.4 Two 4-issue run-time adaptable clustered ρ-VEX cores. 47
4.5 The address generation unit for GR register file in (2 + 2)-way run-time adaptable clustered ρ-VEX cores. 48
4.6 Compilation flow in (2 + 2)-way run-time adaptable clustered ρ-VEX cores. 49

5.1 Speedup for *powerstone* benchmark applications on (2 + 2)-way clustered ρ-VEX processor and 4-issue with 2 load/store unit ρ-VEX processor compared to the baseline 4-issue ρ-VEX processor. 54
5.2 Speedup for *powerstone* benchmark applications in (4 + 4)-way clustered ρ-VEX processor and 8-issue with 2 load/store unit ρ-VEX processor compared to the baseline 8-issue ρ-VEX processor. 55
5.3 Percentile of Operations for *powerstone* benchmark applications in (2 + 2)-way ρ-VEX processor. 57
5.4 Percentile of Operations for *powerstone* benchmark applications in (4 + 4)-way ρ-VEX processor. 58

5.5 Dynamic and static power consumption of design-time reconfigurable 4- and 8-issue ρ-VEX processors, (2 + 2)-way clustered (with copy operation and dedicated issue slot ICC model), and (4 + 4)-way clustered ρ-VEX processors. 59
5.6 Dynamic and static power consumption of clustered and single cluster organizations of the run-time adaptable ρ-VEX processors. 61
5.7 EDP of (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processors normalized to their corresponding single cluster processors. 62
5.8 Speedup for *powerstone* benchmark applications in (2 + 2)-way clustered ρ-VEX processor with dedicated issue ICC model compared to the baseline 4-issue ρ-VEX processor. 63
5.9 Percentile of Operations for *powerstone* benchmark applications in (2 + 2)-way clustered ρ-VEX processor with dedicated issue slot ICC model. 64

List of Tables

2.1 VEX design parameters. 8
2.2 The *issue_ctrl* control signal and the mode of operation of the run-time reconfigurable ρ-VEX processor core. 15

3.1 The machine configuration model used for the (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processor. 23
3.2 Configuration setting used for the (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processors. 25
3.3 FUs placement, Mapping and Borrowing Scheme for 2 cluster 4-issue width clustered ρ-VEX processor. ALU, MUL, MEM, CT , ICC FU and S_F represent ALU unit, MUL unit, MEM unit, Control/Branch unit, inter-cluster communication functional unit and Syllable for *follow* operation respectively. The symbols S and L are for Short and Long immediate operand. 30
3.4 Borrow settings for the (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processor. 30
3.5 SEND operation. Inter-cluster write data path. 33
3.6 RECV operation. Inter-cluster read data path. 33
3.7 Algorithm used to fix the bug of the VEX C compiler. 37
3.8 Number of occurrences of the bugs fixed for different benchmark applications compiled for a (2 + 2)-way clustered ρ-VEX processor architecture. 38

5.1 Description of benchmark applications used for measurement. 52
5.2 Total number of operations and rate of increase on 4- and 8-issue clustered ρ-VEX processor for *powerstone* benchmark applications. 56
5.3 Area Utilization of 4-issue, (2 + 2)-way clustered ρ-VEX processor with default (copy operation) and dedicated issue slot ICC model on Virtex-6 FPGA. 60
5.4 Area Utilization on Virtex-6 FPGA of the (4 + 4)-way clustered and 8-issue ρ-VEX processor. 60
5.5 Area Utilization of clustered and single cluster organizations of the run-time adaptable ρ-VEX processor on Virtex-6 FPGA. 61

List of Acronyms

AES Advanced Encryption Standard.

AGU Address Generation Unit.

ALU Arithmetic and Logic Unit.

ASIC Application-Specific Integrated Circuit.

BOPS Billions of Operations Per Second. VLIW processor.

BRAM Block Random Access Memory. Special FPGA resource that can be used to instantiate memory arrays.

CFG Configuration.

CPU Central Processing Unit.

CTRL Control Unit. Branch Unit.

DEL Delay. Used to specify the latency of FUs in the machine configuration model.

DDG Data Dependency Graph.

DLP Data-Level Parallelism.

D-Cache Data Cache.

EDP Energy Delay Product.

ERA Embedded Reconfigurable Architectures. EU-funded project based on the ρ-VEX processor core.

FPGA Field Programmable Gate Array. A type of reconfigurable hardware.

FU Functional Unit.

GNU GNUs Not UNIX.

GR, BR, LR General Register, Branch Register, Link Register.

HP Hawlett Packard.

IC Integrated Circuit.

ICC Inter-Cluster Communication.

ILP Instruction-level parallelism.

ISA Instruction Set Architecture.

ISE Integrated Software Environment.

I-cache Instruction Cache.

LLVM Low Level Virtual Machine. Open source compiler.

LUT LookUp Table.

MAJC Microprocessor Architecture for Java Computing.

MEM Memory Unit.

MIMD Multiple Instruction Multiple Data.

MUL Multiplier Unit.

NOP No-op, no-operation. VEX operation that does nothing.

PC Program Counter. Register containing the address of the instruction that the processor will execute next.

RAM Random Access Memory.

RES Resource. Used to specify CPU resources in the machine configuration model.

RISC Reduced Instruction Set Computing.

SIMD Single Instruction Multiple Data.

UART Universal Asynchronous Receiver/Transmitter. Peripheral device used for serial communication.

VEX VLIW Example.

VHDL VHSIC (Very High-Speed Integrated Circuit) Hardware Description Language.

VLIW Very Large Instruction Word.

Acknowledgements

Foremost, I would like to express my sincere gratitude to a number of people that helped me during this master thesis project. I would like to thank my thesis advisor Stephan Wong for giving me the opportunity to work on this interesting field of study, for the continuous support he provided me, and for revising the drafts of my thesis and suggesting valuable comments. I would like to thank Zaid Al-Ars for motivating me and proofreading my draft.

I would like to thank Ir. Anthony Brandon and Ir. Joost Hoozemans for supporting me with the technical aspect of the project.

I am grateful to *Justus and Louise Van Effen foundation* for the full scholarship they offered me for my master program.

Last but not the least, I would like to thank my dearest family: my parents Yeshi Assefa and Berhane Reda, for encouraging and pushing me to aim for the highest level throughout my whole life.

Muez Berhane Reda
Delft, The Netherlands
August 19, 2014

1 Introduction

This thesis presents a MSc project that adds clustered organization support to the design-time and run-time reconfigurable ρ-VEX VLIW softcore processors. In Section 1.1 introduces basic concepts and context to understand the thesis. The problem statement and project goals are defined in Section 1.2. The methodology followed and an overview of the thesis are presented in the Sections 1.3 and 1.4.

1.1 Context

The advent of scalar processors in the early 1960's was a new era to the development of embedded and personal computers. Scalar processors have a single execution unit that can only execute one instruction and operate on a single data per cycle, which has a performance disadvantage. This led to a design of vector processors that execute single instruction and operate on a multiple data in a cycle, Single Instruction Multiple Data (SIMD), hence the name *vector*. The compiler for vector processors handles generation of vector operations by exploiting Data-Level Parallelism (DLP) in a program. Instead of having a processor that uses a single operation to operate on multiple data, multiple execution units connected to a register file and data memory can be designed to execute multiple operations in parallel at a given cycle, Multiple Instruction Multiple Data (MIMD). One type of processor with this philosophy is a *Very Large Instruction Word (VLIW) processor*. As its name suggests, a VLIW processor has a very large instruction word and uses multiple execution units called *pipelanes* for issuing multiple operations in one clock cycle to exploit instruction level parallelism (ILP) in a program.

VLIW softcore processor is a hardware description language (HDL) model of VLIW processor and can be implemented on two Integrated Circuit (IC) technologies: Application-Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA). Processors that are implemented in an ASIC have a fixed issue-width which is set at fabrication time. Hence, hardware design programmed on an ASIC can not be reprogrammed. On the other hand, the FPGA is an IC that can reprogrammed with different hardware designs after manufacturing. This flexibility of FPGAs opens up a new dimension in the area of VLIW softcore processors. The ρ-VEX processor is a design-time reconfigurable open source pipelined VLIW softcore processor which is implemented on the *Xilinx Virtex 6 FPGA*. The ρ-VEX processor implements the VEX (VLIW EXample) Instruction Set Architecture (ISA) and uses VEX compiler toolchain developed by Hewlett-Packard (HP) and STMicroelectronics. In addition to design-time reconfiguration, the design of the ρ-VEX softcore processor is provides run-time dynamic reconfiguration [1, 2] between different processor architectures. The ρ-VEX processor

is reconfigurable in the issue-width, number and type of the different functional units (FUs), size and number of registers in the multiported register file, number and type of accessible FUs per execution unit, width of memory buses, and latencies of the FUs. A *register file* is a set of registers in a processor that are architecturally visible data holders.

The ρ-VEX processor has a multiported register file that can be reconfigured in the number of read/write ports and number of registers. Each pipelane in the ρ-VEX VLIW processor requires two read and one write port to the register file. This implies that an 8-issue VEX VLIW processor must read 16 values (two source operands per pipelane) and write 8 values (one destination register from each pipelane), 16R8W, every cycle when operating at its maximum performance. Figure 1.1 depicts the data path width for an 8-issue ρ-VEX architecture.

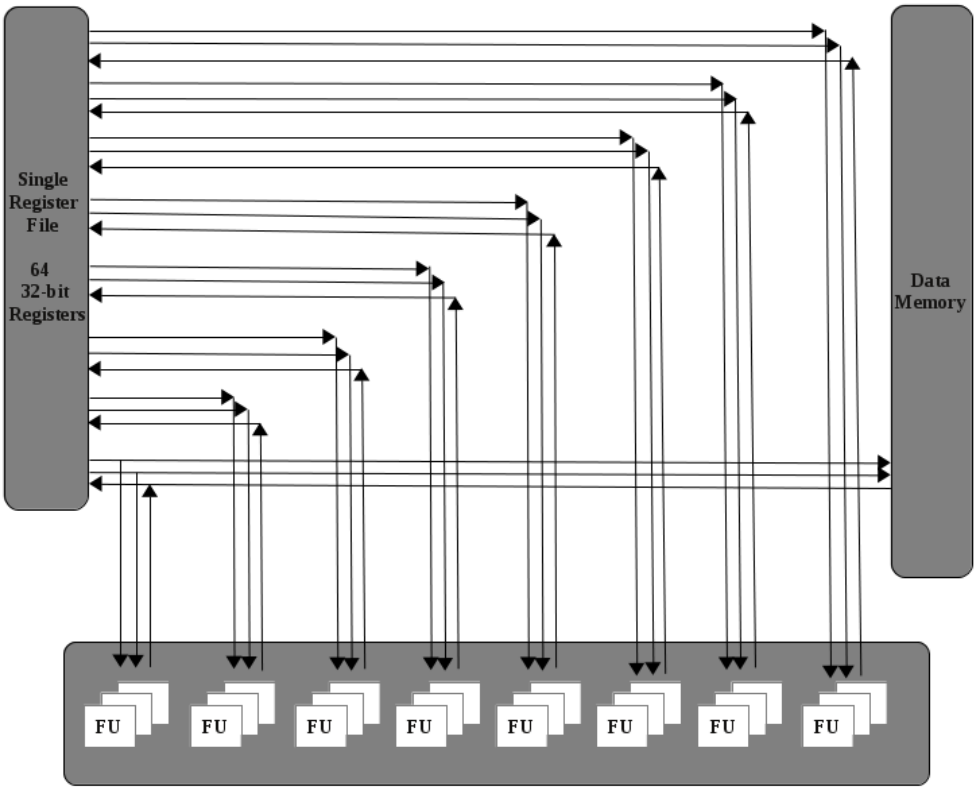


Figure 1.1: Datapath widths in an 8-issue ρ-VEX VLIW architecture.

As the issue-width of the processor increases, the number of read and write ports to the register file increase and the area of a register file grows with the square of the number of read and write ports. Each port needs a new connection from the register file to the functional units in the pipelanes. The design of the register file is limited by routing and not by transistor density [5]. Unfortunately, although the number of transistors incorporated in an IC approximately doubles every two years as predicted by

the *Moore's law* [3, 4], interconnect wires do not scale well with it. Read access time in of a register file grows approximately linearly with the number of ports. The forwarding logic of the register files is largely wire-dominated and it grows with the product of the number of read and write ports.

Clustering is a technique found in literature to reduce area and power overhead of a register file and a VLIW processor. The register file is split into a smaller register files with less read and write ports and those register files are connected to a subset of the functional units. Each pair of register file and subset of the execution functional units is called a *cluster*. Clustering reduces the number of read and write ports to the register files from 8R4W to 4R2W for the four-issue VLIW processor discussed above by using 2 clusters. Figure 1.2 depicts the data path for an 8-issue 2 cluster VEX VLIW processor. The local register files have 8R/4W ports. In a clustered VEX architecture, each cluster has its own load/store unit.

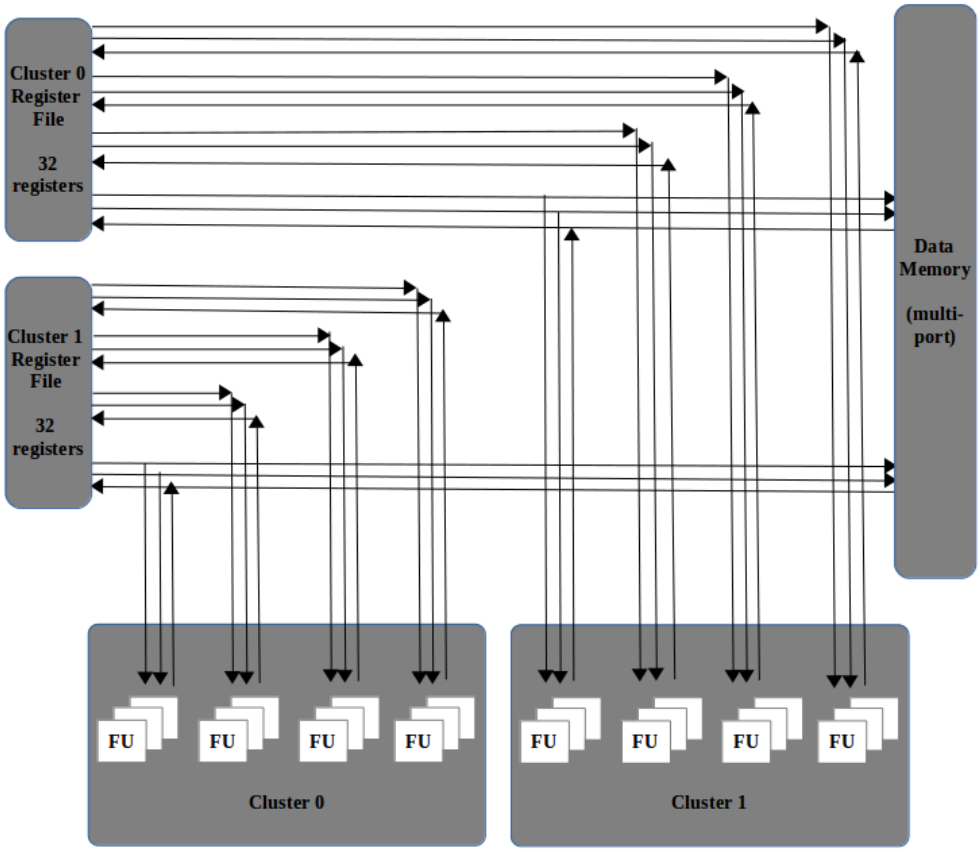


Figure 1.2: Datapath widths in a two cluster, 8-issue VEX VLIW architecture.

Clustering also reduces the amount of pressure [6] on the register resources for a VLIW compiler that is caused by the usage of agressive compiler optimizations that lead to high-level code transformations such as loop unrolling and procedure inlining.

VLIW processors depend on the compiler to exploit the ILP and generate efficiently scheduled code. It suffices to say that the scheduling algorithm used by the compiler is the key component for the generation of an optimized code that can run faster. The compiler takes care of resolving data hazards and scheduling of operations into instruction bundles[1]. Clustering adds another constraint to the scheduler, which is the generation of scheduled code with balanced use of registers on the clusters, optimal division of code between the clusters and minimum inter-cluster data transfers.

1.2 Problem Statement and Project Goals

The problem statements are:

- What are the effects of introducing clustering to the design-time reconfigurable ρ-VEX VLIW processor on its area utilization, performance, and Energy Delay Product (EDP).
- How to provide support for clustered organizations in the 2-4-8-issue run-time reconfigurable ρ-VEX VLIW processor by using the existing register file implementation?

Providing clustered organization support to the ρ-VEX VLIW processor requires designing and implementing both hardware and software components. The design-time and run-time reconfigurable ρ-VEX processor cores and the compiler toolchain has to be adapted to support clustering. The main goal of this thesis are:

1. Design and implement the compiler toolchain that provides generation of binaries for clustered organizations.
2. Adapt the hardware design of the design-time and run-time reconfigurable ρ-VEX processors to support clustered organizations.

1.3 Methodology

The methodology followed to add clustered organization support to the design-time and run-time reconfigurable ρ-VEX VLIW processor are:

1. Develop a compiler toolchain support for clustered VLIW organizations. The VEX compiler generates code for clustered VEX architectures, but the *binutils* has to be modified to support generation of binaries for clustered organizations.
2. Understand the architecture of the design-time and run-time reconfigurable ρ-VEX processors and analyze how clustered organization could be introduced. Design and implement a mechanism for communication of data between the register files of different clusters.

[1] An *instruction bundle* is a set of operations to be executed in a VLIW processor at a given cycle.

3. Develop a technique to enable the run-time reconfigurable ρ-VEX processor to switch between single cluster and clustered organizations at run time.

4. Investigation of performance in terms of cycle count of benchmark applications, area overhead, and EDP of the clustered ρ-VEX processor compared to the single cluster organization. Analysis on the number and type of operations are performed for different benchmark applications.

1.4 Overview

The remainder of the thesis is organized as follows.

Chapter 2 provides a brief introduction to concepts, related work, inter-cluster communication models and compiler scheduling algorithms. The baseline processors, the design-time and run-time reconfigurable ρ-VEX processors, are introduced.

Chapter 3 explains the design-time reconfigurable clustered ρ-VEX processor, its design and implementation. The adaptations of the compiler toolchain, and the processor core are presented. A bug found on the VEX C compiler and the algorithm used to fix it are discussed.

Chapter 4 presents the design and implementation of the run-time reconfigurable clustered ρ-VEX processor. New components introduced such as the inter-cluster path and inter-cluster communication FU and the modification made to different parts of the baseline ρ-VEX processor are discussed.

In Chapter 5, experimental results performed such as comparison of area overhead, total cycle count and power consumption of the clustered design with the ρ-VEX processor using the *powestone* [7] benchmark applications.

Chapter 6 summarizes the previous chapters, lists the main contributions of the thesis project, discusses future work and concludes the thesis.

2 Background

This chapter provides a brief explanation of the basic background concepts and related work to help the reader understand the thesis project. The VEX system is discussed in Section 2.1; the single cluster organization design-time and run-time reconfigurable ρ-VEX VLIW processors that are used as a baseline processor for this thesis are described in Sections 2.2 and 2.3 respectively. In Section 2.4, the pros and cons of different inter-cluster communication models and related work on clustered VLIW architectures are presented.

2.1 The VEX System

The VEX, which stands for VLIW Example, is a system developed by Hewlett-Packard (HP) to help in the design of VLIW processors that offer instruction level parallelism (ILP) that are completely visible to the compiler and in the low level assembly program. VEX includes three basic components [5]:

1. **The VEX Instruction Set Architecture**
   The ISA defines 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 the HP/ST Lx (ST200) family of VLIW embedded cores. The ISA is scalable as it provides the ability to change the number of clusters, number of registers, number and type of functional units and their latencies. Customizability of the VEX ISA enables the users to define new special purpose (custom) operations.

2. **The VEX C Compiler**
   It is derived from the Lx/ST200 C compiler. The *VEX C compiler* [8] is an ISO/C89 compiler that uses *trace scheduling* as its scheduling engine. Trace scheduling will be discussed in Section 2.1.2. A simple machine configuration model defining the target architecture is given as an input file to the VEX C compiler. In this configuration model, different parameters such as the number of clusters, available execution units in each cluster, issue-width of the architecture and clusters, and latencies of the operation can be specified.

3. **The VEX Simulation System**
   The VEX simulator is an architectural-level simulator that uses compiled simulator technology to achieve a speed as fast as million instruction per second (MIPS). It includes a complete 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 to model the memory system.

2.1.1 The VEX Instruction Set Architecture

VEX models a scalable technology platform for embedded reconfigurable VLIW processors that can be configured in the issue-width, number and type of functional unit in each issue slot, and size of the register files. The compiler is responsible for resolving dependencies and generating the instruction of a program.

Flexibility in scheduling of multiple concurrent operation is provided by VEX. Some of features include [5]:

- Parallel execution unit (interger ALUs and multipliers)
- Parallel memory pipelines, including multiported data memory
- A large multiported register set that are architecturally visible
- Partial predication with select operations
- Branch registers with efficient branch architecture
- Long immediate operands encoding within the same instruction bundle

The most basic unit of execution in VEX is an *operation* and an encoded operation is called a *syllable*. An *instruction* is a set of operations scheduled by the compiler to be executed in VLIW mode in a single cycle. Scalar processor has one operation per instruction because its issue-width is one.

The design parameters of VEX VLIW processor architecture [9] are given in the Table 2.1.

Processor Resource	**Design Parameters**
Functional Units	Number of Functional units, type, supported instructions, degree of pipelining
Register File	Register size, register file size, number of read ports, number of write ports
load/store unit	Number of memory ports, memory latency, cache size, line unit
Interconnection network	Number and width of buses, forwarding connections between units

Table 2.1: VEX design parameters.

A default single cluster VEX architecture has an issue-width of 4, four 32-bit integer architecture and logic units (ALUs), one load/store Unit, One branch unit, 16x32 multipliers (MULs), 64 32-bit general-purpose registers (GRs) and an 8 1-bit branch registers (BRs).

2.1.2 The VEX C Compiler

The VEX compiler takes a C program, the machine configuration model of the VLIW architecture and compiler flags (number of clusters and different optimization levels), uses trace scheduling algorithm and generates an VEX assembly instructions for the provided architecture. The syntax of a VEX `add` operation is given as follows:

```
c0 add $r0.4 = $r0.2, $r0.9
```

The `c0` denotes the cluster identifier where the operation will be executed. In a single cluster organization, this section for all the operation is the same, `c0`. The mnemonic of the operation is given next to the cluster identifier, in this case an `add` operation. The destination operand and the source operands are specified separated by `=` sign. In the VEX assembly provided above, destination operand is register number 4 of cluster 0, and the source operands are register number 2 and 9 of cluster 0. The instruction separator `;;` is used to indicate the end of an instruction bundle. During code generation phase, the compiler inserts this instruction separator after each VLIW instruction to signify the end of one instruction bundle and beginning of the next one.

The VEX C compiler uses *trace scheduling* algorithm [10, 11] as its global scheduling engine. Trace scheduling relies on estimates of probability of program jump direction to move around operations from one basic block to another to optimize the generated code. A *trace* is a loop free, linear selection of code. The algorithm [12] is described as follows:

1. The scheduler selects a trace from the code that is not yet scheduled by using profiling information. It applies different code optimization techniques like loop unrolling and inlining to widen the trace search space for the selection of large trace.

2. By treating the traces selected in previous step as if it were a single basic block, the compiler schedules the code. It builds a data dependency graph (DDG) containing all the operations on the trace, including conditional jumps but conserving the order of branch operations.

3. In step 2, some operations that have been scheduled after a conditional jump that they used to precede originally, are duplicated before the off-trace target of the branch. Care should be taken when doing a lot of code motion.

4. The above steps are performed repeatedly until the whole flow graph is covered and no unscheduled operations remain.

Trace scheduling-2, an extension of trace scheduling, is nonlinear scheduling algorithm in a sense that it allows code above a conditional jump from both sides at the same time [12]. It allows code motion decisions to be made at different stages of the scheduling by using an expected value function called *speculative yield*.

In general, there are other compiler scheduling algorithms that target different clustered VLIW architectures. *Modulo Scheduling* algorithm is a scheduling technique that is based on software pipelining principle. It issues an iteration of a loop before the previous iteration has finished its execution. Sanchez et al. [13] proposed a clustered VLIW architecture with copy operation ICC model and a modulo scheduling scheme which takes into account the available the registers, cache memory and inter-cluster communications. The scheduler implements performs memory locality analysis using *Cache Miss Equations* [14] to minimize inter-cluster communications. Porpodas et al. [15] proposed a unified cluster-assignment scheduling and communication reuse (CAeSaR) for clustered VLIW processors. The algorithm assumes that inter-cluster communication of the architecture is using explicit inter-cluster copy operations. They combined the cluster-assignment, cluster scheduling and inter-cluster communication reuse algorithms into one unified algorithm and optimized it. Krishnan et al. [16] combined cluster assignment, register allocation ad instruction scheduling phases into a one phase. It considers the resource constraints while scheduling. In [17], Porpodas et al. designed inter-cluster latency unified scheduling. Fast frequency selection for heterogeneous clustered VLIW architectures that can operate at different clock frequencies, unified instruction scheduling and cluster assignment for such architectures are discussed in [18]. Those scheduling algorithms are efficient in reducing code size and scheduling for multi-cluster VLIW architectures, nevertheless, they do not provide a complete compiler toolchain and they do not target VEX architecture. On the other hand, the scheduling algorithm of the VEX C compiler could not be modified because the compiler is a closed source.

In the next sections, the baseline VLIW processors for this thesis, the design-time and run-time reconfigurable ρ-VEX softcore processors that implemented the VEX single cluster VLIW architecture are discussed.

2.2 Design-time Reconfigurable ρ-VEX VLIW Processor

The ρ-VEX processor [9] was an embedded reconfigurable architecture (ERA) [19] project funded by European Union (EU) that implemented the VEX architecture in a reconfigurable hardware. It is a single cluster organization design time parameterizable and reconfigurable VLIW softcore processor with load/store Harvard architecture[1]. The issue-width, number, type and placement (in which pipelane[2]) of different functional units, number of registers in the multi-ported register file and size of the instruction and data memories can be reconfigured at design time. Fig 2.1 shows the pipeline organization of a 4-issue ρ-VEX VLIW processor [20]. The pipeline has five stages: instruction fetch stage, instruction decode stage, two execute stages and write-back/memory stage. The 4-issue ρ-VEX processor has four arithmetic and logic (ALU) units, 2 multiplier

[1]Harvard architecture is a computer architecture with a physicaly separate instruction and data memories.

[2]Pipelane is a terminology used in the ρ-VEX processor to describe a lane containing a set of functional units capable of executing an operation. The number of pipelanes is equal to the issue-width of the processor.

units (MUL), a branch unit (CTRL), a memory or load/store unit (MEM), 64 32-bit general purpose registers and branch register file with 8 1-bit registers.

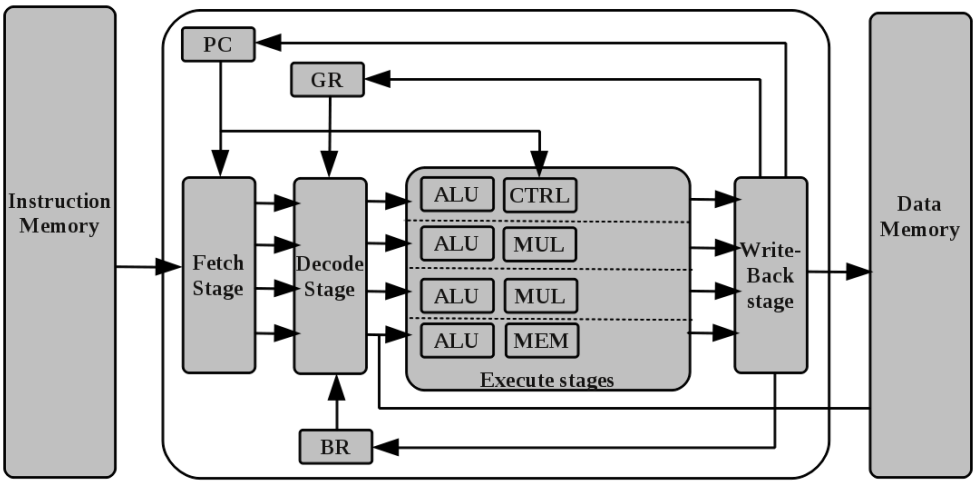


Figure 2.1: The ρ-VEX VLIW processor pipeline organization.

Every cycle, an issue-width amount of operation are retrieved from the instruction memory by the fetch unit and passed to the decoder unit. The opcode, source and destination operands are extracted from each operation in the decode stage of the pipeline. The register numbers of the source operand registers (general purpose, branch and link) are sent to their respective register files and their contents are read. The forwarding logic makes sure that the result of the execution and write-back stages of the pipeline, which are not written back to the register files, are passed to the functional units that require them as their source operands. After all the source operands of the operation are available at the functional unit, the operation is executed in the two execution stages. The latency of ALU operations is one cycles while that of MUL and MEM operations are two clock cycles. The CTRL unit executes branch operations and controls the flow of execution of the program by updating the program counter (PC) when a call/return operation is executed or a branch is taken. Otherwise the fetch unit increments the PC to point to the next VLIW instruction bundle, $PC = PC + INSTRUCTION\_BUNDLE\_SIZE$. Upon execution of a `stop` operation, the branch unit sends a end of program signal to the fetch unit to stop reading instructions. In the write back stage of the pipeline, the results from the two execution stages are stored to the register files and to data memory.

The main advantages of this design-time reconfigurable VLIW processor implemented on an FPGA are [21]:

- simpler hardware design than Reduced Instruction Set Computing (RISC) processors[3],

[3]RISC processors are types of microprocessor that uses small highly optimized set of instructions that can be executed in one cycle.

- availability of existing compiler toolchain which has support for C code. The programmer is not required to have hardware knowledge to program the ρ-VEX VLIW processor core.

2.3 Run-time Reconfigurable ρ-VEX VLIW Processor

Run-time reconfigurability of a VLIW processor core allows it to change between different processor architectures at run-time to take advantage of the available resources. The 2-4-8-issue run-time ρ-VEX processor [1, 2] is a reconfigurable multi-core processor that can change its issue-width and number of general registers as per the requirement of the running application to gain performance and reduce dynamic power consumption. It uses multiple 2-issue independent ρ-VEX cores as basic processors, which can act as a multi-core VLIW processor with four two-issue cores, and combines those cores to make a bigger issue-width core processor when necessary. The register file can also be dynamically partially reconfigured to create a separate register file with a size required by the application [22]. This design of the register file reduces the area considerably when all the registers are not needed. The four two-issue processor organization is depicted in Figure 2.2. Each of the cores are connected to a dedicated data and instruction memory which makes them capable of running different programs. The four cores have a load/store units for accessing the data memory.

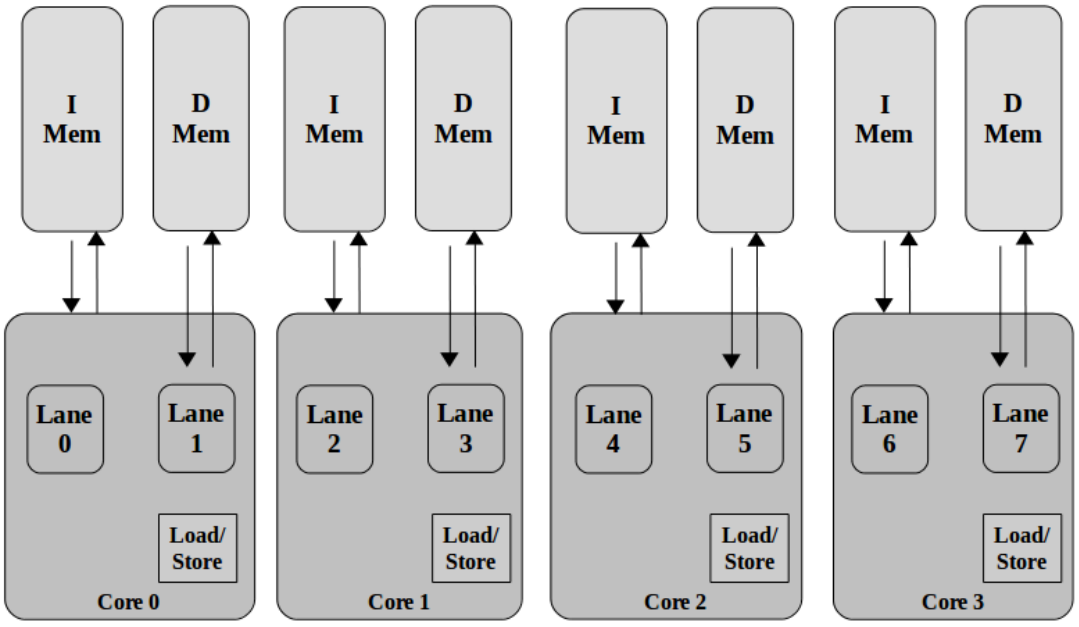


Figure 2.2: The four 2-issue ρ-VEX cores. I-Mem and D-Mem refer to the instruction and data memories of the processor.

Each 2-issue core contains one fetch unit which retrieves two operations from the four

instruction memories every clock cycle. To control the flow of execution of the running programs, the cores have a separate PC and its values are updated by the CTRL units located in each core (pipelanes {0, 2, 4, 6}). An *address select* unit is used to determine the PC addresses from which the fetch units should load the next two operations based on the mode of operation the processor.

To take advantage of applications with high ILP such as vector operations, the 2-4-8-issue ρ-VEX core can be adapted to a bigger issue-width VLIW processor at run-time by changing the value of a signal called *issue_ctrl*. Another signal, *power_ctrl*, is used to control the power consumption of the cores by clock gating the clock sources of processor cores that are not being used. Figure 2.3 shows the organization of a dual core processor each with four issue VLIW processor architecture. The second and fourth data memories are used for the two 4-issue VLIW processor cores. The other two data memories are clock gated to prevent power dissipation.

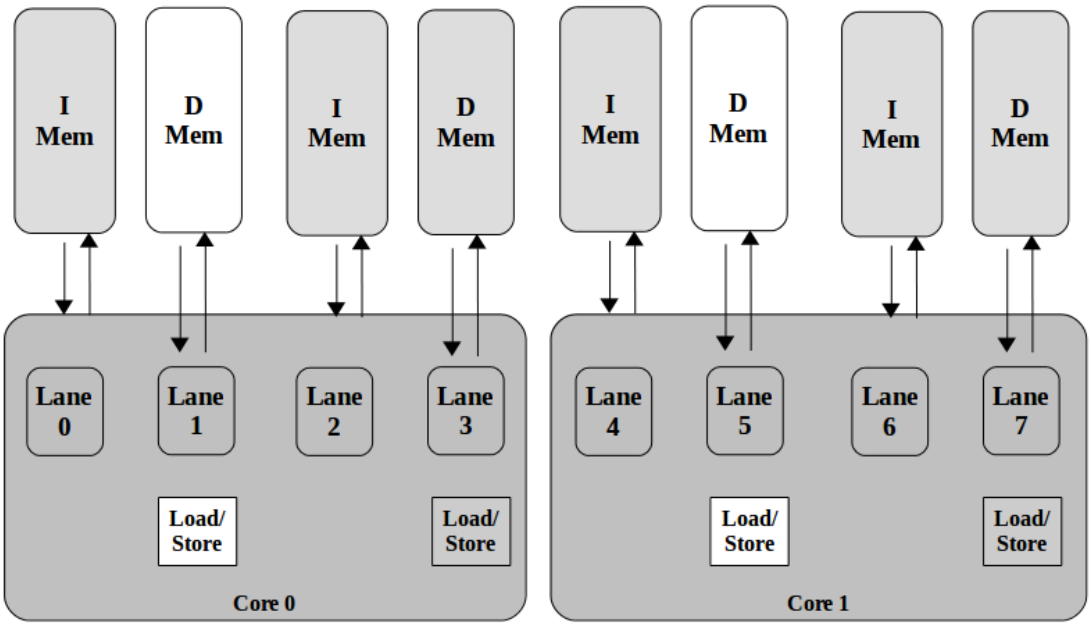


Figure 2.3: The two 4-issue ρ-VEX cores.

In the dual core 4-issue ρ-VEX VLIW processor configuration, the two fetch units in each 2-issue cores are combined to be able to read four continuous operations (instruction bundle) from the instruction memory. The two instruction memories in the two cores should contain the same instructions of a program that are compiled for 4-issue VLIW processor. Two different applications can run at this mode of the 2-4-8-issue ρ-VEX processor simultaneously by loading the instructions of the applications into each of the two instruction memories of the two cores. The PC values of the two cores are controlled by the control unit located at pipelanes {0, 6} respectively.

Figure 2.4 depicts the architecture of the single 8-issue ρ-VEX VLIW processor. Because only one of the three data memories is used for this architecture, the others are gated off. The CTRL unit in pipelane {6} updates the PC of the program. The fetch units are combined to serve as a single 8-issue fetch unit. the fetch units fetches eight operations from the instruction memories every clock cycle based on PC addresses determined by the *address_select* unit. The load/store unit at pipelane {1} is used for executing memory operations. All the memory access operations of a given program are scheduled on this issue slot of the processor by the *binutils* assembler.

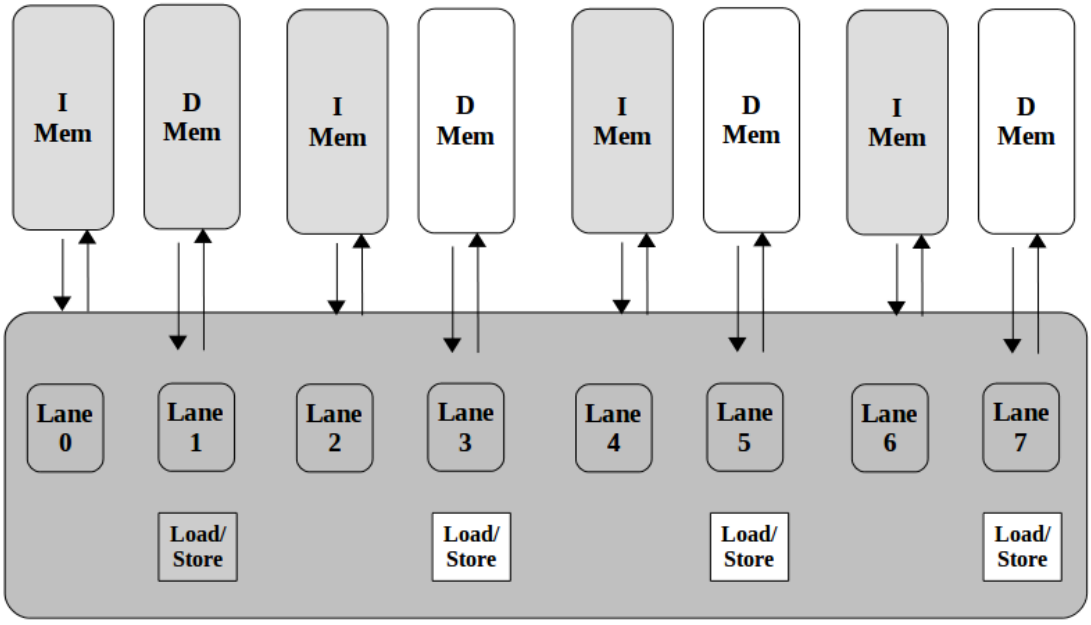


Figure 2.4: One 8-issue ρ-VEX core.

The processor can also operate in a one 4-issue core and two 2-issue cores. In this mode, the processor is capable of executing three different programs at the same time.

The architecture of the 2-4-8 run-time adaptable ρ-VEX processor is mainly divided into two sections:

The Frontend

This is the dynamically adaptable section and it contains the units that have to be reconfigured at run time when changing issue-width of the processor. The Fetch unit, decode unit, write-back unit, the multiported GR register file and the BR register file make up this section.

The GR register file is implemented using 128 BlockRAMS (BRAMS) [23] and it has 8-write and 16-read ports. The VEX ISA defines a VLIW processor to have 64 32-bit multiported general purpose registers. The BRAMs, which can provide up to 256 32-bit registers, are efficiently utilized in the run-time adaptable ρ-VEX processor.

When the processor is running in a two 4-issue VLIW core mode, the first 64 register {0 - 63} in the BRAMs are assigned to the first core, while the other processor core takes the registers from {64-127}. An Address Generation Unit (AGU) maps the read/write register addresses from the cores into their dedicated regions based on the mode of operation of the processor which specified by *issue_ctrl* signal. The first 64 registers from the register file are utilized when the processor is running in 8-issue core mode.

Slice registers are utilized to implement the 32 1-bit branch register file with eight read and write (8R8W) ports. As per the VEX ISA, a VLIW processor should contain 8 1-bit branch registers and the run-time adaptable processor satisfies this requirement by dedicating 8 branch registers to each of the four 2-issue cores. When the processor is in two 4-issue processor mode, the address generation unit of the BR assigns the first 8 branch registers to the first core and the next 8 registers to the second core. In one 8-issue VLIW processor mode, only the first branch registers are used by the processor.

The Backend

The backend section is a static region of the design with units that do not need reconfiguration with increasing or decreasing the issue-width of the processor. The units in this section are 4 ALUs, 4 MULs, 2 load/store units and a write-back unit. Since the mapping of the source and destination operand addresses of the GR and BR register files into their correct dedicated regions in the BRAMs is handled by the AGU, the functional units in the execution stage of the pipeline and the write-back unit does not need adaptation when the processor changes its operating mode.

The *issue_ctrl* is a two bit control signal that defines the processor mode of the 2-4-8-issue ρ-VEX core. Table 2.2 shows the control signal and the mode of operation of the run-time reconfigurable VLIW processor core.

***issue_ctrl* signal**	**Mode of operation of the 2-4-8-issue processor**
"00"	One 8-issue VLIW processor core
"01"	Two 4-issue VLIW processor cores
"10"	One Four-issue and two 2-issue VLIW processor cores
"11"	Four 2-issue VLIW processor cores

Table 2.2: The *issue_ctrl* control signal and the mode of operation of the run-time reconfigurable ρ-VEX processor core.

2.4 Related Work

This section discusses about the previous work in the area of clustered VLIW processors. It presents the inter-communication models for multi-cluster architectures. A discussion on the performance, area and power overhead of those ICC models is performed.

In a clustered VLIW architectures, the functional unit in a cluster are connected to a local register file. During execution of a program, a FU in one cluster may require, as its source operand, a register data produced and stored in register file of another cluster. Inter-cluster communication (ICC) model is a way to exchange data (register values) between different clusters.

Terechko et al. [24, 25] identified and evaluated the performances of five different inter-cluster communication models for clustered VLIW architectures. They used the commercial TriMedia C/C++ compiler and developed a scheduling algorithm for the five ICC models. The models are evaluated based on extra read/write port added, VLIW instruction size and cycle count overhead (extra latency of inter-cluster data transfers), inter-cluster bandwidth and register pressure for two cluster and four cluster 8-issue slot machines using RTL simulations. The design was not implemented in FPGA and does not provide parametric extendability feature. The five ICC models are presented in more depth as follows:

2.4.1 Copy Operation ICC model

In the copy operation ICC model, inter-cluster communication is performed through a bypass network and inter-cluster registers. The VEX ISA supports this ICC model as an inter-cluster communication model. Explicit copy operations (`send` and `recv`) are utilized when communication of data between clusters is required. The `send` operation reads the values of the source register and writes it in to the inter-cluster register while the `recv` operations reads the data from the inter-cluster register and stores it into the destination register in the destination cluster. Those operations will occupy a number of VLIW issue slots which will block scheduling of other regular operations and hence the code size will increase compared to single cluster VLIW architectures. Copy operation ICC model takes one to two cycle depending on the implementation.

The compiler has to divide and schedule the instructions of a given program between the clusters in such a way that it minimizes the number of ICC operations. Advantages of using this model is that it doesn't expand the width of the VLIW instruction bundle which lessens the complexity of the decode hardware logic. In the VLIW processor, all pipelanes have an inter-cluster communication functional unit. The register file is divided into smaller units, reducing the complex multiplexing hardware connecting the FUs and the register file. This ICC model reduces the area utilization substantially without affecting the performance of the processor compared to the single cluster organization.

Lx/ST200 [26] is a scalable and customizable VLIW processor technology platform by HP and STMicroelectronics that allows configuration of instruction issue-width, number and type of functional unit and instruction set of the processor. It supports clustered architecture with copy operation as an inter-cluster communication model. A pair of `send` and `recv` operations are used for transferring data between register files of the clusters. An implementation of copy operation ICC model on a VLIW processor that distributes instruction among clusters at run-time is proposed by Ramon et al [27]. Dynamic register renaming with the use of single register map table is performed. The copy operation is issued by the dispatch logic at run-time. The design of this hardware

logic is complex and dynamic scheduling of instructions to pipelanes is not power efficient.

A run-time length adaptive 8-issue clustered VLIW processor with ability to gate off unused clusters is discussed in [28]. Copy operation is used as an inter-cluster communication model. The parametric extensibility of the processor with machine model parameters is absent. The processor design does not have a pipelining and forwarding logic. This lead the processor to run at a slower clock frequency. The compiler scheduling algorithm was not efficient enough that more ICC operations are generated.

2.4.2 Dedicated Issue Slot ICC model

This ICC model is an extension of the copy operation ICC model and it dedicates a separate issue slot for inter-cluster communication in the VLIW instruction bundle. In each cluster, pipelanes dedicated only to inter-cluster communication has to be designed in hardware and this increases the instruction decoders and number of multiplexers in the bypass network. As a result of dedicated issue slots, the width of the VLIW instruction bundle increases and so does the code size. The number of read ports to each register file in the clusters also increases by the number of dedicated issue slots for inter-cluster communication.

Inter-cluster operations will be scheduled in those dedicated issue slots which frees up the other slots for the regular operations, better scheduling freedom than the other models. Inter-cluster data transfer can happen on those slots without blocking the scheduling and execution of other regular operations. The compiler can take advantage of this ICC model by scheduling data independent operations in the slots for the regular operations and inter-cluster communication operations in the same cycle. This ICC model increases the area utilization of the processor but it outperforms the other ICC models from performance point of view. The power consumption of a VLIW processor with this ICC model is higher.

2.4.3 Extended operand ICC model

In extended operand ICC model, the source register number along with its source cluster number are specified in the syllable of the operations. The value of the register will then be read remotely from another cluster's register file and immediately consumed with out storing it in local register file. This register access takes one or more cycle, which will stall the processor. One solution to this is that to design a hardware logic that is able to track the cluster number of the operations in the instruction decode stage and initiate remote register read as early as possible. This increases the complexity of the bypass network, multiplexers and thus area utilization of the processor. Because the data read from other cluster is directly consumed by the FU, reuse of the data is not possible. The register has to be read again, which is a waste of cycles and resources.

The TI TMS320C64xx family DSP VLIW processors [29, 30] have two separate register files, A and B. *VelociTI* [31] is one of the oldest clustered VLIW processors from Texas Instruments. There are two register file cross paths that allow the functional units to access source operands from the other register file. TI TMS320C64xx DSP

implements an extended source operand as an inter-cluster communication model and it restricts the number of cross path reads to two per clock cycle. This cross path read introduces a delay clock cycle known as cross path stall. Another clustered architecture that uses extended operand ICC model is the TigerSharc architecture [32] from Analog Devices. It exploits instruction and data level parallelism by changing modes between VLIW and SIMD architectures. TigerShark has two computational block with separate register files X and Y. The processor stalls until the data of all source operands are available at the functional units.

The code size for this ICC model is smaller compared to copy operation and dedicated issue slot ICC models because there are no explicit copy operations in this model. The width of the instruction bundle is increased as the cluster number of the source operands of the operations are added to the operations. Additional read ports are required to load the source operand values from other register files. This leads to an increase in the area and power overhead of the register file and complexity of the decoder unit.

2.4.4 Extended results ICC model

Instead of extending the source of operand in extended operand ICC, this model extends the destination register operands with cluster identification. The result of the operation will be forwarded to the destination cluster's register file without storing it on the local cluster where they are computed. The processor should make sure that the result is stored in the destination register file before it can be accessed.

Extra write ports to the register files are required to write the results of an operation when the destination cluster is not the same as the cluster they are executed in. This increases the area usage of the register file. The compiler should take into account the latency of storing results into another cluster during the scheduling of operations. This constraint of considering the transfer latency limits the scheduling freedom of the compiler. The destination cluster and operand of an operation has to be specified, which increases the width of the instruction bundle.

2.4.5 Broadcasting ICC model

The broadcasting ICC model is a variant of extended results inter-cluster communication model where the results of an operations are broadcasted to all clusters in the VLIW processor. One of the hardware implementation of this model is using replicated shared registers with some kind of synchronization mechanism, which takes up a substantial area. Additional read/write ports to the shared registers are required and the synchronization mechanism adds complexity to the design.

The destination operand of an operation is added with a flag to broadcast the result or store it locally. As a result, the width of the instruction bundle increases. Code size complexity of this ICC model is similar to that of extended result ICC model. Broadcasting the result of an operation to all cluster is never a power and area efficient ICC model.

The Billions of Operations Per Second (BOPS) Inc ManArray [33] is a reconfigurable DSP core that can be configured on the number of clusters/ processing elements (PE). It can also be reconfigured as a VLIW multiprocessor, VLIW processor with (without) SIMD support. Each PE in the ManArray contains a multiported 32×32-bit register file, VLIW memory unit and local data memory. Inter-cluster communication is performed in single cycle by using a mix of copy operation (explicit PE exchange instructions) and broadcast ICC models. A cluster switch along with complex multiplexers connecting the processing elements are used to receive the values from other cluster. The MAJC, Microprocessor Architecture for Java Computing [34], also uses broadcast ICC model to send updates of results between units in different clusters. It is scalable VLIW processor designed in the mid 90's to meet the requirements of general-purpose computing. It divides the processing units into clusters and it has synchronized shared registers that can be accessed from all the clusters. The compiler stores values in the shared registers when it wants to transfer results between clusters.

A scalable wide-issue clustered VLIW co-processor with a reconfigurable interconnect that targets kernel loops of a program is presented in [35]. Each cluster has an issue-width of one and the register file dedicates two read and write ports for ICC. Register values are broadcasted to clusters when data transfer is required. Usage of boadcast ICC model increases the power overhead of processor and the complexity of the multiplexers in the bypass network.

Yukinori et al. [36] investigated the implementation of clustering and optimal degree of partitioning using alpha 21264 microprocessor [37] as a baseline architecture. They divided the clustered organizations into two: multiple coherent register file organization (with broadcasting ICC model) and non-consistent register file (with copy operation ICC model).

2.5 Conclusion

In this chapter, basic background concepts and related work on clustered VLIW architectures are presented. The VEX system that includes three components, the VEX instruction architecture, VEX compiler and a simulator system, are introduced. Pros and cons of different ICC models and variety of compiler scheduling techniques for clustered architectures are discussed. The broadcasting ICC model consume more power than the other ICC model while dedicated issue slot ICC model increases the area utilization of the processor. The VEX system provides complete support for clustered organization of VLIW processors in its ISA and compiler while the other scheduling techniques do not provide that. Thus, a clustered organization of the ρ-VEX reconfigurable processor is implemented using the trace scheduling algorithm of the VEX compiler and ISA supported copy operations ICC model. Clustering the register file reduces the register pressure and area overhead. But increasing the number of clusters increases the inter-cluster communication overhead, cycle count and thus does not lead to performance improvement of the processor [38]. Thus, an optimal clustering of two with 32 register in each cluster is chosen for the implementation of the clustered ρ-VEX processor.

Design-time Reconfigurable Clustered ρ-VEX Processor

3

In this chapter, detailed design and implementation of the design-time reconfigurable clustered ρ-VEX VLIW processor is presented. The VEX ISA and compiler support for clustered VLIW processors are introduced in Section 3.1. In Section 3.2, modification performed on the compiler toolchain are discussed. The detailed design and implementation of the clustered ρ-VEX processor is presented in Section 3.3. In Section 3.4, a bug in the VEX compiler for clustered VLIW processors and its fix are discussed. The implementation of clustered ρ-VEX processor using dedicated issue slot inter-cluster communication model is presented in Section 3.5.

3.1 VEX ISA and Compiler Support for Clustered VLIW Processors

The VEX ISA is a scalable and flexible architecture for embedded VLIW processors and it supports not only a single cluster organizational model but also multi-cluster execution architectures. VEX is a clustered architecture and provides scalability in issue width and functionality using modular execution clusters. A cluster consists of local register files and set of FUs that are directly connect to the register files. Each cluster contains a load/store unit to access the data memory.

Figure 3.1 depicts the multi-cluster organization of VEX architecture with N clusters. The clustered architecture can have multiple data cache blocks connected to the different clusters by a crossbar to allow multiple memory configurations. An inter-cluster path is provided for inter-cluster communication between the clusters. The program counter (PC) address is controlled by the control unit in cluster 0.

The clusters obey the following rules [5]:

- A cluster can issue multiple operations from the same instruction bundle in a given cycle.
- Clusters can have different issue width and functional units.
- Clusters can have different ISA support other than VEX ISA.
- FUs within a cluster are indistinguishable, a cluster can execute a finite set of simultaneous operations. The decoder logic assigns operations to units in a cluster.

Cluster 0 is a *special* cluster where control operations are executed and must be present in any VEX implementation. A single program counter (PC) and a unified

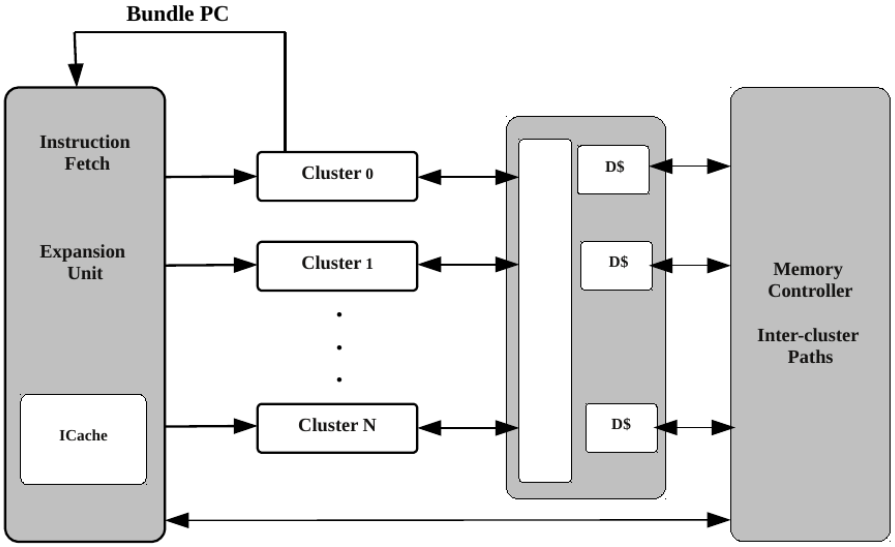


Figure 3.1: High level structure of a VEX multi-cluster organization.

instruction cache (I-cache) are used to control all the clusters so that they run in proper sequence, one instruction bundle per clock cycle.

Clustering the register files and the functional units of a VLIW processor reduces the read/write ports, area utilization and power consumption of the processor but at a cost of reduced performance which is caused by communication overhead. Additional operations for inter-cluster communications are introduced when a functional unit in one cluster requires data that is produced and stored in register file of another cluster. The VEX ISA supports clustered VLIW architectures with copy operation inter-cluster communication (ICC) model that uses a pair of operations, `send` and `recv` operations. The `send` inter-cluster operation copies the data from the source register into an inter-cluster path while the corresponding `recv` operation retrieves the data from the inter-cluster path and places it in the destination register. Both operations should be issued in the same instruction bundle for the inter-cluster communication to occur correctly.

A machine configuration model defining the detailed architecture of the target clustered VLIW processor must be provided to the VEX compiler during compilation of a program. In this configuration file, parameters such as the number of clusters, the types and number of functional units in each cluster, the number of registers in the register files (GRs, BRs and LR register files), source/destination for inter-cluster copy operations and the latencies of each type of operations (ALU, MUL, MEM) can be specified. Cluster-specific resources/parameters are defined by indexing the cluster number next to the parameter separated by a dot (.); for example `IssueWidth.0 2`. The machine configuration models in Table 3.1 shows the resources used in a 2-cluster 4-issue or *(2 + 2)-way clustered ρ-VEX processor* and 2-cluster 8-issue or *(4 + 4)-way clustered ρ-VEX processor*.

The number of clusters of the VLIW architecture can be passed as an input argument

# 4 issue vex 2 clusters	# 8 issue vex 2 clusters
CFG: Debug 0	CFG: Debug 0
RES: IssueWidth 4	RES: IssueWidth 8
RES: IssueWidth.0 2	RES: IssueWidth.0 4
RES: IssueWidth.1 2	RES: IssueWidth.1 4
RES: MemLoad 2	RES: MemLoad 2
RES: MemStore 2	RES: MemStore 2
RES: ALU.0 2	RES: ALU.0 4
RES: ALU.1 2	RES: ALU.1 4
RES: Mpy.0 1	RES: Mpy.0 2
RES: Mpy.1 1	RES: Mpy.1 2
RES: CopySrc.0 2	RES: CopySrc.0 4
RES: CopyDst.0 2	RES: CopyDst.0 4
RES: Memory.0 1	RES: Memory.0 1
RES: CopySrc.1 2	RES: CopySrc.1 4
RES: CopyDst.1 2	RES: CopyDst.1 4
RES: Memory.1 1	RES: Memory.1 1
REG: $r0 32	REG: $r0 32
REG: $r1 32	REG: $r1 32
REG: $b0 8	REG: $b0 8
DEL: Alu.0 0	DEL: Alu.0 0
DEL: Alu.1 0	DEL: Alu.1 0
DEL: Multiply.0 1	DEL: Multiply.0 1
DEL: Multiply.1 1	DEL: Multiply.1 1
DEL: Load.0 1	DEL: Load.0 1
DEL: Store.0 1	DEL: Store.0 1
DEL: Load.1 1	DEL: Load.1 1
DEL: Store.1 1	DEL: Store.1 1
DEL: CpGrBr.0 1	DEL: CpGrBr.0 1
DEL: CpBrGr.0 0	DEL: CpBrGr.0 0
DEL: CpLrGr.0 0	DEL: CpLrGr.0 0
DEL: CpBrGr.1 0	DEL: CpBrGr.1 0
DEL: CpLrGr.1 0	DEL: CpLrGr.1 0

Table 3.1: The machine configuration model used for the (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processor.

to the VEX compiler as **-width <clusters>**. The default value this argument is 1, implying a single cluster organization.

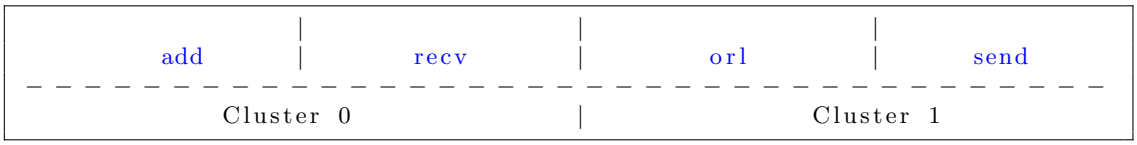
When compiling an application for a multi-cluster VLIW organizations, the VEX compiler generates a special kind of `mov` operation when a register data transfer between clusters is required. In this `mov` operation, the source and destination clusters, and

source and destination register numbers are specified.

```
## Copy register no. 4 of cluster 1 to register no. 8 of cluster 0
##
   c0     add $r0.2 = $r0.7, 10
   c1     orl $r1.3 = $r1.5, $r1.16
   c0=c1  mov $r0.8 = $r1.4
;;
```

Listing 3.1: Inter-cluster `mov` operation

The Listing 3.1 shows the syntax of inter-cluster `mov` operation in an assembly code generated by the compiler for a (2 + 2)-way clustered ρ-VEX processor. In addition to the syntax of inter-cluster `mov` operation, the Listing shows what an assembly code generated by VEX compiler for multi-cluster VLIW architectures looks like. The first operation, `add`, will be executed in cluster 0 of the clustered ρ-VEX processor. This is indicated by the `c0` prefix to the operation. The operation adds the content of register number 7 of cluster 0 and an immediate operand of value *10* and stores the result in register number 2 of cluster 0. The second operation is `c1 orl`, which will be executed in cluster 1. It performs logic `or` operation on the two register operands, register 5 and 16 of cluster 1, and stores the result in register 3 of cluster 1. The last operation, which is an inter-cluster communication operation, transfers data from register number 4 of cluster 1 to register number 8 of cluster 0. To perform this inter-cluster communication using the copy operation ICC model, the assembler is modified to generate an inter-cluster `send` operation that will be executed on cluster 1 and an inter-cluster `recv` operation to be executed in cluster 0. The double semicolon `;;` signifies the end of an instruction bundle. Any line that starts with a `#` is a comment and is ignored by the assembler. The instruction bundle for the above assembly code will be as follows:



3.2 Compiler Toolchain Adaptation for Clustered Architectures

In this Section, the development of the compiler toolchain support for clustered architectures such as the borrowing schemes, syllable of inter-cluster copy operations and conditional branch operations are presented.

3.2.1 Modified GNU Binutils for Clustered ρ-VEX Architectures

The *GNU binutils* [39] are collection of compiler binary utilities for creating object files. It includes tools such as assembler, linker, disassembler, profiler and others. The *GNU binutils* was modified to be able to generate object files and hex files for the single cluster ρ-VEX VLIW architecture. The modified *GNU binutils* assembler is adapted to include support for multi-cluster ρ-VEX architectures. The assembler parses the inter-cluster `mov` operation generated by the VEX compiler and outputs two inter-cluster operations: a `send` operation to be issued on the issue slot of the cluster containing the source register and a `recv` operation to be issued on the issue slot of the destination cluster.

The assembler accepts as an input argument a configuration specifying the functional units that are available in each issue slot. The configuration represents the functional units using bit positions which are ordered as inter-cluster communication functional unit, CTRL unit, MEM unit, MUL and ALUs unit. It uses binary 0's and 1's bit to show their availability. For example, an issue slot with configuration value of *b10101* or *0x15* signifies that the issue slot have inter-cluster communication functional unit, load/store unit and an arithmetic and logic unit. Table 3.2 presents the configuration setting used for the (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processors.

(2 + 2)-way Clustered	--config B773
(4 + 4)-way Clustered	--config 93351335

Table 3.2: Configuration setting used for the (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processors.

The inter-cluster communication functional unit is present in every issue slot, thus the assembler implicitly adds a 1 to that bit position. The configuration setting with a 1 on the bit positions of those ICC FUs can also be explicitly specified, as --config 1b171713 for the (2 + 2)-way clustered ρ-VEX processor.

A new input argument is added to provide the assembler with the number of clusters of the ρ-VEX architecture. The default value of the cluster argument is 1, which implies a single cluster organization architecture. The input argument is specified as:

as-new --cluster <clusters>

The assembler for clustered VLIW processors parses each operation and its operands. To identify whether an operation is a regular `mov` operation or an inter-cluster `mov`

operation, the assembler checks for an = sign after it read the destination cluster number of the operation. When it finds an = sign, the next two characters are read to identify the source cluster number for the inter-cluster `mov` operation. The destination and source registers of the `mov` operation are then parsed.

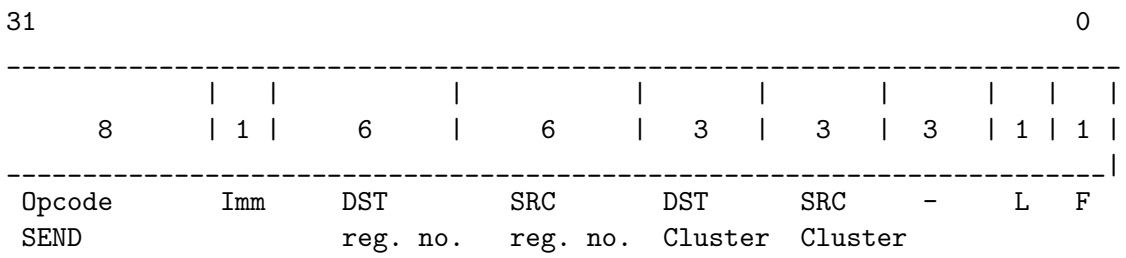
The *GNU objdump* display different information about the object files based on the compiler options provided. One of these compiler options is **-D** or **-d**, the *disassembler*, that outputs the assembler mnemonics of the machine instructions in all sections of the object file. For the clustered ρ-VEX architectures, the disassembler is modified to extract the cluster number identifier, recognize the inter-cluster copy operations along with the source and destination operands and display it. The output of the disassembler for an object file compiled for a (2 + 2)-way clustered ρ-VEX processor is depicted in Figure 3.2. It displays the PC address, cluster number identifier, mnemonic of the operation, operands, and the syllable of each operation respectively.

```
f50:       ;;      c0     br       $b0.0, 0x4e                  24 00 4e 21
f54:               c0   recv       $r0.11 = $r1.14              2b 1c 58 20
f58:               c1    ldb       $r1.15 = 0xc[$r1.3]          13 9e 18 30
f5c:               c1   send       $r1.14 = $r0.11              2a 1c 58 20
f60:       ;;      c0     br       $b0.2, 0x4a                  24 00 4a 2b
f64:               c0    nop                                   60 00 00 02
f68:               c1  cmplt       $b1.0 = $r1.14, $r1.4        4f 00 70 80
f6c:               c1  cmpgt       $b1.2 = $r1.14, $r1.4        47 04 70 82
f70:       ;;      c0    brf       $b0.1, 0x46                  25 00 46 05
f74:               c0   recv       $r0.10 = $r1.12              2b 18 50 20
f78:               c1    ldb       $r1.4 = 0xc[$r1.5]           13 88 28 30
f7c:               c1   send       $r1.12 = $r0.10              2a 18 50 20
f80:       ;;      c0     br       $b0.3, 0x42                  24 00 42 2d
f84:               c0   recv       $r0.9 = $r1.10               2b 14 48 20
f88:               c1    ldb       $r1.14 = 0xd[$r1.3]          13 9c 18 34
f8c:               c1   send       $r1.10 = $r0.9               2a 14 48 20
f90:       ;;      c0     br       $b0.4, 0x3e                  24 00 3e 33
f94:               c0    nop                                   60 00 00 02
f98:               c1  cmplt       $b1.3 = $r1.15, $r1.4        4f 06 78 80
f9c:               c1  cmpgt       $b1.4 = $r1.15, $r1.4        47 08 78 82
```

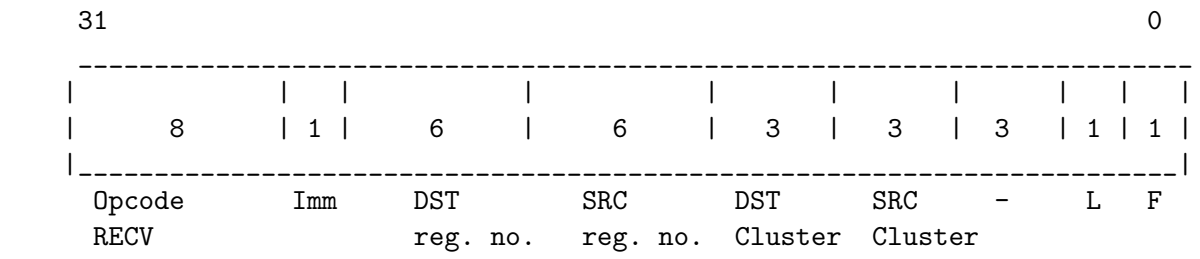
Figure 3.2: Output of disassembler for a (2 + 2)-way clustered ρ-VEX processor.

3.2.2 Inter-cluster Copy Operations Syllable

The syllables of the inter-cluster `send` and `recv` operations must specify the destination register number, source register number, destination and source cluster number. The syllable for `send` operation is defined as:



The syllable for the inter-cluster `recv` operation is given as:



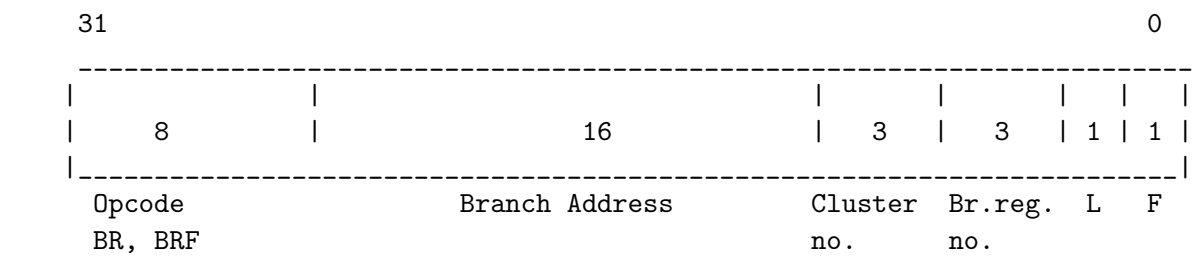
This structure of the syllable, position of the opcode, immediate flag, source and destination registers, is similar to the structure used in the baseline ρ-VEX architecture. The L and F bits are used to specify the start and end of an instruction bundle.

3.2.3 Conditional Branch Operations Syllable

The functional units of the ρ-VEX processor can read/write its result to a register file in the same cluster directly. The branch unit has an additional property. Cluster 0, the only one that has a branch unit, can also access the data stored in a register file of another cluster.

The clustered ρ-VEX processor implements this property by modifying the syllable of those operations that can access the branch registers of other cluster to include the cluster number from which it has to read. The *br* and *brf* operations are the two conditional branch operations. Three bits are dedicated to store the cluster number of the branch source operand in the syllable. The branch address of the operation is now stored using 16 bits which limits the maximum branch distance for those branch operations to 2^{16}. The *gas/config/tc-rvex.c* of the *binutils* is modified to place the 16 bits of the offset branch address of the branch operations in the syllable starting from *startbit* of 8.

The new syllable for the conditional branch operations is given as follows:



In *binutils*, *binutils/elf32-rvex.c*, a new entry is added to the tables that defines the relocation map and *HOWTO* relocate of the assembler for the ρ-VEX architecture. The start bit address, destination mask and the bit size of the branch offset address are specified in the table entry, see Listing 3.2.

```
 . . .
static const struct rvex_reloc_map rvex_reloc_map[R_RVEX_MAX] =
{
  {BFD_RELOC_NONE,                R_RVEX_NONE},
  {BFD_RELOC_32,          R_RVEX_32},
  {BFD_RELOC_RVEX_BRANCH,         R_RVEX_BRANCH},
 . . .
  {BFD_RELOC_RVEX_BR_BRF,         R_RVEX_BR_BRF},
};

static reloc_howto_type rvex_elf_howto_table[] =
{

 . . .

  HOWTO (R_RVEX_BRANCH,                 /* type */
         (2+LOG_MIN_WIDTH),           /* rightshift */
         2,                             /* size */
         19,                            /* bitsize */
         TRUE,                          /* pc_relative */
         5,                             /* bitpos */
         complain_overflow_bitfield, /* complain_on_overflow */
         bfd_elf_generic_reloc,         /* special_function */
         "R_RVEX_BRANCH",               /* name */
         FALSE,                         /* partial_inplace */
         0x0,                           /* src_mask */
         0xfffe0,                       /* dst_mask */
         TRUE),                         /* pcrel_offset */

 . . .

  HOWTO (R_RVEX_BR_BRF,                 /* type */
         (2+LOG_MIN_WIDTH),           /* rightshift */
         2,                             /* size */
         16,                            /* bitsize */
         TRUE,                          /* pc_relative */
         8,                             /* bitpos */
         complain_overflow_bitfield, /* complain_on_overflow */
         bfd_elf_generic_reloc,         /* special_function */
         "R_RVEX_BR_BRF",               /* name */
         FALSE,                         /* partial_inplace */
         0x0,                           /* src_mask */
         0xffff00,                      /* dst_mask */
         TRUE),                         /* pcrel_offset */
};
 . . .
```

Listing 3.2: Relocation map

3.2.4 Borrowing Scheme for Clustered ρ-VEX Processor

A VEX operation can have two kinds of operands, *register operands* (branch, link or GR register) and *immediate operands*. In addition, the immediate operands are of two types, those that can be represented using 9-bits (*short immediate*) and those that require 32-bits (*long immediate*). The syllable of any VEX operation can fit a short immediate operands. For operations with long immediate operands, as it cannot be accommodated in the 32-bit syllable of the operation, VEX supports an immediate extension operation to store the remaining 23-bits. In ρ-VEX, this operation is called *follow* operation and must be issued in the same instruction bundle with the operation that has the long immediate operand them so that there will be no performance penalty. When decoding the operation with long immediate operand, the ρ-VEX processor can read the remaining 23-bits of the immediate operand in the same cycle.

The issue slot scheduler of the assembler should be able to know in which of the free issue slots in the instruction bundle it can issue the *follow* operation when it finds an operation with long immediate operands. This information is called *Borrowing Scheme*. Although the position, type and number of the FUs per issue slot is design time configurable parameter, a default placement of FUs for the clustered ρ-VEX processor is used to determine the borrowing scheme. The 2-cluster 4-issue clustered ρ-VEX has 4 ALU units, 2 MEM units (one per cluster), 2 MUL units. Table 3.3 displays the positioning of the FUs, the possible combinations of operations (with or without long immediate) in an instruction bundle and the borrowing scheme for a clustered ρ-VEX. The mapping scheme shows how the operations can be scheduled and where the *follow* operation can be issued for operations with long immediate operands. Operations that can have long immediate operand are the ALU, MUL and MEM operations. Inter-cluster communication FUs are not considered in the mapping scheme because they do not have immediate operands and they are available in each issue slot of the ρ-VEX processor. Branch operations can fit its immediate branch address (maximum 19 bits) in one syllable. Since every issue slot have an ALU unit, ALU operations are not considered in the mapping scheme.

The resulting borrowing scheme in Table 3.3 shows that every issue slot can borrow from any other issue slot in the instruction bundle for the (2 + 2)-way clustered ρ-VEX architecture. When the decoder unit of a pipelane decodes an operation with long immediate operand, it searchs for a follow operations in the instruction bundle for the slots that are specified in the borrow setting of the lane.

The (4 + 4)-way clustered ρ-VEX architecture contains 8 pipelanes divided into two clusters with local register files. It has eight ALU units, four multiplier units, two Load/ Store units and eight inter-cluster communication functional units. The combination of those functional units along with the operations with or without long immediate operands is large. As a result, the borrowing scheme for this architecture is that any issue slot can borrow from any other issue slot in the instruction bundle. From a hardware resource point of view, the search for the follow operation of an operation with a long immediate operand is expensive as it has to go through every operation in the instruction bundle.

Slot Number	**3**	**2**	**1**	**0**
Functional Units	ALU MUL ICC FU	ALU MEM ICC FU	ALU MEM ICC FU	ALU MUL ICC FU CT
Borrowing Scheme	(0, 1, 2)	(0, 1, 3)	(0, 2, 3)	(1, 2, 3)
Mapping scheme for operations				
MUL0 L, MUL1 L	MUL1 L	S_F(3)	S_F(0)	MUL0 L
MUL0 L, MEM0 L	S_F(1)	S_F(0)	MEM0 L	MUL0 L
MUL0 L, MEM1 L	S_F(2)	MEM1 L	S_F(0)	MUL0 L
MEM0 L, MEM1 L	S_F(2)	MEM1 L	MEM0 L	S_F(1)
MUL1 S, MEM0 L, CT	MUL1 S	S_F(1)	MEM0 L	CT
MEM0 S, MEM1 L, CT	S_F(3)	MEM1 S	MEM0 L	CT
MUL1 L, MEM0 S, CT	MUL1 L	S_F(3)	MEM0 S	CT
MUL1 L, MEM1 L	MUL1 L	MEM1 L	S_F(3)	S_F(2)
MUL1 L, MEM0 S, MEM1 S	MUL1 L	MEM1 S	MEM0 S	S_F(3)
MEM1 L, MUL1 S, CT	MUL1 S	MEM1 L	S_F(2)	CT
MUL0 L, MM0 S, MEM1 S	S_F(0)	MEM1 S	MEM0 S	MUL0 L

Table 3.3: FUs placement, Mapping and Borrowing Scheme for 2 cluster 4-issue width clustered ρ-VEX processor. ALU, MUL, MEM, CT , ICC FU and S_F represent ALU unit, MUL unit, MEM unit, Control/Branch unit, inter-cluster communication functional unit and Syllable for *follow* operation respectively. The symbols S and L are for Short and Long immediate operand.

The borrowing scheme is one of the input arguments to the assembler. The argument is provided as a sequence of numbers specifying the issue slots that an issue slot can borrow from separated by a comma (,) for each pipelane. The borrow scheme for the pipelane are separated by a dot (.). The borrowing scheme argument for the (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processor are given in Table 3.4.

(2 + 2)-way	**--borrow 1,2,3.0,2,3,0,1,3.0,1,2.**
(4 + 4)-way	**--borrow 1,2,3,4,5,6,7.0,2,3,4,5,6,7.0,1,3,4,5,6,7.0,1,2,4,5,6,7.0, 1,2,3,5,6,7.0,1,2,3,4,6,7.0,1,2,3,4,5,7.0,1,2,3,4,5,6**

Table 3.4: Borrow settings for the (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processor.

In the *binutils* assembler, each issue slot of an instruction bundle has a flag called $filled$ to identify if an operation is issued on it. This flag is set to 1 when an operation is already issued on it. When the assembler finds an operation with long immediate operand, it creates the $follow$ operation and adds the 23-bits of the long immediate

operand to the operation. In the issue slot assigning stage of the assembly, the scheduler iterates through the issue slots in the borrow setting of the slot where the operand with long immediate operand is to be issued and checks if the slots are unfilled. The $follow$ operand will be issued at the first slot that is identified to be not filled and that the lane containing the operation is allowed to borrow from.

3.3 Design of Clustered ρ-VEX VLIW processor

The design-time reconfigurable clustered ρ-VEX processor is designed to support the VEX instruction set architecture. The register file of a single cluster organization ρ-VEX processor, which contains 64 32-bit registers, is split into the number of clusters and inter-cluster path is designed for the inter-cluster communication along with ICC functional unit. Each cluster have its own MEM or load/store unit for executing memory read/write operation. This is an advantage for clustered VLIW architectures as they can execute multiple load/store operations per clock cycle. An optimal cluster of two is chosen for the clustering of the ρ-VEX because excessive clustering leads to an increase in the inter-cluster communication and addition of read and write port (multi-port) to the data memory which requires the design of complicated redundant data memories with synchonization of data between them.

Figure 3.3 shows the architecture for a (2 + 2)-way clustered ρ-VEX VLIW processor with local register files, inter-cluster bypass network and dual-ported data memory. Each pipelane contains an inter-cluster functional unit for transferring register data between the clusters. The local register files contain 32 registers and the number of read and write ports to the register files are reduced by half to 4R/2W compared to 8R/4W in the single cluster ρ-VEX VLIW architecture. This reduces the area utilization of the processor because the multiplexer complexity of the register file is decreased.

In a similar fashion, the (4 + 4)-way clustered ρ-VEX processor have two clusters each connected to a local register file with 32 registers. The read/write port of the register files are reduced to 8R/4W from 16R/8W in the 8-issue ρ-VEX processor. The pipelanes have an ICC functional unit which is connected to the inter-cluster path with 1R/1W port.

In addition to the inter-cluster path and ICC FUs, the decoder unit is modified to decode the inter-cluster operations and the forwarding logic is adapted to forward the correct data from the execution and write-back pipeline stages to the inter-cluster communication functional units. Detailed hardware implementations are presented in the following sections.

3.3.1 Decoder unit

Modifications have been made to the decoder unit of the ρ-VEX processor to decode the ICC and branch operations. The unit has to extract the source and destination, cluster and register numbers from the syllable of the ICC operations.

As explained in Section 3.2.3, the syllable of the conditional branch operations, `br`

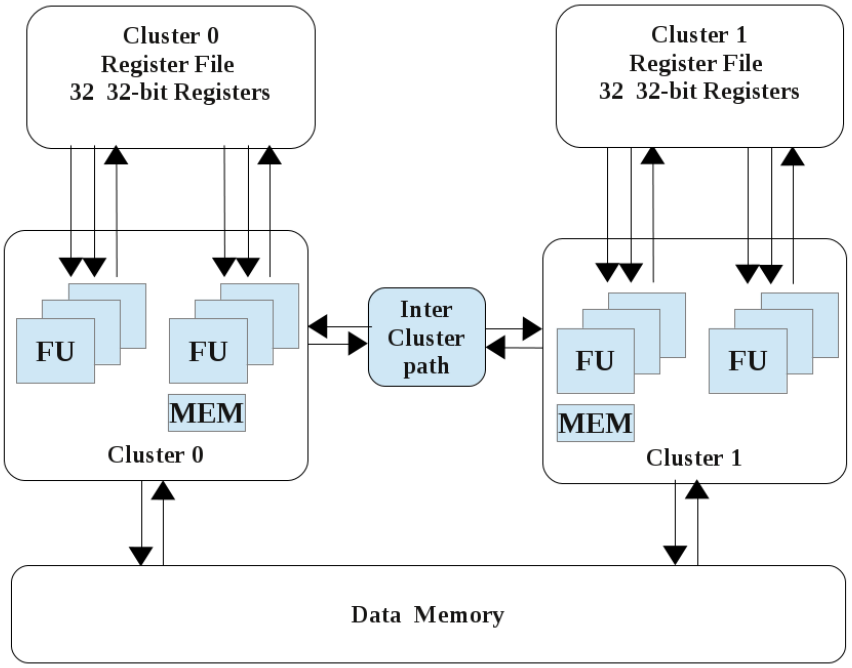


Figure 3.3: Clustered ρ-VEX architecture with inter-cluster path and local register files.

and `brf`, are altered so that the cluster number of the branch register number of the operation can be added to it. 3-bits are used for the cluster number and 16 bits for the branch address. The decoder logic of the ρ-VEX processor is modified to extract the 16 bit branch offset address of the operation, instead of 19 as in the baseline processor. The branch address is then passed to the branch unit of the processor.

3.3.2 Copy Operation Inter-cluster Communication model

To support copy operation ICC model in the ρ-VEX processor, inter-cluster path and ICC funcational units are designed and implemented. The inter-cluster paths are used to temporarily store the data from source cluster for the inter-cluster communication between clusters. The ICC functional units in the pipelane write into or read from the inter-cluster path based on the inter-cluster operation.

The inter-cluster path is a bypass network made of multiplexers that are connected to each inter-cluster communication functional units using 1 read and 1 write (1R1W) port. Figure 3.4 depicts the copy operations inter-cluster communication model in a (2 + 2)-way clustered ρ-VEX VLIW processor. Each pipelane has an inter-cluster communication functional unit for transfer of register values between clusters. Similar concept design is applied in the (4 + 4)-way clustered ρ-VEX processor. The hardware description language (HDL) programs for the ICC functional unit and inter-cluster path can be found in Appendix A and B respectively.

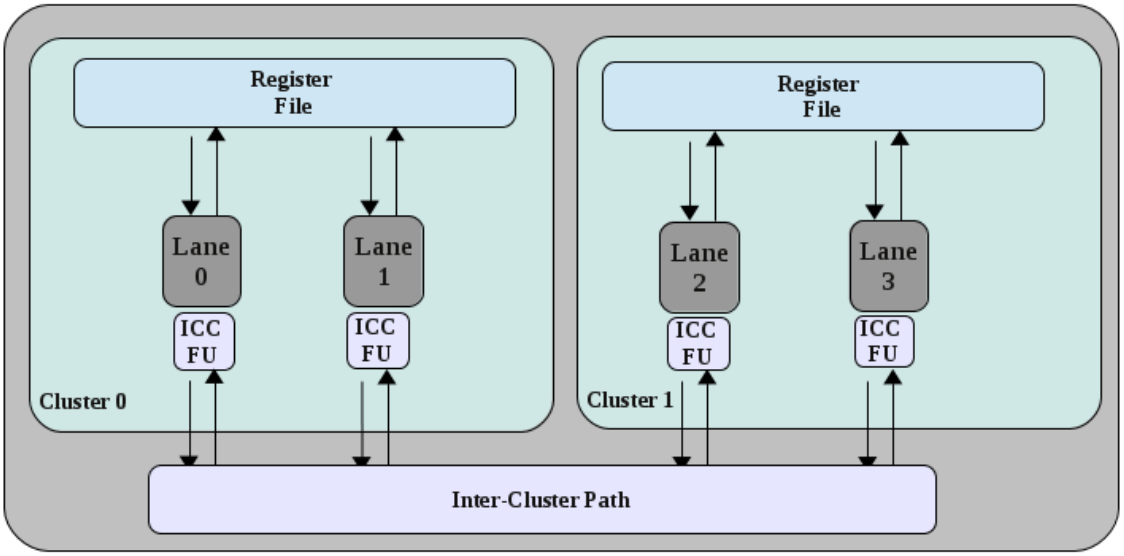


Figure 3.4: Copy operation ICC model in a (2 + 2)-way clustered ρ-VEX processor.

The two inter-cluster communication operations are described as follows:

1. **The SEND operation**
   The `send` operation writes a header and the register data to the inter-cluster write data path, as shown in Table 3.5. The header consists of the destination cluster, and source cluster and destination register number.

5	3	3	32
DST Register	DST Cluster	SRC Cluster	Data

Table 3.5: SEND operation. Inter-cluster write data path.

2. **RECV operation**
   The `recv` operation, on the other hand, writes the header into the inter-cluster read data path, see Table 3.6. The header should have the same content as the one in the `send` operation for the transfer of the data to be successful.

5	3	3
DST Register	DST Cluster	SRC Cluster

Table 3.6: RECV operation. Inter-cluster read data path.

The inter-cluster path compares the two headers received from the `recv` and the `send` operation and when it finds a match, it copies the 32-bit register data written by the `send` operation and bypasses it to inter-cluster path read port of the destination cluster. After which the register data will be stored in the register file of the destination cluster.

3.3.3 Forwarding Logic

Forwarding Logic passes values of a register (general purpose, link and branch) from the execute pipeline stage when they are required by a data dependent operation before they are written back to the register files. The forwarding logic in the ρ-VEX processor compares the register addresses of the source operands from decode/fetch pipeline stage of the current operation with the destination register addresses from execute stages of the pipeline and when it finds a match it forwards the value at that register address to the corresponding functional unit in the execute stage. By doing this, it makes sure that the operation that enters the execute stage gets the correct values for its source operands.

Forwarding of register values in ρ-VEX processor is performed in two stages. In the first stage, the source register addresses from decode pipeline stage are compared with the destination addresses from the execution stages of the pipeline and indexes of the matching addresses are identified. In the second stage of the forwarding logic, the register values in the matching register addresses are passed to their respective functional units in the execution stage of the pipeline. The destination register addresses and results of execute pipeline stages of all lanes are stored into the pipeline registers every clock cycle.

When an ICC operation is decoded, the forwarding logic of the pipelane:

1. bypasses the destination register number as the first operand when the operation is a `recv` operation. The updated content of the source register either from the pipeline register or from the register file is forwarded to the ICC FU as the first operand when the operation is an inter-cluster `send` operation.

2. forwards the source and destination clusters decoded from the operations as the second operand for both ICC operations.

In the clustered implementation of the ρ-VEX processor, the forwarding logic compares the register addresses of a given operation against the destination addresses of the execute pipeline stages within the same cluster. Each pipelane of the VLIW is assigned with a cluster identifier, $CLUSTER\_ID$. This identifier along with the *number of clusters* and $ISSUE\_WIDTH$ of the processor are used to identify with which portion of the execute pipeline results the comparison should be made. It computes these indexes as follows:

((CLUSTER_ID+1) * λ) - 1 downto CLUSTER_ID * λ

Where λ is *ISSUE_WIDTH/NUM_CLUSTER*.

For instance, in an (4 + 4)-way clustered ρ-VEX processor, the forwarding logic in the first four pipelanes ($CLUSTER\_ID = 0$) compares the register addresses from the decode unit with the first four destination register addresses from the execute pipeline results. Similarly forwarding logic in cluster 1, pipelanes {4 to 7}, compares the register addresses of their source operands with the last four destination register addresses of the execute pipeline.

3.4 HP VEX Compiler for Clustered Architectures: BUG and its FIX

The VEX C compiler supports code generation for clustered VLIW architectures. It generates operations to be executed in different clusters in one instruction bundle. A bug is found in the way the scheduling is performed in the compiler when compiling a code for (2 + 2)-way clustered VLIW architecture. The compiler scheduled 4 operations including operations with long immediate operand in one instruction bundle without considering a slot for the $follow$ operation. The modified assembler could not fit those operations in the available 4 issue slots as the operation with long immediate will have a $follow$ operation to store part of the long immediate operand, making it (4 + #$follow$ operations) in one instruction bundle. To better explain the problem, two examples are provided. The examples are taken from an assembly code generated by the VEX compiler.

- **Example 1**
  The problem with the instruction bundle in Listing 3.3 is that each `mov` operation has a long immediate operand. Thus, the total number of operations will be 4 + 4 $follow$ = 8 operations. Since the issue-width of the VLIW processor for which the code was compiled is 4, the issue slot scheduler of the assembler cannot place those 8 operations in 4 slots. Consequently, the assembler throws an error.

```
## case 1
##
        c0      mov $r0.9 = 528
        c0      mov $r0.11 = 660
        c1      mov $r1.13 = 1440
        c1      mov $r1.8 = 4384
 ;;
```

  Listing 3.3: All operations with long immediate operand

- **Example 2**
  The `ldb` operation in the instruction bundle, see Listing 3.4, has a long immediate operand, which implies that the total number of operations is 5, which cannot fit in the 4 issue slots.

```
## case 2
## b contains address of an array
```

```
        c0    shl $r0.5 = $r0.3, 31
        c0    ldb $r0.9 = ((b + 0) + 90)[$r0.2]
        c1    add $r1.2 = $r1.4, $r1.8
        c1    or  $r1.1 = $r1.2, 3
;;
```

Listing 3.4: `ldb` operation with long immediate operand

The *GNU binutils* assembler is modified to ensure that only ISSUE_WIDTH amount of operations, in this case 4, are issued in one instruction bundle. It handles this by inserting the end of instruction bundle `;;` right after the fourth operation and handling read after write data dependency issues between the operations in the instruction bundle. This dependency occurs when one or more operations in an instruction bundle are reading from the same register number of the same cluster and one other operation is writing to that register. The read operations should get the content of the register before the write operation updates it with a new value. Listing 3.4 shows this case, where the register number 2 of cluster 1 is being read by the fourth operation and operation 3 write to it. A simple split after the fourth operation will not solve the bug because it will cause data coherency problem. Table 3.7 describes the algorithm used to fix all the bugs in the assembly generated by the HP compiler.

The variable *exception_flag* is used to signals the assembler that end of instruction bundle is reached due to an exception. The assembler will then schedule the operations in that instruction bundle into the issue slots. This technique solves the scheduling bug found in the VEX C compiler. In the issue slot scheduling stage of the assembler, the *valid* flag for each operation is checked which indicates if an operation is a valid operation and should be considered for scheduling. If an operation has *valid* with a value of 1, then the corresponding operation is scheduled in one of the available issue slots; otherwise the operation is skipped. This way the operations that were invalidated in case 3 and case 4 above will not be considered for scheduling by the assembler.

Table 3.8 shows the number of occurrences of the bug fixed by using the algorithm presented above for different benchmark applications compiled using HP VEX compiler for a (2 + 2)-way clustered VEX VLIW processor.

Algorithm: FIX_BUG: Operations With Long Immediate Operand

INPUT:

operations: Operations to be issued
current_syllable: Count of the number of syllable/operations in the current instruction bundle
ISSUE_WIDTH: The issue width of the clustered ρ-VEX VLIW processor

begin

switch *current_syllable, operation*

case $ISSUE\_WIDTH$ - 1 operations + Inter-cluster `mov` operation
i- Set exception_flag variable
ii- Insert the end of instruction bundle
iii- Reset exception_flag variable
iv- Issue the inter-cluster mov operation in a new instruction bundle

case $ISSUE\_WIDTH$ - 1 operations + 1 operation with long immediate operand
i- Set exception_flag variable
ii- Insert the end of instruction bundle
iii- Reset exception_flag variable
iv- Issue the operation with long immediate operand in a new instruction bundle

case $ISSUE\_WIDTH$ operations + 1 operation with read after write hazard if split
i- Identify the operation that writes to one of the registers
ii- Copy write operation to temp
iii- Invalidate write operation so that it will not be issued in the current instruction bundle, [valid = 0]
iv- Set exception_flag
v- Insert the end of instruction bundle
vi- Reset exception_flag variable
vii- Issue the write operation and the read operation in a new instruction bundle

case $ISSUE\_WIDTH$ operations + 1 Operation with Long Immediate operation with read after write hazard if split
i- Identify the operation that writes to one of the registers in the operation to be splitted
ii- Copy write operation to temp
iii- Invalidate write operation and its corresponding follow operation so that they will not be issued in the current instruction bundle, [valid = 0]
iv- Set exception_flag
v- Insert the end of instruction bundle
vi- Reset exception_flag variable
vii- Issue the write operation, follow operation and the read operation in a new instruction bundle

end

Table 3.7: Algorithm used to fix the bug of the VEX C compiler.

Benchmark	Number of occurrences of the bug
MATRIX	17
ADPCM	102
JPEG	41
SOMA	0
FLOATLIB	45
DFT	1
CJPEG	18
QURT	3
G3FAX	21
UCBQSORT	2
BCNT	15
BLIT	29
COMPRESS	63
CRC	2
DES	29
ENGINE	4
FIR	0
POCSAG	8
V42	56

Table 3.8: Number of occurrences of the bugs fixed for different benchmark applications compiled for a (2 + 2)-way clustered ρ-VEX processor architecture.

3.5 Clustered ρ-VEX Processor with Dedicated Issue Slot ICC Model

In addition to the ICC model which is supported by the ISA, the copy operation ICC model, a dedicated issue slot inter-cluster communication model is implemented for the 4-issue clustered ρ-VEX processor.

In this ICC model, a separate pipelane is dedicated to FUs that only perform inter-cluster data transfers. Figure 3.5 shows a four issue clustered ρ-VEX VLIW processor with 4 issue slots dedicated to inter-cluster communication. The inter-cluster path is a set of multiplexers to forward the register values written by the source cluster to the destination cluster. By dedicating ipipelanes for inter-cluster communication, more scheduling freedom is obtained for the regular operations at a cost of increased instruction bundle width and code size.

The format of the instruction bundle for a 4-issue clustered ρ-VEX processor with dedicated issue ICC model is depicted in Figure 3.6. Issue slots {0, 1, 4, 5} can be used by the regular operations while the other slots are dedicated for ICC operations.

The VEX C compiler is a closed source compiler and it only supports copy operation

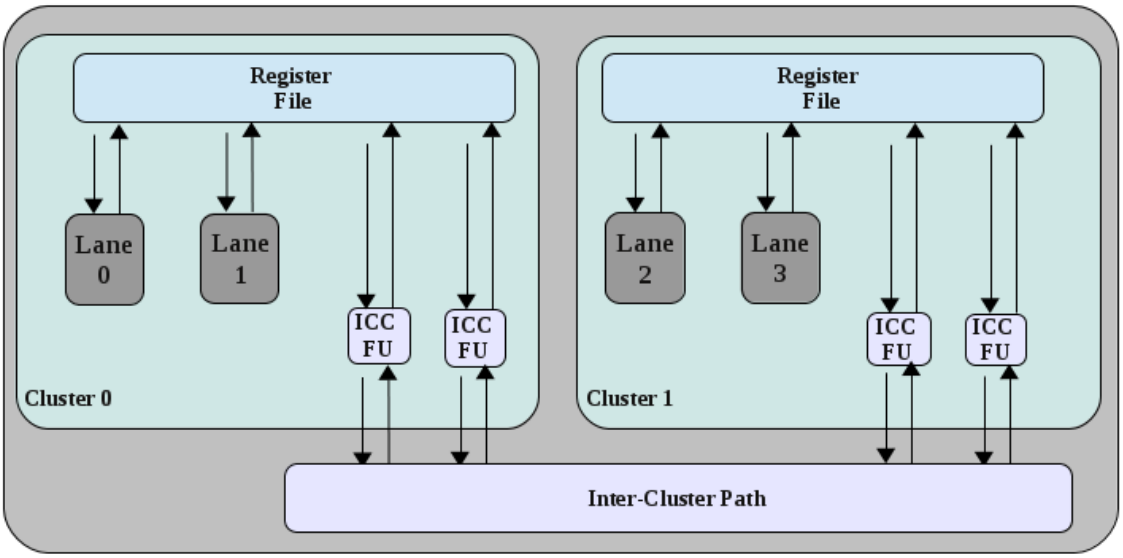


Figure 3.5: Dedicated Issue Slot Inter-Cluster communication Model.

Lane0	Lane 1	ICC	ICC	Lane 2	Lane 3	ICC	ICC

Figure 3.6: Instruction Bundle Format for Dedicated Issue Slot Inter-Cluster Communication Model.

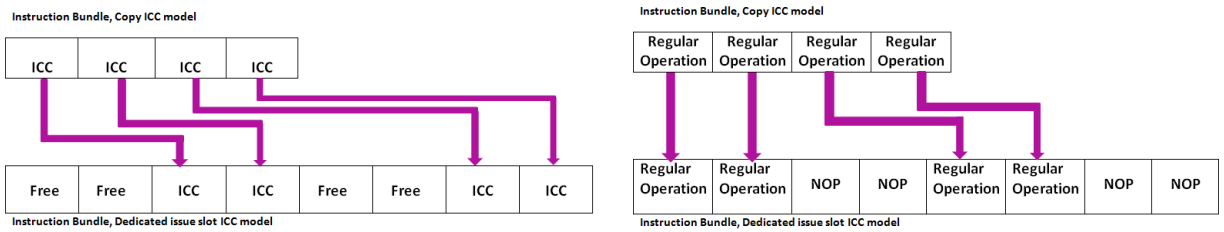
ICC model, thus its scheduling algorithm could not be altered to schedule a code for clustered VLIW architectures with dedicated issue slot ICC model. As a result, the same assembly code generated by the VEX compiler is used for both ICC models. The issue slot mapping is handled by the assembler.

The modified *GNU binutils* assembler is provided with the configuration containing the functional units that are present in each issue slot. For the clustered ρ-VEX processor with dedicated issue slot ICC model, the issue slots {2, 3, 6, 7} contain only inter-cluster communication functional unit. Thus, the configuration of functional units for the 4-issue 2 cluster ρ-VEX processor is:

(2 + 2)-way Clustered, Dedicated ICC model	--config 0B07101007031010

Pipelanes with *0x10* configuration contain only inter-cluster communication functional units. The mapping of operations in an instruction bundle from (2 + 2)-way clustered ρ-VEX processor with a copy operation ICC model to dedicated issue slot ICC model is depicted in Figure 3.7. The ICC operations are moved to their dedicated slots and the regular operations are scheduled in the other slots. When an instructions bundle contains only ICC operations, only four of the slots are occupied; the other slots are free

and will either contain NOP operation or a $follow$ operation for an operation that had a long immediate operand.



(a) ICC Operations (b) Regular Operations

Figure 3.7: Issue slot scheduling in 4-issue ρ-VEX processors with dedicated issue slot ICC model.

3.6 Conclusion

This chapter presented the detailed design and implementation of the design-time reconfigurable clustered ρ-VEX VLIW processor. The (2 + 2)-way and (4 + 4)-way clustered organization are the two ρ-VEX architectures discussed.

Two inter-cluster communication models, the copy operation ICC model and dedicated issue slot ICC model, are implemented on the clustered ρ-VEX processors. The VEX ISA supports copy operation ICC model by providing explicit pair of operations (`send` and `recv`) for transferring local register values between the clusters. In addition to that, a model that dedicates pipelanes for inter-cluster communication is implemented. The structure and instruction bundle layout for this ICC model are presented. The modifications made to the compiler toolchain, the ρ-VEX processor core, the syllable of branch operations and the borrowing scheme for the clustered ρ-VEX processor are discussed. A bug found in the scheduling of the VEX compiler and the algorithm used to fix it are also presented.

Run-time Adaptable Clustered ρ-VEX Processor

4

The 2-4-8-issue run-time adaptable ρ-VEX processor [2] is a reconfigurable multi-core system that can change its issue width and number of general registers at run-time as per the requirement of the application to gain performance and save dynamic power consumption. To optimally reduce power consumption, it uses a technique called *clock gating* to shut down some of the ρ-VEX processor cores that are not in use. In this chapter, the introduction of clustered organization to this run-time adaptable ρ-VEX VLIW processor is presented to reduce the area and power overhead of the processor.

In Section 4.1, the modifications made to the run-time adaptable processor core and the compiler toolchain are presented. In Sections 4.2 and 4.3, the design and implementation of the 8-issue and dual core 4-issue clustered VLIW processors are discussed. The compilation steps to be followed to generate a single instruction and data binaries containing multiple programs with different VLIW architectures are presented in Section 4.4.

4.1 Design of Clustered Run-time Reconfigurable ρ-VEX VLIW Processor

The 8-issue and the two 4-issue run-time reconfigurable ρ-VEX VLIW processors are adapted to support clustered organization in the processor. The VEX compiler and the modified *GNU binutils* for clustered ρ-VEX architectures that are presented in the design-time reconfigurable ρ-VEX processor, Section 3.2, are used in these clustered reconfigurable architectures.

Figure 4.1 shows a block diagram of the cluster organization of the run-time reconfigurable ρ-VEX processor. The new components added and parts of the baseline processor that were modified will be presented as follows:

The Decoder Unit

The decoder unit of the 2-4-8-issue run-time reconfigurable ρ-VEX core is modified to decode inter-cluster `send` and `recv` operations and extract the source and destination cluster from the syllable. For the `send` operation, the source register number is read either from the local register file or from the pipeline forwarding register in cases where the result was not yet written back to the register file. In additon, as a result of the change of the syllable of some of the conditional branch operations, the decoder unit have to extract the cluster number, branch register number and the branch offset address (16 bits) from the syllables of `br` and `brf` operations.

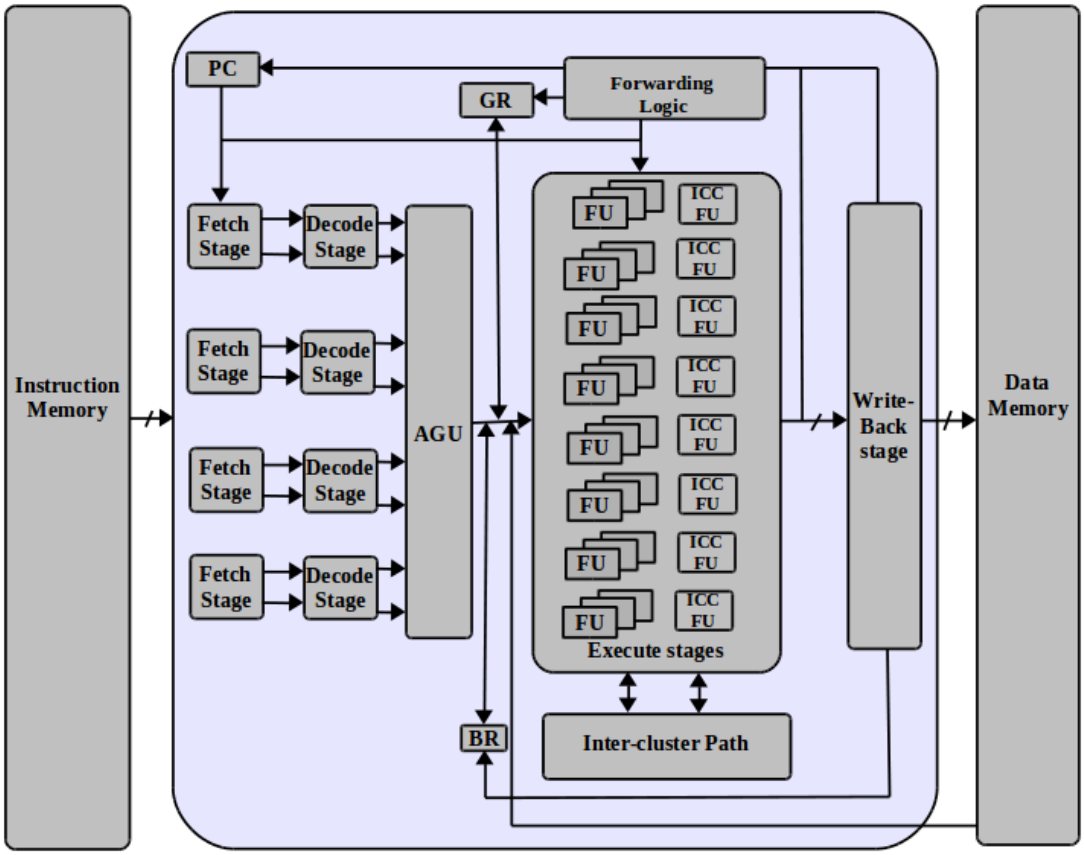


Figure 4.1: The 2-4-8-issue run-time adaptable clustered ρ-VEX processor.

Inter-cluster Communication Functional Units

ICC FUs are added in each pipelane of the run-time adaptable VLIW processor and inter-cluster path bypass network is designed. Copy operation ICC model is used as an inter-cluster communication model. The inter-cluster communication functional units in all pipelanes are connected to the inter-cluster path using 1 read and 1 write port. After the register data of the source operand is available in the functional units in the execution stage of the pipeline, the ICC functional unit attaches a *header* (destination register number, source and destination cluster) to the register data and writes it to its write port of the inter-cluster path. The ICC FU in the destination cluster writes the same header to the inter-cluster path write port. The inter-cluster path compares the headers from the destination cluster with that of source clusters and forwards the data on a match. The mode of operation of the processor is used to determine the sections of the inter-cluster path the headers should be compared against. The register data is then forwarded to the local register file. Inter-cluster communication takes one cycle.

The Forwarding Logic

The forwarding logic of each pipelane compares the register number of the source operands of the operation with the destination addresses of the results from the pipeline register within the same cluster. By using the cluster number identifier and pipelane number that are assigned to each pipelane, it is made sure that the forwarding logic uses the correct indexes from the pipeline registers for the comparison with the source register numbers. When a match is found, the content of the matching register number from the pipeline registers is forwarded to the corresponding functional unit. Details of this implementation are discussed in Section 3.3.3.

The New Control Signals

New signals are introduced to the baseline 2-4-8-issue run-time ρ-VEX processor to allow run-time reconfiguration between clustered and single cluster architectures. One of these signals is *num_cluster* which specifies the number of clusters in the architecture. The *interclust* is another signal added into the design to flag if the VLIW organization of the processor is clustered or not.

The Multiplexer Unit for Data Memory

The read and write addresses and data of the Load/ Store units located in each cluster have to be connected to their respective read and write ports of the data memory. A multiplexer unit is added to the design to forward the read and write addresses and data from the MEM units to the input and output ports of the data memory based on the mode of operation of the processor which are specified by the control signals.

The Link Register

Depending on the mode of operation of the processor core, the forwarding logic of the link register compares and writes the data in the read ports of the link register. When the core is in one 8-issue processor or in a (4 + 4)-way clustered VLIW processor mode, the forwarding logic checks the write enables of all the eight pipelanes for a '1' and forwards the data to the read port of all lanes from the write port of the lane. In a two 4-issue single cluster core or two (2 + 2)-way clustered core mode, there are two link register and the forwarding logic checks the write enables of the first and next four pipelanes. Each 2-issue core in the four 2-issue core mode of the processor have a separate link register and thus the writes enables of every two pipelanes is checked for a '1'. When a match is found, the forwarding logic copies the data into the read ports of the two pipelanes of the core.

The Address Generation Unit

The BlockRAMs which provide up 256 32-bit registers have been used to create the GR register file in the single cluster organization of the 2-4-8-issue run-time reconfig-

urable core. The design of the register file is adapted in such a way that they can still be used for multi cluster organization without loosing run-time reconfigurability feature of the processor. To address the 256 registers, 8 bits are required. The AGU is modified to subdivide the registers into the clusters. The 64 registers defined by the VEX architecture are divided into the number of clusters and the AGU takes care of the mapping the register addresses decoded from the different clusters into the correct addresses in the BRAMs. The register addresses of a clustered ρ-VEX architecture are represented using 5-bits (32 registers). Subsequently, the AGU converts these 5-bit register addresses into 8-bit based on the mode of operation of the processor and the pipelane number.

The branch unit of cluster 0 can access the register file of other clusters. Thus, an address generation unit that control the mapping of the branch register file from the decoder units of the clusters is designed. Based on the *issue_ctrl*, *interclust* signal and the cluster identifier of the branch operations, the register number extracted from the operation is accessed from the correct dedicated region in the branch register file. The branch register number is represented using 3-bits (8 registers per cluster) and the BR register file required 5-bits. As a result, the BR AGU is adapted to translates the addresses and handles the register accesses from cluster 0 into branch register files of other clusters.

4.2 The (4 + 4)-way Clustered ρ-VEX VLIW Processor

The eight pipelanes of the 2-4-8-issue run-time adaptable VLIW processor are grouped into clusters of two. In VEX clustered organization, each cluster should possess a load-/store unit, thus the first data memory of 2-4-8-issue ρ-VEX processor is multi-ported to allow access from the load/store units of the clusters, see Figure 4.2. The other three data memories are gated off to reduce their dynamic power dissipation. In this 8-issue clustered architecture, the read and write addresses and data of the load/store unit in pipelane {7} is forwarded by the multiplexer unit to read and write port 0 of the data memory. Similarly, the read and write addresses and data of the MEM unit in pipelane {1} are connected to the corresponding port 1 of the the first data memory.

To switch the mode of operation of the core to (4 + 4)-way clustered processor organization at run-time:

- The signal *issue_ctrl* should be set to “00”,
- The signal *num_cluster* should be assigned a value of 2 and
- The *interclust* signal must be set to *true*.

When the processor is in this mode of operation, the instruction memories should be loaded with the code compiled for (4 + 4)-way clustered ρ-VEX architecture. The processor has to be preempted with a new program counter so that it executes the program.

The AGU of GR register file splits the 64 registers into two so that the first 32 registers are assigned to cluster 0 and the register numbers {32 - 63} are utilized by the second

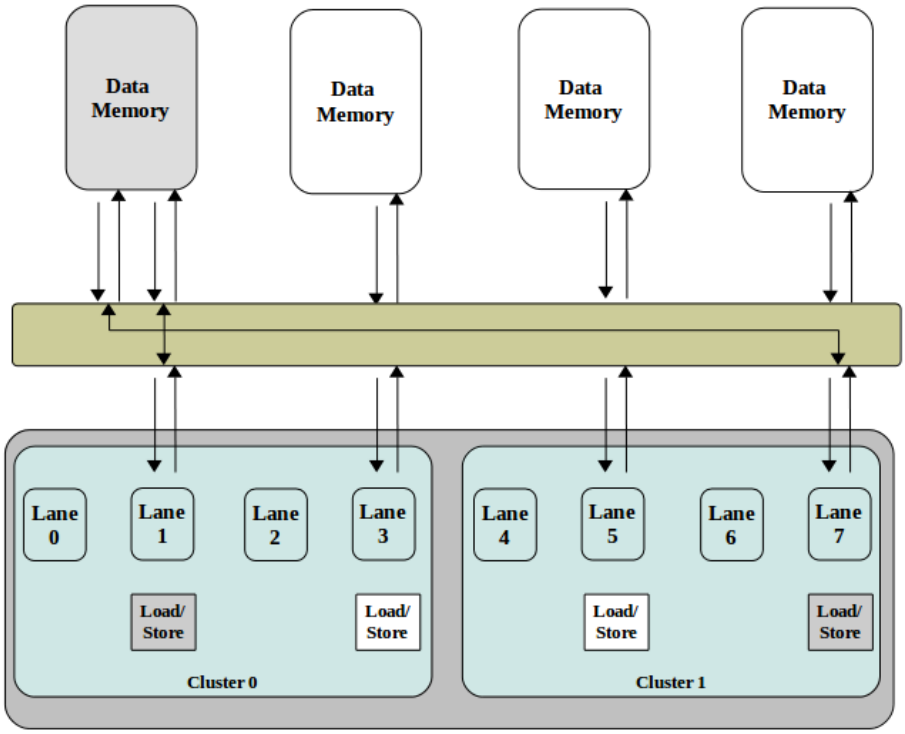


Figure 4.2: The 8-issue run-time adaptable clustered ρ-VEX core.

cluster. It extends the 5-bit register addresses of the operands with 3 more bits that will identify the corresponding region in the BRAMs. Fig 4.3 shows the mapping used by the AGU for the GR register file in the (4 + 4)-way clustered ρ-VEX architecture. After the decoder unit extracts the source register operands of an operation, the register addresses pass through the address generation unit and then the content of those register addresses are fetched. The destination register addresses are also translated by the AGU unit after they are decoded from the syllable of the operations. The source register operand addresses can be the same between different clusters, but after the addresses are mapped to their corresponding region in the BRAM by the AGU, two source/ destination registers can have the same address if and only if they are from the same cluster.

The address generation unit of the branch register file maps the first 8 1-bit branch registers to cluster 0 while cluster 1 is assigned with branch registers {9-15}. When pipelane {0} decodes a branch operation with a source register operand from another cluster, the AGU unit ensures the correct branch register address from the specified cluster is accessed.

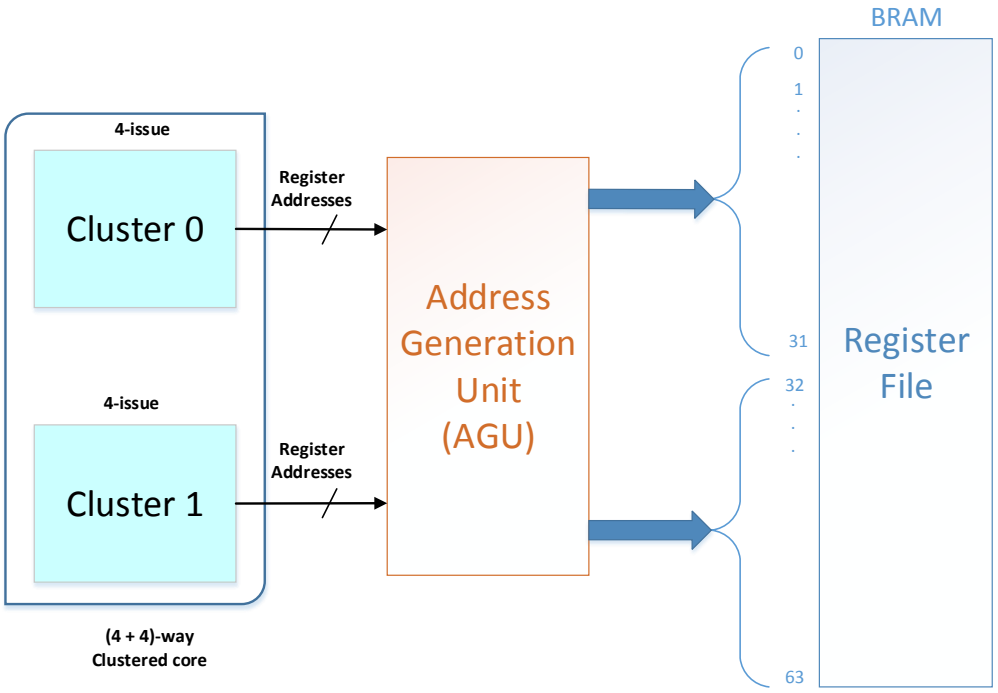


Figure 4.3: The address generation unit for GR register file in (4 + 4)-way clustered run-time adaptable ρ-VEX core.

4.3 The Dual-core (2 + 2)-way Clustered ρ-VEX VLIW Processor

Each 2-issue core is considered as a cluster in this clustered ρ-VEX VLIW processor organization. Figure 4.4 depicts the overall organization of the clustered 4-issue ρ-VEX VLIW processor or (2 + 2)-way clustered core. In addition to the introduction of the ICC function unit and the inter-cluster bypass network to the design, two of the data memories are multiported so that the load/store units in all clusters can have access to a data memory.

The read and write input and outputs of port 0 of the first data memory are connected to load/store unit in pipelane {3} by the multiplexer unit while that of the second data memory are multiplexed to the load/store unit in pipelane {7}. Similarly, the read and write addresses and data of load/store units in pipelanes {1,5} are forwarded to port 1 of the first and third data memories respectively.

The 2-4-8-issue core can be reconfigured to this processor architecture at run time by setting the signals;

- *issue_ctrl* to "01",
- *num_cluster* to 2 and

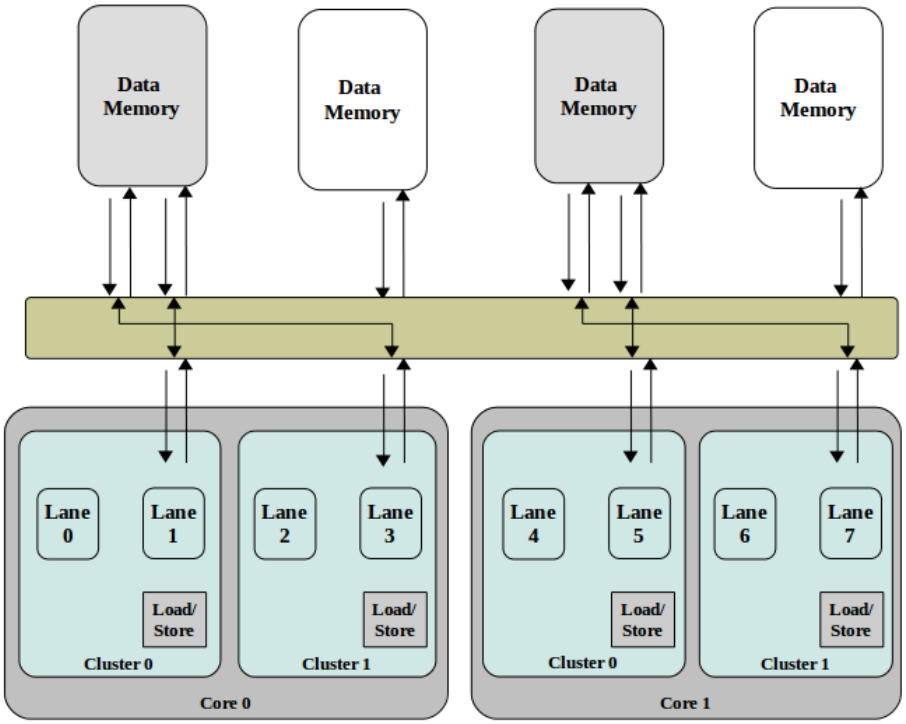


Figure 4.4: Two 4-issue run-time adaptable clustered ρ-VEX cores.

- *interclust* signal to *true*.

The branch units of this processor are located at pipelanes {0, 3}. The first control unit controls the program counter of the application running at the first core while the later controls the flow of execution of an application running at second 4-issue clustered ρ-VEX core.

The branch register file of the two (2 + 2)-way clustered ρ-VEX processors consists of 16 1-bit registers where each cluster is assigned 8 registers. The branch units of the cores that are placed in cluster 0 of both cores can access the branch register file of the other clusters. The AGU of the branch register file controls the mapping of the source and destination addresses into the region dedicated for the cluster in the branch register file.

Figure 4.5 depicts the AGU unit for the GR register file in (2 + 2)-way clustered ρ-VEX cores. Every decoder unit in each pipelane forwards 5-bit GR register number to the AGU unit which maps it to its corresponding region in the BRAMs. 128 registers from the GR register file are utilized in this processor mode. Each cluster makes use of 32 registers. Cluster 0 of the first (2 + 2)-way clustered core uses register numbers from {0 - 31} while cluster 1 of the first core is assigned registers {32-63}. Register addresses {64-95} are dedicated to cluster 0 of the second (2 + 2)-way clustered core and registers {96-127} are utilized by cluster 1 of the second (2 + 2)-way clustered core.

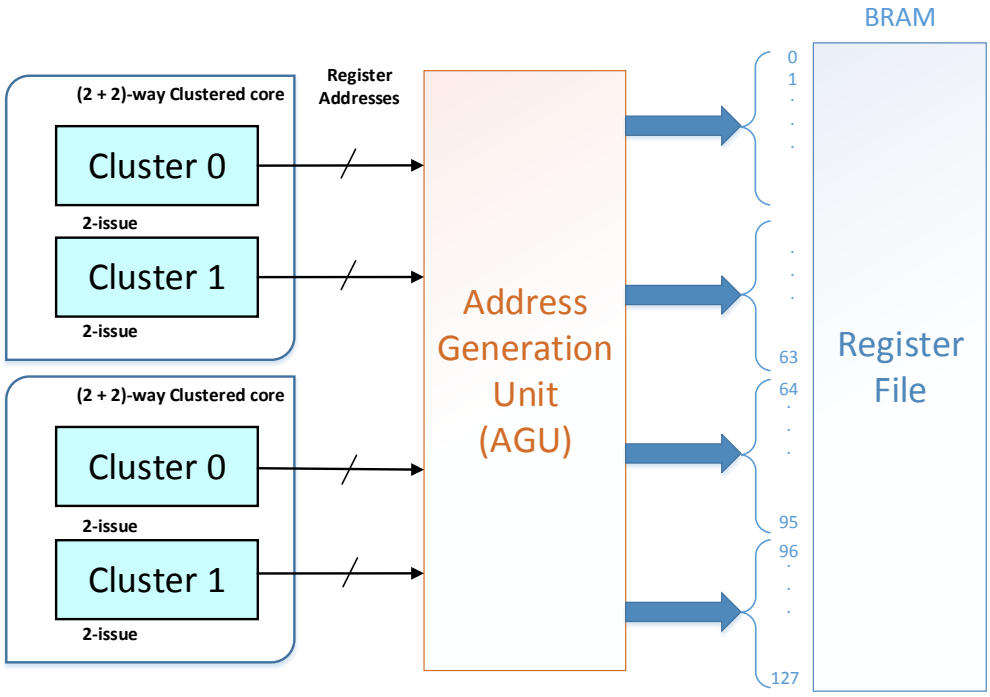


Figure 4.5: The address generation unit for GR register file in (2 + 2)-way run-time adaptable clustered ρ-VEX cores.

4.4 Run-time Dynamic Reconfiguration of the 2-4-8-issue VLIW processor

The 2-4-8-issue processor can be reconfigured at run time between different VLIW processor architectures, clustered and single cluster organizations. Figure 4.6 shows the flow of actions to be followed to generate hex files containing two programs with different VLIW architectures. To execute two or more programs with different processor architectures, the programs have to be separately compiled and assembled using their corresponding machine configuration models, borrowing scheme and FU configuration to create the object files. The *_start_A.s* and *_start_B.s* assembly files are program loaders that contains a call to the *main* functions of programs A and B respectively and a `stop` operation. The object files of the programs are linked together to generate one executable file. The *elf2vhd* tool is then used to extract the instruction and data hex files from the executable file which will be loaded to the instruction and data memories of the run-time reconfigurable processor. The programs must have different function names (main and others functions) and global variable names.

After the processor completes the execution of the first program, it sets the *done*

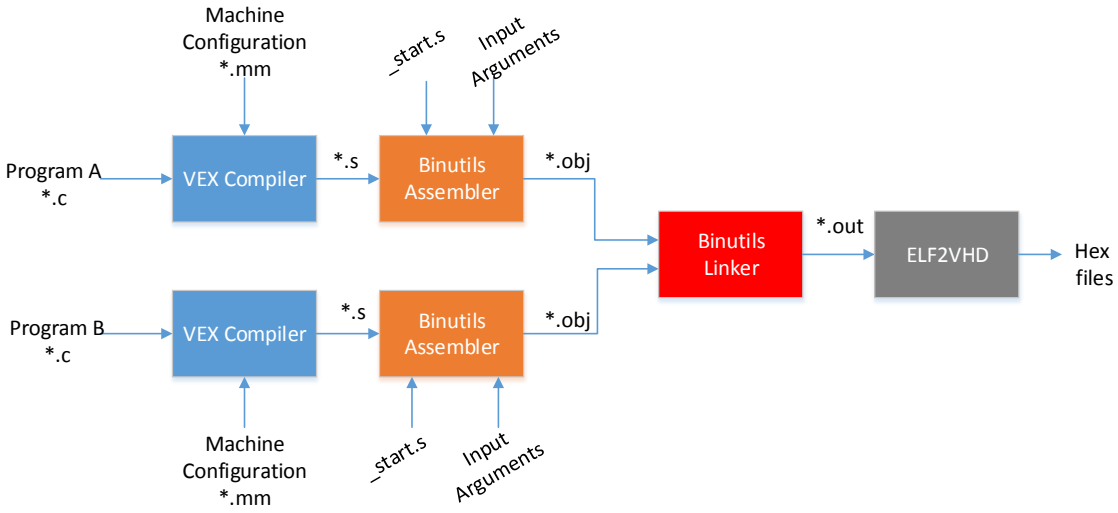


Figure 4.6: Compilation flow in (2 + 2)-way run-time adaptable clustered ρ-VEX cores.

signal to high, which is used to dynamically reconfigure the core to another processor architecture at run time by changing the control signals and interrupting the processor with the start address of the next program.

4.5 Conclusion

This chapter presented the introduction of clustering organization to the 2-4-8-issue run-time adaptable ρ-VEX VLIW processor. Inter-cluster communication functional unit in each pipelane and inter-cluster bypass network are designed and implemented. The design supports the VEX clustered architecture, thus explicit inter-cluster copy operations are used as inter-cluster communication model. `send` and `recv` inter-cluster operations are scheduled in the source and destination clusters by the assembler and it takes one cycle to transfer the register value between the clusters.

The single cluster organization of the 8-issue run-time reconfigurable ρ-VEX core and the dual 4-issue cores are clustered into clusters of two. A multiplexer unit is added to forward read and write addresses and data from the load/store unit of the clusters to their respective data memory based on the mode of operation of the processor. The detailed architectures of the two run-time reconfigurable clustered organizations, the two (2 + 2)-way and the (4 + 4)-way clustered processors, are presented. The compiler toolchain adaptations discussed in Section 3.2.1 has been used for this run-time reconfigurable processor. To the best of our knowledge, this makes our 2-4-8-issue dynamically reconfigurable processor is the only run-time reconfigurable clustered processor with a complete compiler toolchain.

5 Results

The design-time and run-time reconfigurable clustered ρ-VEX VLIW processors are designed and implemented in Chapters 3 and 4. The two (2 + 2)-way and (4 + 4)-way clustered VLIW architectures are the two clustered organizations implemented. This chapter evaluates the performances of these processors.

In Section 5.1, the experimental setup such as the hardware, tools and different benchmark applications used for the evaluation are presented. Experimental results, the area and power overhead, and cycle count analysis, on the design-time and run-time reconfigurable clustered ρ-VEX processor are discussed in Section 5.2. The results of the design-time reconfigurable clustered ρ-VEX processor with dedicated issue slot ICC model are presented in Section 5.3.

5.1 Experimental Setup

In this section, the experimental setups used for the evaluation of clustered organization of the ρ-VEX processors such as the benchmark applications, the hardware platform and software tools are presented.

5.1.1 Benchmark applications

The *powerstone* benchmark is used to measure the performance of the clustered ρ-VEX processors. It consists of different applications that are suitable for embedded low-power processors. It includes applications such as paging, imaging, faxing and signal processing. Table 5.1 presents the category and description of the benchmark applications used for the comparison of performance between the clustered organization and baseline ρ-VEX processors.

5.1.2 Hardware Platform and Software Tools

The design-time and run-time reconfigurable clustered ρ-VEX softcore processors designed are implemented into a Xilinx Virtex-6 XC6VLX240T-1FF1152 FPGA, which is available on the ML605 Evaluation Board [40] and different benchmark applications are run on it. The Xilinx ISE design suit [41] version 14.3 is used for the synthesis and bitstream generation from the designs. A Universal Asynchronous Receiver/Transmitter (UART) is added to the design to print the number of cycles and the result section from the data memory. The synthesis device utilization summary generated by the ISE design suit displays the area usage of the processors in terms of the total number of slices, number of slice LUTs, number of slices used as memory and BlockRAMs.

Benchmark	Category	Description
ADPCM	Audio	Adaptive differential pulse-code modulation (encoder and decoder)
BCNT	Image	Bit count
BLIT	Image	Graphics application
COMPRESS	Image	A Unix image compression utility
CRC	Network	Cyclic redundancy check
CJPEG	Image	Forward discrete fourier transform of an image
DES	Network	Data Encryption Standard
ENGINE	Audio	Engine control application
FIR	Filter	Float FIR filter
G3FAX	Image	Group three fax decode (single level image decompression)
JPEG	Image	JPEG 24-bit image decompression standard
MATRIX	Vector	Matrix multiplication
POCSAG	Paging	POCSAG communication protocol for paging application
QURT	Audio	Square Root calculation using floating point
UCBQSORT	Audio	U.C.B. Quick Sort
V42	Audio	Modem encoding/decoding

Table 5.1: Description of benchmark applications used for measurement.

The dynamic power estimation and analysis software tool for programmable logic designs from Xilinx, *XPower analyzer tool* [42], is utilized to evaluate the power consumption of the new design and compare it against the single cluster organization of both the design-time and run-time reconfigurable ρ-VEX cores.

For hardware simulation, *ModelSim* [43] is used. The hardware design of both design-time and run-time reconfigurable clustered ρ-VEX processors are simulated and their correctness is confirmed using different benchmark applications. After which the softcore processors are synthesized, the bitstreams are generated and programmed on the ML605 FPGA board using the *Xilinx iMPACT* tool.

5.2 Clustered ρ-VEX processor with Copy Operation ICC Model

Different measurements and comparisons are conducted between the clustered and non-clustered design-time and run-time reconfigurable ρ-VEX VLIW processors. The

performance, code size, and cycle and operations count of the benchmark applications, area usage and power consumption of the processors are discussed in the next subsections. In addition, energy delay product (EDP) metric is calculated for the processors..

5.2.1 Performance Analysis: Maximum Clock Frequency

The design-time reconfigurabe clustered ρ-VEX processor improved the clock frequency by up to 1.55×. The single cluster 4-issue and the (2 + 2)-way clustered ρ-VEX processors run at *137 MHz* and *195 MHz* respectively while the single cluster 8-issue and the (4 + 4)-way clustered ρ-VEX processors run at a maximum clock frequency of *100 MHz* and *155 MHz* respectively. This drastic speedup in the clock frequency is because of the reduction of long data paths and complex multiplexers from the design of the baseline ρ-VEX processor by the introduction of clustering. On the other hand, the baseline and the run-time reconfigurable clustered processor operate at the same clock frequency of *110 MHz*.

5.2.2 Total Cycle Count and Speedup

The *Cycle count* of a program is the number of clock cycles the processor took to execute the program completely. The cycle count of the *powerstone* benchmark applications is measured for the (2 + 2)-way and (4 + 4)-way clustered ρ-VEX VLIW processors and it is compared with the cycle count of the benchmarks on the following two single cluster organizations:

- 4- and 8-issue ρ-VEX VLIW processors with 1 load/store unit (default ρ-VEX architecture),
- 4- and 8-issue ρ-VEX VLIW processors with 2 load/store unit.

The clustered ρ-VEX processors have two load/store units (one in each cluster), thus the comparison of cycle count is performed against single cluster ρ-VEX processor with one and two load/store units.

The cycle count of the benchmarks on clustered processors and the default single cluster organization of the ρ-VEX processors are measured from the UART print of the FPGA while the cycle count of the single cluster ρ-VEX processors with two memory unit are measured by using the *xSTsim* simulator by STMicroelectronics because it is not implemented in ρ-VEX.

The Figure 5.1 shows the speedup of *powerstone* benchmark applications on (2 + 2)-way clustered ρ-VEX processor and 4-issue ρ-VEX processor with 2 load/store unit compared to baseline 4-issue ρ-VEX processor. It can be observed from the chart that for applications with higher ILP, such as vector operations (*soma*, *matrix* and *adpcm*), the speedup in the (2 + 2)-way clustered ρ-VEX processor is larger than in the single cluster architecture with one load/store unit. This is mainly because of the additional load/store unit in the clustered processor and the minimal data dependency between the clusters leading to reduced inter-cluster communication. The *matrix* application

has the maximum speedup in the cycle count of 1.556×. On the other hand, the *pocsag* benchmark has the maximum increase in cycle count on the clustered processor with respect to the 4-issue ρ-VEX processor which resulted in a speedup of 0.697×.

For most of the benchmark applications, the clustered organization of the ρ-VEX processor takes more clock cycles than the 4-issue ρ-VEX processor with 2 load/store units. The reason for this increase in the cycle count is the addition of inter-cluster communication operations in the clustered organization of the ρ-VEX processor. A minimum speedup of 0.672× is obtained by *pocsag* application on clustered ρ-VEX processor.

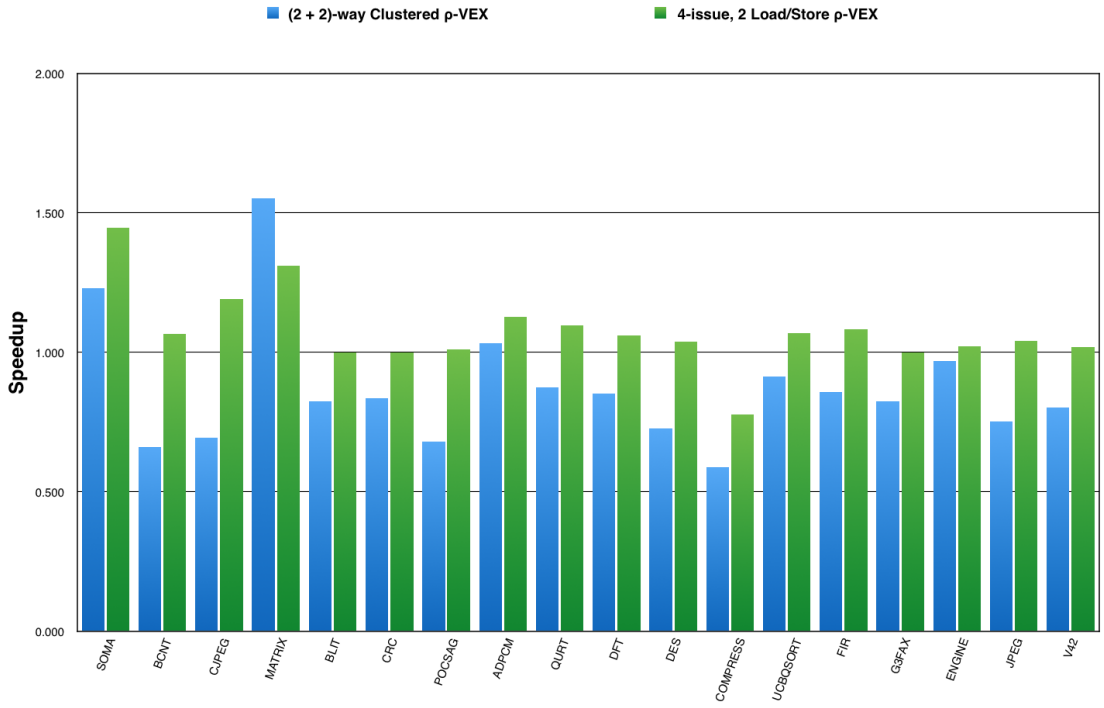


Figure 5.1: Speedup for *powerstone* benchmark applications on (2 + 2)-way clustered ρ-VEX processor and 4-issue with 2 load/store unit ρ-VEX processor compared to the baseline 4-issue ρ-VEX processor.

The speedup of the benchmark applications on the different organizations of (4 + 4)-way clustered ρ-VEX processor and 8-issue ρ-VEX processor with 2 load/store unit compared to 8-issue ρ-VEX processor are depicted in Figure 5.2. Benchmark applications with high ILP such as *matrix*, *dft*, *adpcm* and *des* have higher speedup on the (4+4) way clustered ρ-VEX processor. The *matrix* benchmark achieved a maximum speedup of 3.04×. For the rest of the benchmark applications, the cycle count slightly increased on the clustered processor. A minimum speeddown of 0.843× is obtained for the *jpeg* application.

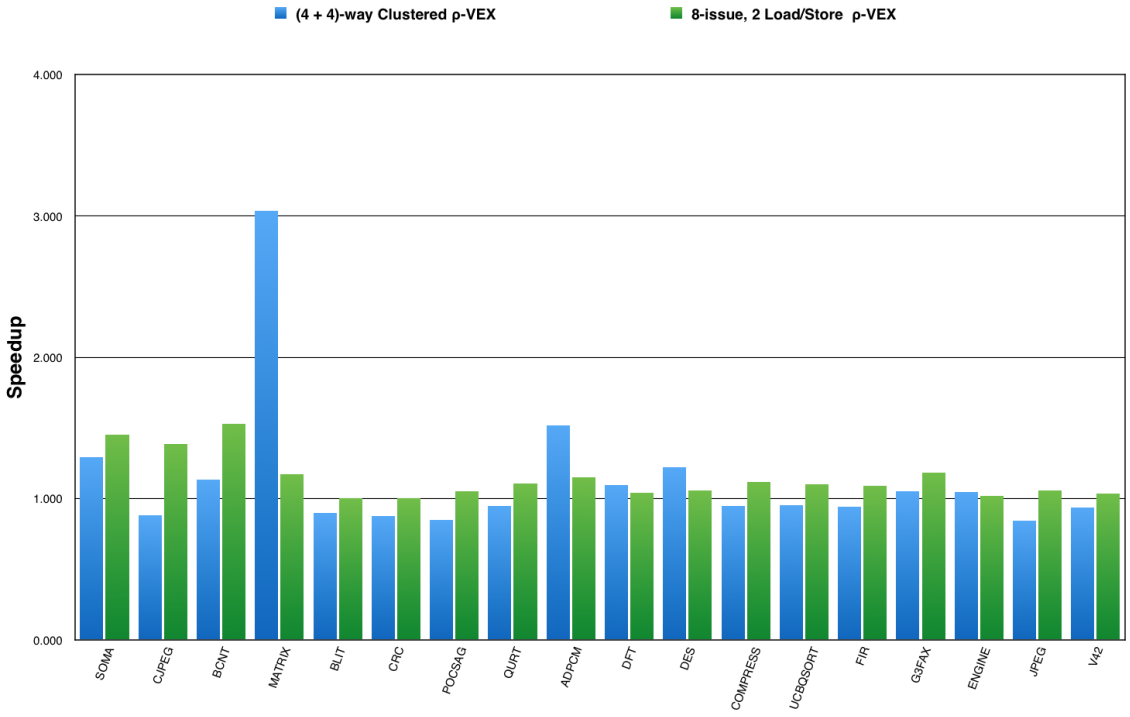


Figure 5.2: Speedup for *powerstone* benchmark applications in (4 + 4)-way clustered ρ-VEX processor and 8-issue with 2 load/store unit ρ-VEX processor compared to the baseline 8-issue ρ-VEX processor.

Generally speaking, the cycle count of the benchmarks on the single cluster organization of ρ-VEX processor with one memory unit is larger than with two memory units for both 4-issue and 8-issue machines. In the ρ-VEX processor with two load/-store units, the compiler takes advantage of the extra memory unit and schedules two memory operations in one instruction bundle to generate code with smaller cycle count. The magnitude of the speedup depends on the nature of the benchmark application, the number of memory accesses it makes in the program. The more memory accesses the benchmark application has, the more the processor architecture would benefit from having two load/store units.

5.2.3 Operations Count

For a given application, identifying the total number and types of operations executed in VLIW processor architectures with different issue width helps to:

- better understand the ILP of the application or suitability of the application for the underlying processor architectures and,
- evaluate the efficiency of the scheduling algorithm of the compiler in exploiting the

instruction level parallelism,

- understand how many useful operations the processor executed and how many times the functional units in the pipelanes were doing nothing, executing (NOP operations).

Table 5.2 shows the total number of operations and the ratio of increase in the number of operations for all the benchmark applications on a 4-issue (2 + 2)-way and 8-issue (4 + 4)-way clustered ρ-VEX processor. By increasing the issue width of the VLIW processor, the number of operations executed can be reduced for applications with high ILP keeping the total number of operations almost the same. *matrix*, *bcnt and des* are vector operation benchmarks with smallest ratio of increase in the total number of operations between the 4-issue and 8-issue clustered processors. Those applications also have smaller cycle count compared with the corresponding single cluster organizations. The ratio of increase in the number of operations executed from (2 + 2)-way to (4 + 4)-way clustered ρ-VEX processor for the *matrix* benchmark application is only 1.148×. On the other hand, application such as *crc, engine, ucbqsort*, which have less ILP, almost doubled the total number of operations as the issue with of the ρ-VEX processor is increased from four to eight; of course the efficiency of the scheduling algorithm of the compiler, to extract ILP from a program, is also a factor.

Benchmark	**Total number of operations**		**Rate of increase**
	(2+2) way Clustered	**(4+4) way Clustered**	
BCNT	2980	3392	1.138
MATRIX	10544	12104	1.148
BLIT	51056	92072	1.803
CRC	60668	114968	1.895
ADPCM	90116	141384	1.569
POCSAG	102592	158896	1.549
QURT	124412	210376	1.691
DES	248536	362824	1.459
COMPRESS	438040	738640	1.686
UCBQSORT	833220	1638800	1.967
FIR	2141596	3592288	1.677
G3FAX	2594968	4632736	1.785
ENGINE	2721732	5305160	1.949
JPEG	7721752	12593296	1.631

Table 5.2: Total number of operations and rate of increase on 4- and 8-issue clustered ρ-VEX processor for *powerstone* benchmark applications.

All VEX operations executed in all pipelanes of the clustered ρ-VEX processors with different issue widths for the benchmark applications are logged into files and a program

is written in *C* to analyze the log files by counting the total inter-cluster communication operations, total number of *NOP* operations and overall percentile of those operations.

Figure 5.3 displays a chart with percentile of operations for *powerstone* benchmark applications that are executed on the (2 + 2)-way clustered ρ-VEX processor. It shows the number of ICC operations, *NOP* operations and the other regular operations with respect to the total number of operations. It can be observed from the chart that the number of inter-cluster communication operations range between 1.5% of *ucbqsort* to 12.8% of *compress* application.The *bcnt* application has 14.1% ICC operations but the total number of operations is small compared to the other benchmark applications, see Table 5.2. The percentile of *NOP* operations for the benchmark applications on average is 40% - 50%. This is one of drawback of VLIW architectures; the compiler does not always fit useful operations in all the issue slots. For applications with less ILP, most of the issue slots in the instruction bundles are filled with *NOP* operations.

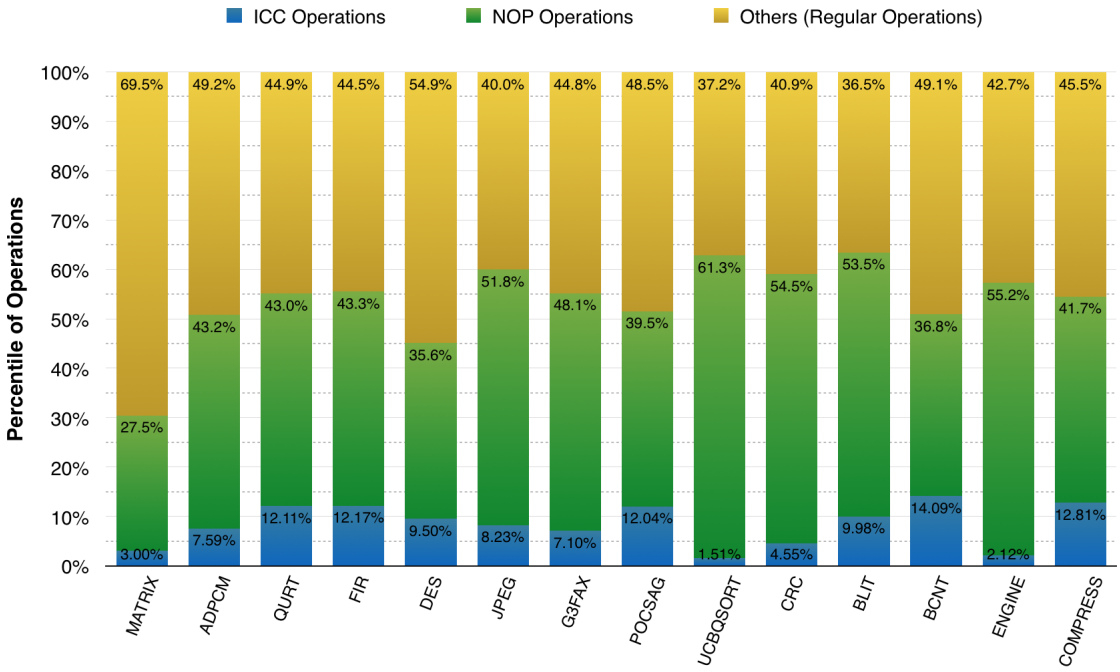


Figure 5.3: Percentile of Operations for *powerstone* benchmark applications in (2 + 2)-way ρ-VEX processor.

The percentile of operations for the benchmarks that are executed on an 8-issue 2 cluster ρ-VEX processor is depicted in Figure 5.4. The percentile of ICC operations of the benchmarks lies between 0.7% of *ucbqsort* to 12.5% of *bcnt* application. The percentile of *NOP* operations for all the applications in 8-issue clustered ρ-VEX processor increased compared to that of a 4-issue clustered ρ-VEX processor.

The *ucbqsort* application is a quick sort application that contains many conditional and branch operations which are not suitable for optimal instruction parallelism as the

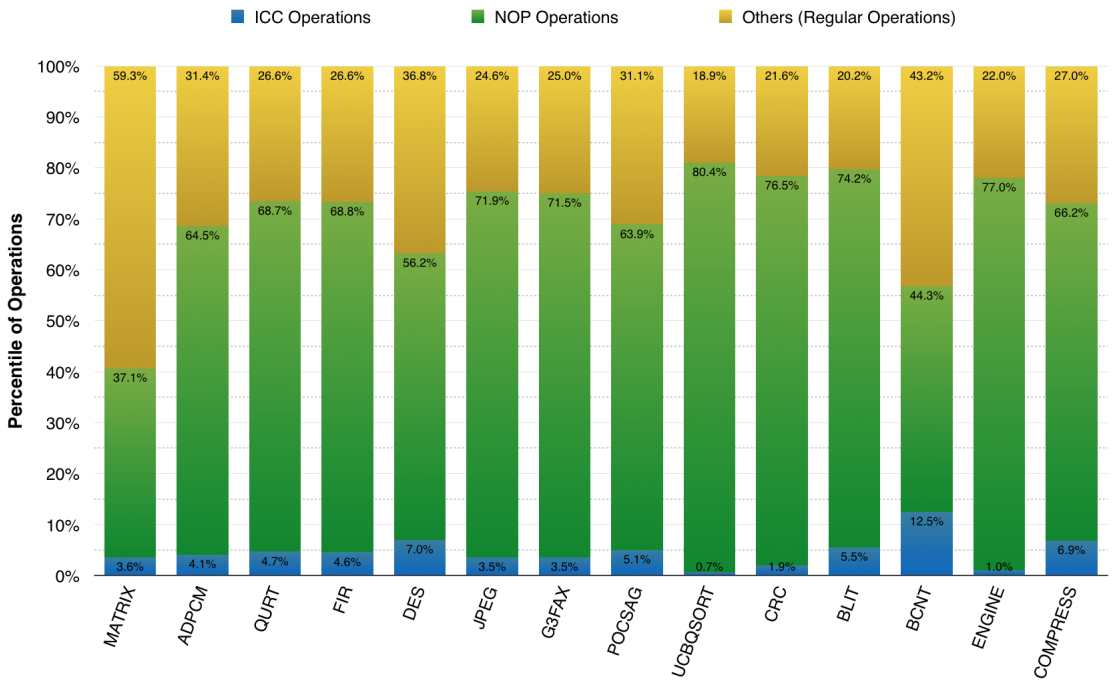


Figure 5.4: Percentile of Operations for *powerstone* benchmark applications in (4 + 4)-way ρ-VEX processor.

branch operations will have to be executed in a single branch unit of the clustered ρ-VEX processor. As a result, the total number of operations of the application in an 8-issue clustered ρ-VEX processor is 1.967×, almost double, total number of operations in 4-issue clustered ρ-VEX processor. The VEX compiler scheduled most of the operations in cluster 0 and most instruction bundles contain one to two useful operations. Consequently, 80.4% of the total number of operations are no-operations, *NOP*s.

5.2.4 Power Consumption and Area Overhead Analysis

The area usage and power consumption of the design-time and run-time reconfigurable clustered ρ-VEX processors are measured by taking the *Virtex-6 ML605 FPGA Evaluation board* as a target hardware platform.

Design-time Reconfigurable Clustered ρ-VEX Processors

The Table 5.3 depicts the slice logic utilization of the design-time reconfigurable (2 + 2)-way clustered and 4-issue ρ-VEX processors. The number of slice registers is increased in the (2 + 2)-way clustered ρ-VEX processor because the additional inter-cluster communication functional units in each issue slot are implemented using flip flops (Slice registers). On the other hand, the number of slice logic used as memory is significantly decreased by 74.5% in the clustered organization as the sizes of the register

files, their read and write ports and the complexity of the multiplexers in the register files are reduced. Similarly, the design-time reconfigurable (4 + 4)-way clustered ρ-VEX processor have a reduced area utilization than the 8-issue ρ-VEX processor as shown in Table 5.4. It uses 70.5% less lookup tables (LUTs) and occupies 57.2% lesser slices than the corresponding single cluster organization of the ρ-VEX processor. The number of slices used as memory in the (4 + 4)-way clustered processor is reduced by 74.7% compared to that of the 8-issue ρ-VEX processor.

Figure 5.5 shows the dynamic and quiescent (static) power consumption of the design-time reconfigurable 4- and 8-issue ρ-VEX processors, (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processors estimated by using the *Xilinx XPower analyzer tool*. The (2 + 2)-way clustered processor has similar dynamic power consumption with the 4-issue processor while the (4 + 4)-way clustered ρ-VEX processor uses less power than the 8-issue processor. The (4 + 4)-way clustered processor saves 0.09% dynamic power compared to the single cluster 8-issue processor. The static power overhead of all the processor architectures are in the same range.

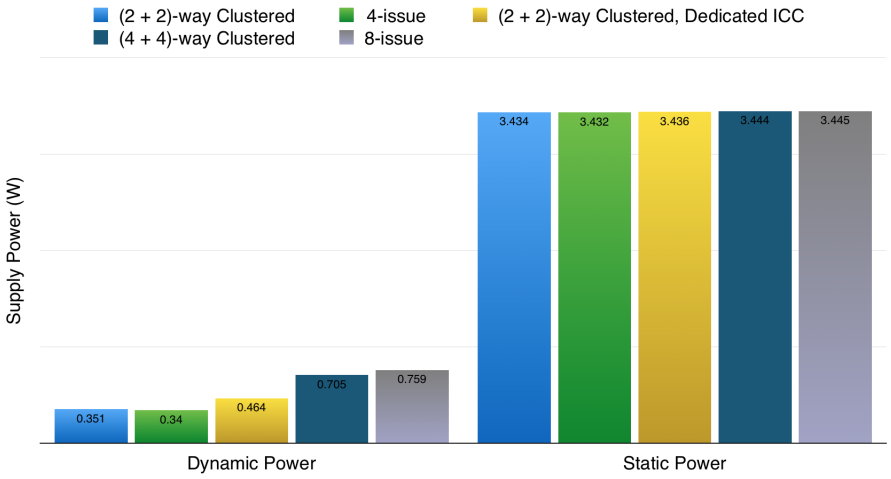


Figure 5.5: Dynamic and static power consumption of design-time reconfigurable 4- and 8-issue ρ-VEX processors, (2 + 2)-way clustered (with copy operation and dedicated issue slot ICC model), and (4 + 4)-way clustered ρ-VEX processors.

Run-time Reconfigurable Clustered ρ-VEX Processors

The area utilization of the baseline and clustered 2-4-8-issue run-time reconfigurable ρ-VEX processors are given in Table 5.5. The ICC functional units are implemented using registers, which increases the number slice registers in the clustered organization of the processor. On the other hand, a notable reduction in the number of occupied slices (by 61.30%) and LUTs (by 61.17%) is acheived.

Slice Logic Utilization (%)	Available	4-issue ρ-VEX processor		(2 + 2)-way Clustered		(2 + 2)-way Clustered, Dedicated ICC	
		Used	Utilization (%)	Used	Utilization (%)	Used	Utilization (%)
Number of Slice Registers	301440	1285	0.426	1513	0.502	2095	0.695
Number of Slice LUTs	150720	9889	6.536	9914	6.578	12827	8.510
Number used as Memory	58400	1408	2.411	360	0.616	1136	1.945
Number of occupied Slices	37680	3290	8.731	3587	9.519	4721	12.529
Number of RAMB36E1/ FIFO36E1s	416	96	23.077	96	23.077	114	27.404

Table 5.3: Area Utilization of 4-issue, (2 + 2)-way clustered ρ-VEX processor with default (copy operation) and dedicated issue slot ICC model on Virtex-6 FPGA.

Slice Logic Utilization	Available	8-issue ρ-VEX processor		(4 + 4)-way Clustered	
		Used	Utilization (%)	Used	Utilization (%)
Number of Slice Registers	301440	2198	0.729	2579	0.856
Number of Slice LUTs	150720	27950	18.544	19706	13.073
Number used as Memory	58400	5632	9.644	1424	2.438
Number of occupied Slices	37680	11488	30.488	6568	17.431
Number of RAMB36E1/ FIFO36E1s	416	128	30.047	128	30.047

Table 5.4: Area Utilization on Virtex-6 FPGA of the (4 + 4)-way clustered and 8-issue ρ-VEX processor.

Slice Logic Utilization	Available	2-4-8-issue		2-4-8-issue Clustered	
		Used	Utilization (%)	Used	Utilization (%)
Number of Slice Registers	301440	3277	1.087	3407	1.130
Number of Slice LUTs	150720	58860	39.052	22851	15.163
Number used as Memory	58400	0	0.000	0	0.000
Number of occupied Slices	37680	24187	64.190	9388	24.915
Number of RAMB36E1/ FIFO36E1s	416	256	61.538	64	15.385
Number of RAMB18E1/ FIFO18E1s	832	128	15.384	128	15.384

Table 5.5: Area Utilization of clustered and single cluster organizations of the run-time adaptable ρ-VEX processor on Virtex-6 FPGA.

The dynamic and static power consumptions of the single cluster and clustered organization of the run-time reconfigurable 2-4-8-issue ρ-VEX processors are depicted in Figure 5.6. It shows that the clustered processor consumes 41.6% less dynamic power than the baseline processor.

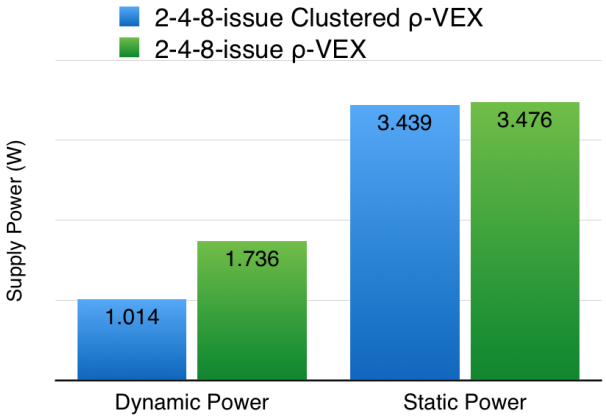


Figure 5.6: Dynamic and static power consumption of clustered and single cluster organizations of the run-time adaptable ρ-VEX processors.

5.2.5 Energy Delay Product (EDP)

The energy delay product is a performance evaluation metric that takes into account both the energy consumption and execution times of the processor. The EDP is calculated with the following equation:

$$EDP = Dynamic\_power\_consumed * Cycle\_time * Execution\_cycles^2 \quad (5.1)$$

The EDP of the design-time reconfigurable (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processors normalized to the EDP of the single cluster 4-issue and 8-issue ρ-VEX processors respectively are depicted in Figure 5.7. Applications such as *compress* and *pocsag* have high EDP in both clustered processors which implies that they consume more energy than the performance gain. The EDP of all benchmarks achieve more performance in the 8-issue of the clustered processor than the 4-issue. The largest reduction in the EDP in the (4 + 4)-way clustered ρ-VEX processor is for the *matrix* and *qurt* benchmarks which provide more performance improvement with less energy consumption.

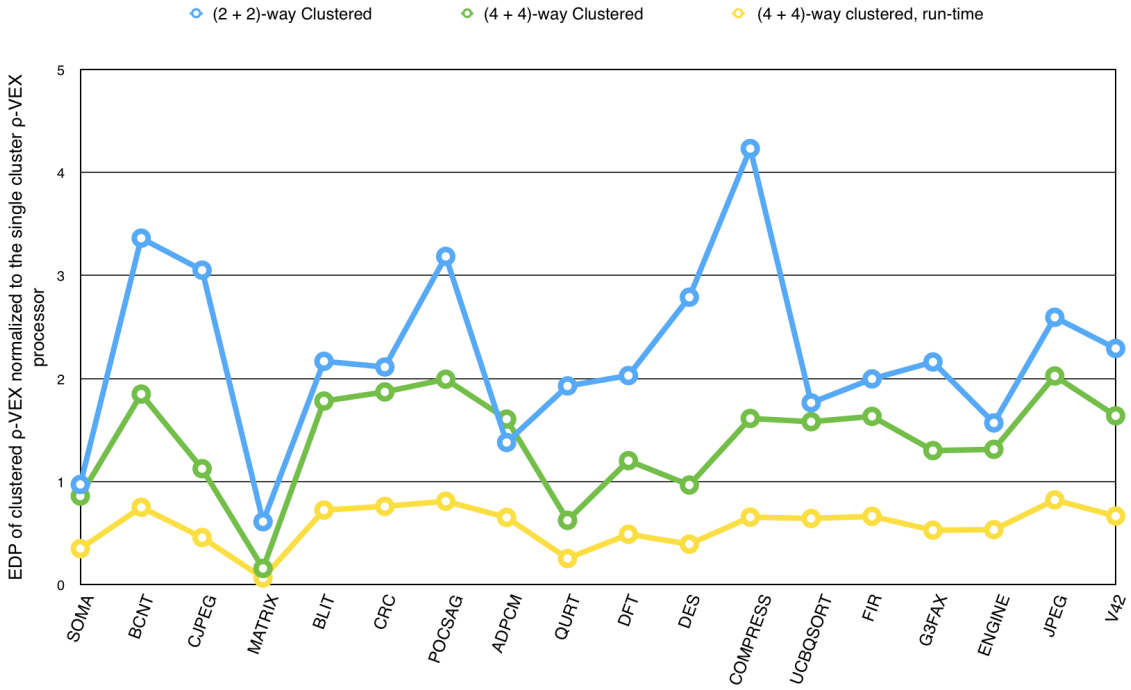


Figure 5.7: EDP of (2 + 2)-way and (4 + 4)-way clustered ρ-VEX processors normalized to their corresponding single cluster processors.

The run-time reconfigurable baseline and the clustered processors run at the same clock frequency. Thus, the EDP becomes the measure of the dynamic power and the cycle count. As a result, the EDP of the benchmarks are the smallest compared to that of the design-time reconfigurable processors. The run-time adaptable clustered processor acheives better performance by consuming less power.

5.3 Clustered ρ-VEX Processor with Dedicated Issue Slot ICC Model

5.3.1 Total Cycle Count and Speedup

The speedup of benchmark applications on the design-time reconfigurable (2 + 2)-way clustered ρ-VEX processor with dedicated issue slot ICC model is depicted in Figure 5.8. The cycle counts are similar to the (2 + 2)-way clustered ρ-VEX processor with copy operation ICC model because the assembly code for the benchmarks are generated using the same machine configuration model, Table 3.1, as discussed in Section 3.5.

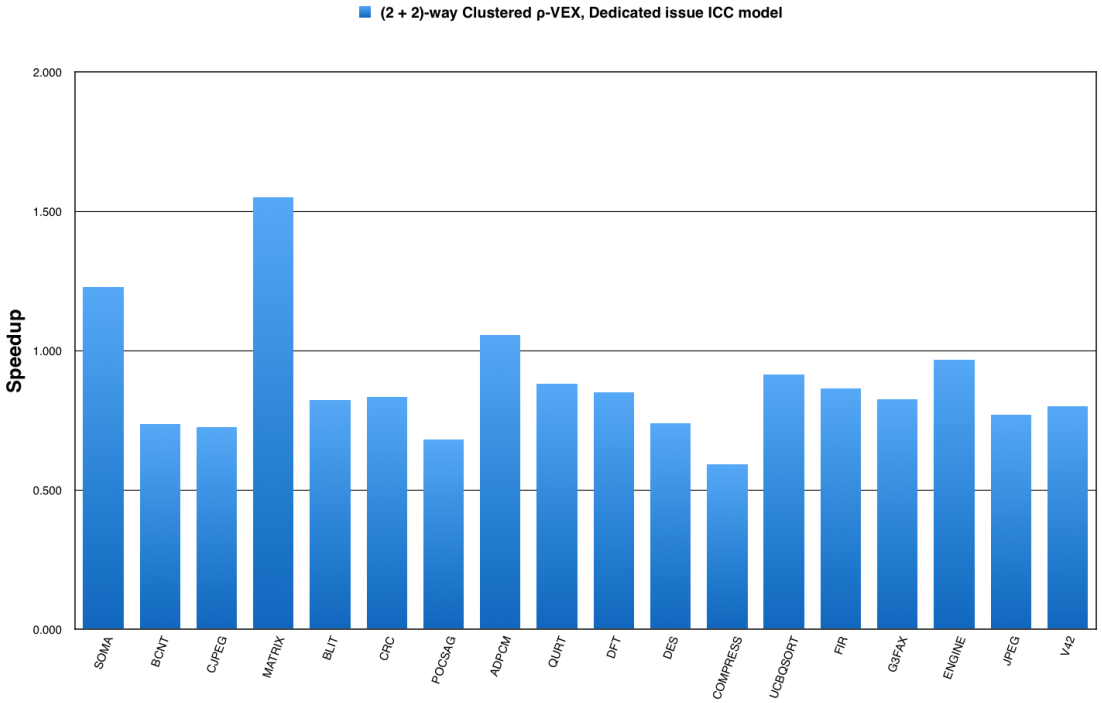


Figure 5.8: Speedup for *powerstone* benchmark applications in (2 + 2)-way clustered ρ-VEX processor with dedicated issue ICC model compared to the baseline 4-issue ρ-VEX processor.

The width of an instruction bundle is larger in dedicated issue slot ICC model than in copy operation ICC model; this in some cases fixed the bugs that were being generated by the VEX compiler as the inter-cluster communication operations moved into the dedicated ICC slots which leaves free slots for operations with long immediate operand. The assembler took advantage of freed issue slots and scheduled the operations on them.

5.3.2 Operations Count

The VEX compiler does not support clustered VLIW architectures with dedicated issue slot ICC model. The assembler is modified so that it schedules inter-cluster communication operations on a dedicated slots. The ICC model increases the instruction bundle size, more issue slots. These ICC operations are moved to those slots and freeing the slots where they would have been scheduled in copy operation ICC model as the inter-cluster operations can be scheduled on any issue slot. This creates a huge increase in the no-operation count. Evidently, Figure 5.9 shows that 64 - 81% of the operations executed on the clustered ρ-VEX processor for the benchmarks are NOP. The number of ICC operations on the architecture for the benchmark applications is between 0.3% and 12.1% of the total operations.

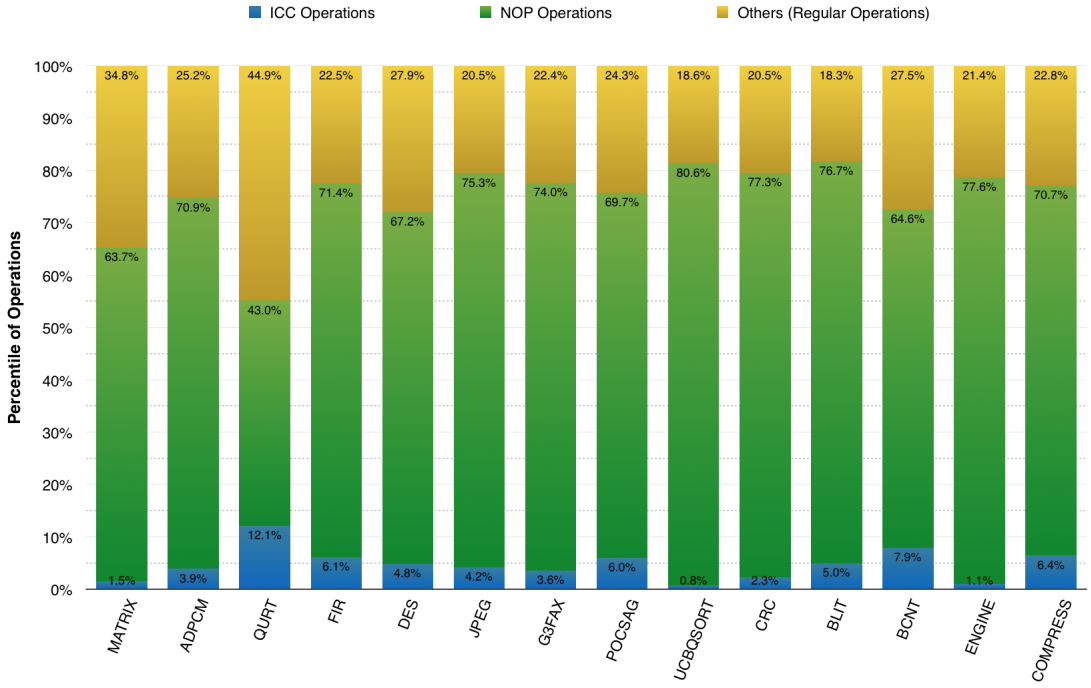


Figure 5.9: Percentile of Operations for *powerstone* benchmark applications in (2 + 2)-way clustered ρ-VEX processor with dedicated issue slot ICC model.

5.3.3 Power Consumption and Area Overhead Analysis

Table 5.3 shows the slice logic utilization of the design-time reconfigurable (2 + 2)-way ρ-VEX processors with dedicate issue slot ICC model. This architecture uses more slices as registers, BRAM and LUTs and less slices as memory than the 4-issue ρ-VEX processor and the (2 + 2)-way clustered with copy operation ICC model.

The static and dynamic power consumption of the (2 + 2)-way clustered ρ-VEX processor with dedicated issue slot ICC model is depicted in Figure 5.5. This proces-

sor consumes 36.7% more dynamic power than the single cluster and clustered ρ-VEX architectures.

5.4 Conclusion

In this chapter, experimental results on the design-time and run-time reconfigurable clustered ρ-VEX processors are presented. The processors are implemented on a Xilinx Virtex-6 FPGA evaluation kit. The design-time reconfigurable clustered processor runs achieved a drastic speedup of upto 1.55× in the clock frequency. The (4 + 4)-way clustered processor operates at 155 MHz while the 8-issue ρ-VEX processor run at 100 MHz. In addition, the clustered processor significantly reduces the area utilization by a maximum of 74.7% compared to the single cluster processor. But with respect to performance, it takes more cycles than single cluster ρ-VEX processor except for benchmarks that have high instruction level parallelism. A minimum speedup of 0.672× and 0.843× are obtained in the (2 + 2)-way and (4 + 4)-way clustered processors respectively.

The register file for the run-time reconfigurable clustered processor is not physically split into smaller subsets as otherwise the processor will lose its run-time reconfigurability feature. Thus an address generation unit is used to map the addresses to their dedicated regions in the BlockRAMs according to the mode of operation of the processor which are specified by the $issue\_ctrl$, $interclust$ and $num\_cluster$ control signals. Nevertheless, the addition of clustering to the processor resulted in a reduced area utilization, power consumption and EDP than the single cluster organization of the ρ-VEX processor. The number of slices used as LUTs are reduced by 61.3%. The clustered processor consumes 41.6% less dynamic power than the baseline processor. The run-time adaptable single cluster and clustered ρ-VEX processor runs at the same clock frequency of *110 MHz*.

The clustered architecture with copy operation ICC model and dedicated issue slot ICC model has similar performance because they basically use the same assembly code generated. The architecture with dedicated issue slot ICC model could achieve better performance by modifying the scheduling algorithm of the compiler. Nevertheless, the architecture with dedicated issue slot ICC model consumes more power and have a larger area utilization.

6 Conclusions

This chapter summarizes the overall work and assesses whether the goals of the project defined in Section 1.2 have been met and to what extent. The main contributions of the project are discussed in Section 6.2. Recommendations for future work on the area of multi-cluster VLIW organizations concludes the chapter and the thesis.

6.1 Summary

In Chapter 2, basic background concepts and related work on clustered VLIW architectures are presented. The VEX system that includes three components, the VEX instruction architecture, VEX compiler and a simulator system, are introduced. Pros and cons of different ICC models and variety of compiler scheduling techniques for clustered architectures are discussed. The broadcasting ICC model consume more power than the other ICC model while dedicated issue slot ICC model increases the area utilization of the processor. The VEX system provides complete support for clustered organization of VLIW architectures in its ISA and compiler toolchain while the other scheduling techniques do not provide that. Thus a clustered organization of the ρ-VEX reconfigurable processor is implemented using the trace scheduling algorithm of the VEX compiler and ISA supported copy operations ICC model. Clustering the register file reduces the register pressure and area overhead. Excessive clustering leads to an increase in the ICC and addition of read and write port (multi-port) to the data memory which requires the design of complicated redundant data memories with synchonization of data between them. Thus, an optimal clustering of two with 32 register in each cluster is chosen for the implementation of the clustered ρ-VEX processor.

Chapter 3 presents the detailed design and implementation of the design-time reconfigurable clustered ρ-VEX VLIW processor. The (2 + 2)-way and (4 + 4)-way clustered organization are the two ρ-VEX architectures discussed. Two inter-cluster communication models, the copy operation ICC model and dedicated issue slot ICC model, are implemented on the clustered ρ-VEX processors. The VEX ISA supports copy operation ICC model by providing explicit pair of operations (`send` and `recv`) for transferring local register values between the clusters. In addition to that, a model that dedicates pipelanes for inter-cluster communication is implemented. The structure and instruction bundle layout for this ICC model are presented. The modifications made to the compiler toolchain, the ρ-VEX processor core, the syllable of branch operations and the borrowing scheme for the clustered ρ-VEX processor are discussed. A bug found in the scheduling of the VEX compiler and the algorithm used to fix it are also presented.

Chapter 4 presents the introduction of clustering organization to the 2-4-8-issue

run-time adaptable ρ-VEX VLIW processor. ICC functional unit in each pipelane and inter-cluster bypass network are designed and implemented. The design supports the VEX clustered architecture, thus explicit inter-cluster copy operations are used as ICC model. `send` and `recv` inter-cluster operations are scheduled in the source and destination clusters by the assembler and it takes one cycle to transfer the register value between the clusters.

The single cluster organization of the 8-issue run-time reconfigurable ρ-VEX core and the dual 4-issue cores are clustered into clusters of two. A multiplexer unit is added to forward read and write addresses and data from the load/store unit of the clusters to their respective data memory based on the mode of operation of the processor. The detailed architectures of the two run-time reconfigurable clustered organizations, the two (2 + 2)-way and the (4 + 4)-way clustered processors, are presented. The compiler toolchain adaptations discussed in Section 3.2.1 has been used for this run-time reconfigurable processor. To the best of our knowledge, this makes our 2-4-8-issue dynamically reconfigurable processor is the only run-time reconfigurable clustered processor with a complete compiler toolchain.

In Chapter 5, experimental results on the design-time and run-time reconfigurable clustered ρ-VEX processors are presented. The processors are implemented on a Xilinx Virtex-6 FPGA evaluation kit. The design-time reconfigurable clustered processor runs achieved a drastic speedup of up to 1.55× in the clock frequency. The (4 + 4)-way clustered processor operates at 155 MHz while the 8-issue ρ-VEX processor run at 100 MHz. In addition, the clustered processor significantly reduces the area utilization by a maximum of 74.7% compared to the single cluster processor. But with respect to performance, it takes more cycles than single cluster ρ-VEX processor except for benchmarks that have high instruction level parallelism. A minimum speedup of 0.672× and 0.843× are obtained in the (2 + 2)-way and (4 + 4)-way clustered processors respectively.

The register file for the run-time reconfigurable clustered processor is not physically split into smaller subsets as otherwise the processor will lose its run-time reconfigurability feature. Thus an address generation unit is used to map the addresses to their dedicated regions in the BlockRAMs according to the mode of operation of the processor which are specified by the $issue\_ctrl$, $interclust$ and $num\_cluster$ control signals. Nevertheless, the addition of clustering to the processor resulted in a reduced area utilization, power consumption and EDP than the single cluster organization of the ρ-VEX processor. The number of slices used as LUTs are reduced by 61.3%. The clustered processor consumes 41.6% less dynamic power than the baseline processor. The run-time adaptable single cluster and clustered ρ-VEX processor runs at the same clock frequency of *110 MHz*. The clustered architecture with copy operation ICC model and dedicated issue slot ICC model has similar performance because they basically use the same assembly code generated. The architecture with dedicated issue slot ICC model could achieve better performance by modifying the scheduling algorithm of the compiler. Nevertheless, the architecture with dedicated issue slot ICC model consumes more power and have a larger area utilization.

6.2 Main Contributions

The goals of the thesis project that were defined in Section 1.2 have been fully met in the following ways:

- The compiler toolchain support for clustered organizations are developed in Section 3.2.
- The design and implementation of the design-time reconfigurable ρ-VEX processor is presented in Chapter 3.
- The introduction of clustered organization to the run-time reconfigurable ρ-VEX processor is implemented in Chapter 4.

The main contributions of the thesis are listed as follows:

- Designed and implemented clustered organization support to the design-time and run-time reconfigurable ρ-VEX processor which led to a reduced area, power utilization and increased maximum clock frequency of the processors.
- Implemented an inter-cluster communication functional unit and inter-cluster path for the clustered ρ-VEX processor. Those hardware components are responsible for transfering the register values between clusters.
- Modified the compiler toolchain to support multi-cluster VLIW organization. The *binutils* assembler, disassember, and linker are adapted to generate correct binaries for clustered organizations.
- Developed an algorithm to fix a bug generated by the VEX compiler for a (2 + 2)-way clustered VEX architectures. The compiler was not considering an issue slot to store part of the immediate operand for operations with long immediate operand. Thus, more operations than the available slots in an instruction bundle were being generated. The algorithm used to fix the bug is given in Section 3.7.

6.3 Future Work

Although the results of the introduction of clustering organization to both the design-time and run-time reconfigurable ρ-VEX processors have assured a significant reduction in area and power overhead and an increased clock frequency, the work can be further developed in many ways such as:

- VEX C compiler is a closed source compiler, thus the scheduling algorithm could not be altered to support new and efficient inter-cluster communication models other than the explicit copy operation ICC model. A mechanism to change the scheduling algorithm is required. An open source compiler such as Low Level Virtual Machine (LLVM) compiler [44] could be used for this purpose.

- Design of efficient cluster assignment and scheduling algorithm of compiler for clustered VLIW architectures with dedicated issue slot ICC model. Similar performance has been achieved with this ICC model by using the same code generated for clustered ρ-VEX processor with copy operation ICC model. Better performance can be obtained with this ICC model by utilizing efficient scheduling algorithm.
- Design of generic binaries that can run on different clustered organizations. Instead of compiling an application for different issueiwidth of clustered organization and loading those binaries to instruction and data memories, it is effective to have one binary that can be run on different clustered organizations. When a processor core is freed at run-time, the application can be run faster by only changing the mode of operation of the clustered ρ-VEX processor. Similar work for single cluster organization of the ρ-VEX processor can be found in [45].

Bibliography

[1] F. Anjam, M. Nadeem, and S. Wong, "A vliw softcore processor with dynamically adjustable issue-slots," in *Field-Programmable Technology (FPT), 2010 International Conference on*. IEEE, 2010, pp. 393–398.

[2] F. Anjam, "Run-time adaptable vliw processors."

[3] G. E. Moore, "The future of integrated electronics," *Fairchild Semiconductor internal publication*, vol. 2, 1964.

[4] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," 1965.

[5] J. A. Fisher, P. Faraboschi, and C. Young, *Embedded computing: a VLIW approach to architecture, compilers and tools*. Elsevier, 2005.

[6] V. N. Makarov, "Fighting register pressure in gcc," in *GCC Developers Summit*, 2004, p. 85.

[7] J. Scott, L. H. Lee, J. Arends, and B. Moyer, "Designing the low-power m. core tm architecture," in *Power Driven Microarchitecture Workshop*. Citeseer, 1998, pp. 145–150.

[8] "VEX C Compiler," http://www.hpl.hp.com/downloads/vex/, [Online; accessed 15-July-2014].

[9] S. Wong, T. Van As, and G. Brown, "ρ-vex: A reconfigurable and extensible softcore vliw processor," in *ICECE Technology, 2008. FPT 2008. International Conference on*. IEEE, 2008, pp. 369–372.

[10] J. A. Fisher, "Trace scheduling: A technique for global microcode compaction," *IEEE Transactions on Computers*, vol. 30, no. 7, pp. 478–490, 1981.

[11] P. Faraboschi, G. Desoli, and J. A. Fisher, *Clustered instruction-level parallel processors*. Hewlett Packard Laboratories, 1999.

[12] J. A. Fisher, *Global code generation for instruction-level parallelism: Trace scheduling-2*. Hewlett-Packard Laboratories, 1993.

[13] J. Sánchez and A. González, "Modulo scheduling for a fully-distributed clustered vliw architecture," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*. ACM, 2000, pp. 124–133.

[14] S. Ghosh, M. Martonosi, and S. Malik, "Cache miss equations: An analytical representation of cache misses," in *Proceedings of the 11th international conference on Supercomputing*. ACM, 1997, pp. 317–324.

[15] V. Porpodas and M. Cintra, "Caesar: unified cluster-assignment scheduling and communication reuse for clustered vliw processors," in *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*. IEEE Press, 2013, p. 9.

[16] K. Kailas, K. Ebcioglu, and A. Agrawala, "Cars: A new code generation framework for clustered ilp processors," in *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*. IEEE, 2001, pp. 133–143.

[17] V. Porpodas and M. Cintra, "Lucas: latency-adaptive unified cluster assignment and instruction scheduling," in *ACM SIGPLAN Notices*, vol. 48, no. 5. ACM, 2013, pp. 45–54.

[18] Porpodas, Vasileios and Cintra, Marcelo, "Uciff: Unified cluster assignment instruction scheduling and fast frequency selection for heterogeneous clustered vliw cores," in *Languages and Compilers for Parallel Computing*. Springer, 2013, pp. 127–142.

[19] S. Wong, A. Brandon, F. Anjam, R. Seedorf, R. Giorgi, Z. Yu, N. Puzovic, S. A. Mckee, M. Sjalander, L. Carro *et al.*, "Early results from eraembedded reconfigurable architectures," in *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*. IEEE, 2011, pp. 816–822.

[20] R. Seedorf, F. Anjam, A. Brandon, and S. Wong, "Design of a pipelined and parameterized vliw processor: ρ-vex v. 2.0," in *HiPEAC Workshop on Reconfigurable Computing (WRC)*, 2012.

[21] S. Wong and F. Anjam, "The delft reconfigurable vliw processor," *system*, vol. 1, p. 3, 2009.

[22] S. Wong, F. Anjam, and F. Nadeem, "Dynamically reconfigurable register file for a softcore vliw processor," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 969–972.

[23] C. E. LaForest and J. G. Steffan, "Efficient multi-ported memories for fpgas," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2010, pp. 41–50.

[24] A. Terechko, E. Le Thenaff, M. Garg, J. Van Eijndhoven, and H. Corporaal, "Inter-cluster communication models for clustered vliw processors," in *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on*. IEEE, 2003, pp. 354–364.

[25] A. Terechko and H. Corporaal, "Inter-cluster communication in vliw architectures," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 4, no. 2, p. 11, 2007.

[26] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. Homewood, *Lx: a technology platform for customizable VLIW embedded processing.* ACM, 2000, vol. 28, no. 2.

[27] R. Canal, J.-M. Parcerisa, and A. González, "Dynamic cluster assignment mechanisms," in *High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium on.* IEEE, 2000, pp. 133–142.

[28] B. V. Iyer, *Length adaptive processors: a solution for the energy/performance dilemma in embedded systems.* ProQuest, 2009.

[29] Texas, Instruments (2000), "Tms320c64x technical overview," *Texas Instruments, Feb*, 2000.

[30] Texas, Instruments (2006), "Tms320c64x/c64x+ dsp cpu and instruction set reference guide," *SPRU732C edn*, August, 2006.

[31] N. Seshan, "High velociti processing [texas instruments vliw dsp architecture]," *Signal Processing Magazine, IEEE*, vol. 15, no. 2, pp. 86–101, 1998.

[32] J. Fridman and Z. Greenfield, "The tigersharc dsp architecture," *IEEE micro*, vol. 20, no. 1, pp. 66–76, 2000.

[33] G. G. Pechanek and S. Vassiliadis, "The manarray tm embedded processor architecture," in *Euromicro Conference, 2000. Proceedings of the 26th*, vol. 1. IEEE, 2000, pp. 348–355.

[34] M. Tremblay, J. Chan, S. Chaudhry, A. W. Conigliaro, and S. S. Tse, "The majc architecture: A synthesis of parallelism and scalability," *IEEE Micro*, vol. 20, no. 6, pp. 12–25, 2000.

[35] O. Colavin and D. Rizzo, "A scalable wide-issue clustered vliw with a reconfigurable interconnect," in *Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems.* ACM, 2003, pp. 148–158.

[36] Y. Sato, K.-i. Suzuki, and T. Nakamura, "Partitioned register file designs for clustered architectures," *INFORMATION-YAMAGUCHI-*, vol. 9, no. 1, p. 119, 2006.

[37] R. E. Kessler, E. J. McLellan, and D. A. Webb, "The alpha 21264 microprocessor architecture," in *Computer Design: VLSI in Computers and Processors, 1998. ICCD'98. Proceedings. International Conference on.* IEEE, 1998, pp. 90–95.

[38] A. Terechko, M. Garg, and H. Corporaal, "Evaluation of speed and area of clustered vliw processors," in *VLSI Design, 2005. 18th International Conference on.* IEEE, 2005, pp. 557–563.

[39] "GNU Binutils," http://www.gnu.org/software/binutils/, [Online; accessed 15-July-2014].

[40] "Xilinx Virtex-6 FPGA ML605 Evaluation Kit," http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm, [Online; accessed 15-July-2014].

[41] Xilinx, ISE, "Ise design suite," 2012.

[42] "xPower Analysis Tool," http://www.xilinx.com/products/design_tools/logic_design/verification/xpower.htm, [Online; accessed 15-July-2014].

[43] "ModelSim," http://www.mentor.com/products/fpga/model/, [Online; accessed 15-July-2014].

[44] C. Lattner and V. Adve, "The llvm compiler framework and infrastructure tutorial," in *Languages and Compilers for High Performance Computing*. Springer, 2005, pp. 15–16.

[45] A. Brandon and S. Wong, "Support for dynamic issue width in vliw processors using generic binaries," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 827–832.

Inter-Cluster Communication Functional Unit

A

The following VHDL code is a the hardware program for inter-cluster communication functional unit. The ICC FU in the source cluster copies and write the content of the register to the inter-cluster path while the destination cluster writes an identifier or a header into its write port of the inter-cluster path and receives the data.

```
1
-- r-VEX processor
3 -- Copyright (C) 2008-2014 by TU Delft.
-- All Rights Reserved.
5
-- THIS IS A LEGAL DOCUMENT, BY USING r-VEX,
7 -- YOU ARE AGREEING TO THESE TERMS AND CONDITIONS.

9 -- No portion of this work may be used by any commercial entity, or for any
-- commercial purpose, without the prior, written permission of TU Delft.
11 -- Nonprofit and noncommercial use is permitted as described below.

13 -- 1. r-VEX is provided AS IS, with no warranty of any kind, express
-- or implied. The user of the code accepts full responsibility for the
15 -- application of the code and the use of any results.

17 -- 2. Nonprofit and noncommercial use is encouraged. r-VEX may be
-- downloaded, compiled, synthesized, copied, and modified solely for
    nonprofit,
19 -- educational, noncommercial research, and noncommercial scholarship
-- purposes provided that this notice in its entirety accompanies all
    copies.
21 -- Copies of the modified software can be delivered to persons who use it
-- solely for nonprofit, educational, noncommercial research, and
23 -- noncommercial scholarship purposes provided that this notice in its
-- entirety accompanies all copies.
25
-- 3. ALL COMMERCIAL USE, AND ALL USE BY FOR PROFIT ENTITIES, IS EXPRESSLY
27 -- PROHIBITED WITHOUT A LICENSE FROM TU Delft (J.S.S.M.Wong@tudelft.nl).

29 -- 4. No nonprofit user may place any restrictions on the use of this
    software,
-- including as modified by the user, by any other authorized user.
31
-- 5. Noncommercial and nonprofit users may distribute copies of r-VEX
33 -- in compiled or binary form as set forth in Section 2, provided that
-- either: (A) it is accompanied by the corresponding machine-readable
    source
```

```
35 -- code, or (B) it is accompanied by a written offer, with no time limit,
       to
   -- give anyone a machine-readable copy of the corresponding source code in
37 -- return for reimbursement of the cost of distribution. This written offer
   -- must permit verbatim duplication by anyone, or (C) it is distributed by
39 -- someone who received only the executable form, and is accompanied by a
   -- copy of the written offer of source code.
41
   -- 6. r-VEX was developed by Stephan Wong, Thijs van As, Fakhar Anjam, Roel
        Seedorf,
43 -- Anthony Brandon. r-VEX is currently maintained by TU Delft (J.S.S.M.
       Wong@tudelft.nl).

45 -- 7. Clustered organization was introduced to the r-VEX design by Muez
       Berhane Reda.

47 -- Copyright (C) 2008-2014 by TU Delft.

49 library IEEE;
   use IEEE.std_logic_1164.all;
51 use IEEE.numeric_std.ALL;

53 library work;
   use work.opcode_pkg.all;
55 use work.rvex_pkg.all;

57 entity send_recv is
     port (
59     clk            : in  std_logic;                          -- system clock
       clk_enable     : in  std_logic;                          -- clock enable
61     reset          : in  std_logic;                          -- system reset
       INTERCLUST     : in  boolean := false;
63     opcode         : in  std_logic_vector(7 downto 0);       -- opcode
       dst_reg        : in  std_logic_vector(GR_ADDR_WIDTH -1 downto 0);  --
       destination reg
65     src_reg        : in  std_logic_vector(31 downto 0);      -- operand 1
       clusters       : in  std_logic_vector(31 downto 0);      -- operand 2
67     write_enable_icc : out std_logic;
       write_data       : out std_logic_vector(43 downto 0);
69     read_reg_no    : out  std_logic_vector(11 downto 0);
       read_data      : in std_logic_vector(31 downto 0)
71     );
   end entity send_recv;
73
   architecture behv of send_recv is
75
       signal write_data_s       :  std_logic_vector(43 downto 0) := (others
       => '0');
77     signal write_enable_icc_s :  std_logic := '0';
       signal read_reg_no_s      :  std_logic_vector(11 downto 0) := (others
       => '0');
79
       begin
```

```
81
    l0 : process (dst_reg, src_reg, clk, INTERCLUST)
83      constant con : std_logic_vector(2 downto 0) := "000";
          variable reset : std_logic := '0';
85  begin

87    if(rising_edge(clk)) then

89    if INTERCLUST and std_match(opcode, INTR_SEND) then
          write_enable_icc_s    <= '1';
91        write_data_s          <= (con(6 - GR_ADDR_WIDTH - 1 downto 0) &
    dst_reg & clusters(5 downto 0) & src_reg);

93    elsif INTERCLUST and  std_match(opcode, INTR_RECV) then
          read_reg_no_s       <= (con(6 - GR_ADDR_WIDTH - 1 downto 0) &
    dst_reg & clusters(5 downto 0));
95
            else
97        write_enable_icc_s  <= '0';
          write_data_s        <= (others => '0');
99        read_reg_no_s       <= (others => '0');
      end if;
101   end if;
    end process l0;
103
     l1 : write_enable_icc <= write_enable_icc_s;
105  l2 : write_data     <= write_data_s;
     l4 : read_reg_no      <= read_reg_no_s;
107
end behv;
```

B Inter-Cluster Path

The inter-cluster path is a bypass network that is responsible for transferring the register data between clusters. It is connected to each issue lane with one read and one write port. The following code is the hardware description of the inter-cluster path for the 2-4-8-issue run -time reconfigurable clustered ρ-VEX processor.

```
-- r-VEX processor
-- Copyright (C) 2008-2014 by TU Delft.
-- All Rights Reserved.

-- THIS IS A LEGAL DOCUMENT, BY USING r-VEX,
-- YOU ARE AGREEING TO THESE TERMS AND CONDITIONS.

-- No portion of this work may be used by any commercial entity, or for any
-- commercial purpose, without the prior, written permission of TU Delft.
-- Nonprofit and noncommercial use is permitted as described below.

-- 1. r-VEX is provided AS IS, with no warranty of any kind, express
-- or implied. The user of the code accepts full responsibility for the
-- application of the code and the use of any results.

-- 2. Nonprofit and noncommercial use is encouraged. r-VEX may be
-- downloaded, compiled, synthesized, copied, and modified solely for
    nonprofit,
-- educational, noncommercial research, and noncommercial scholarship
-- purposes provided that this notice in its entirety accompanies all
    copies.
-- Copies of the modified software can be delivered to persons who use it
-- solely for nonprofit, educational, noncommercial research, and
-- noncommercial scholarship purposes provided that this notice in its
-- entirety accompanies all copies.

-- 3. ALL COMMERCIAL USE, AND ALL USE BY FOR PROFIT ENTITIES, IS EXPRESSLY
-- PROHIBITED WITHOUT A LICENSE FROM TU Delft (J.S.S.M.Wong@tudelft.nl).

-- 4. No nonprofit user may place any restrictions on the use of this
    software,
-- including as modified by the user, by any other authorized user.

-- 5. Noncommercial and nonprofit users may distribute copies of r-VEX
-- in compiled or binary form as set forth in Section 2, provided that
-- either: (A) it is accompanied by the corresponding machine-readable
    source
-- code, or (B) it is accompanied by a written offer, with no time limit,
    to
```

```
35 -- give anyone a machine-readable copy of the corresponding source code in
   -- return for reimbursement of the cost of distribution. This written offer
37 -- must permit verbatim duplication by anyone, or (C) it is distributed by
   -- someone who received only the executable form, and is accompanied by a
39 -- copy of the written offer of source code.

41 -- 6. r-VEX was developed by Stephan Wong, Thijs van As, Fakhar Anjam, Roel
        Seedorf,
   -- Anthony Brandon. r-VEX is currently maintained by TU Delft (J.S.S.M.
       Wong@tudelft.nl).
43
   -- 7. Clustered organization was introduced to the r-VEX design by Muez
       Berhane Reda.
45
   -- Copyright (C) 2008-2014 by TU Delft.
47
   library IEEE;
49 use IEEE.STD_LOGIC_1164.all;
   use IEEE.NUMERIC_STD.all;
51
   library work;
53 use work.type_pkg.all;
   use work.util_pkg.all;
55
   entity interclust_comm is
57   generic (
       ISSUE_WIDTH : natural := 4);
59   port (
       clk           : in  std_logic;
61     clk_enable    : in  std_logic;
       issue_ctrl    : in std_logic_vector(1 downto 0);
63     INTERCLUST    : in boolean;
       write_enable  : in  std_logic_vector(ISSUE_WIDTH - 1 downto 0);
65     write_data    : in  data_vector_icc(ISSUE_WIDTH - 1 downto 0);
       read_reg_no   : in  address_vector_reg_no_icc(ISSUE_WIDTH - 1 downto 0)
       ;
67     read_data     : out data_vector(ISSUE_WIDTH - 1 downto 0));
   end interclust_comm;
69
   architecture Behavioral of interclust_comm is
71
     signal write_enable_s : std_logic_vector(ISSUE_WIDTH - 1 downto 0) := (
       others => '0');
73   signal read_data_s : data_vector(ISSUE_WIDTH - 1 downto 0) := ((others =>
       (others => '0')));
     signal read_data_next : data_vector(ISSUE_WIDTH - 1 downto 0) := ((others
       => (others => '0')));
75   constant FORWARDING  : boolean := true;

77 begin

79   write_enable_process : process (write_enable, INTERCLUST)
     begin  -- process write_enable_process
```

```
81      for index in 0 to ISSUE_WIDTH - 1 loop
          if INTERCLUST and write_enable(index) = '0' then
83          write_enable_s(index) <= '0';
          else
85          write_enable_s(index) <= write_enable(index);
          end if;
87      end loop;  -- index
    end process write_enable_process;
89
      forwarding_process : process (read_reg_no, write_data, write_enable_s,
      issue_ctrl, INTERCLUST, read_data_s)
91
      begin  -- process forwarding_process
93      if INTERCLUST and issue_ctrl = "00" then   -- one 8-issue clustered
      core
          for read_index in 0 to ISSUE_WIDTH - 1 loop
95          read_data(read_index) <= (others => '0');
            for write_index in 0 to ISSUE_WIDTH - 1 loop
97              if std_match(read_reg_no(read_index), write_data(write_index)
      (43 downto 32)) and
                unsigned(read_reg_no(read_index)) /= 0 and
99              write_enable_s(write_index) = '1' then
                  read_data(read_index) <= write_data(write_index)(31 downto
      0);
101                 exit;
                end if;
103         end loop;  -- write_index
          end loop;  -- read_index
105
        elsif INTERCLUST and issue_ctrl = "01" then -- two 4-issue clustered
      cores
107       for i in 0 to 1 loop
            for read_index in 0 to (ISSUE_WIDTH / 2) - 1 loop
109           read_data(read_index + i*(ISSUE_WIDTH / 2)) <= (others => '0');
              for write_index in 0 to (ISSUE_WIDTH / 2) - 1 loop
111             if std_match(read_reg_no(read_index + i*(ISSUE_WIDTH / 2)),
      write_data(write_index + i*(ISSUE_WIDTH / 2))(43 downto 32)) and
                  unsigned( read_reg_no(read_index + i*(ISSUE_WIDTH / 2)))
      /= 0 and
113                 write_enable_s(write_index + i*(ISSUE_WIDTH / 2)) = '1'
      then
                  read_data(read_index + i*(ISSUE_WIDTH / 2)) <= write_data(
      write_index + i*(ISSUE_WIDTH / 2))(31 downto 0);
115             end if;
            end loop;  -- write_index
117       end loop;  -- read_index
        end loop;
119   else
        read_data <= (others => (others => '0'));
121   end if;
    end process forwarding_process;
123 end Behavioral;
```