# Targeting Code Diversity with Run-time Adjustable Issue-slots in a Chip Multiprocessor

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

***Abstract*—This paper presents an adaptable softcore chip multiprocessor (CMP). The processor instruction set architecture (ISA) is based on the VEX ISA. The issue-width of the processor can be adjusted at run-time (before an application starts). The processor has eight 2-issue cores that can run independently from each other. If not in use, each core can be taken to a lower power mode by gating off its source clock. Multiple 2-issue cores can be combined at run-time to form a variety of configurations of very long instruction word (VLIW) processors. The CMP is implemented in the Xilinx Virtex-6 *XC6VLX240T* FPGA. It has a single ISA and requires no specialized compiler support. The CMP can target a variety of applications having instruction and/or data level parallelism. We found that applications/kernels with larger instruction level parallelism (ILP) performs better when run on a larger issue-width core, while applications with larger data level parallelism (DLP) performs better when run on multiple 2-issue cores with the data distributed among the cores.**

## I. Introduction

To exploit the instruction level parallelism (ILP)[1], very long instruction word (VLIW) processors are utilized to increase the performance beyond single issue-width processors [1]. While single issue-width processors can only take advantage of temporal parallelism (by utilizing pipelining), VLIW architectures can additionally take advantage of the spatial parallelism by utilizing multiple functional units (FUs) to execute several operations simultaneously. In addition, VLIW processors are simpler in design when compared to their more complex (out-of-order) reduced instruction set computer (RISC) counterparts.

When a chip multiprocessor (CMP) is implemented in application-specific integrated circuit (ASIC), the configuration and issue-width of the available cores are fixed at design time. Therefore, the issue-width of the available cores and the combination of these cores cannot be adjusted to suit a different set of applications after fabrication. A softcore VLIW processor can be implemented in an FPGA, and its organization can be changed easily by loading a new bitstream. Design-time reconfigurability requires full configuration of the whole FPGA if a certain parameter of the FPGA-implemented processor is to be changed. In literature, all of the available softcore VLIW processors [2][3][4][5][6][7] only provide some form of design-time and not run-time reconfigurability. In order to change the issue-width or any other parameter of the processor, a new full bitstream is to be downloaded to configure the whole FPGA and all other circuits in the FPGA have to be stopped.

We implemented a softcore CMP that is run-time reconfigurable/adjustable. In [8][9], the implementation of an open-source parameterized and extensible softcore VLIW processor ($\rho$-VEX) is presented. In this paper, we utilized this processor as a base processor for our CMP design. The advantage of utilizing a softcore VLIW processor is that its parameters can be changed. After the processor is implemented in an FPGA, it can adjust its issue-width while other circuits in the FPGA are running. A set of external signals control the configuration/adjustment of the issue-slots. The CMP has eight 2-issue cores each of which can be run independently and multiple (up to four) cores can be combined to form a larger issue-width core or split vice versa. Hence, before an application starts executing, the machine's organization can be changed/adjusted to suit the application. Applications with more fine-grain (instruction level) parallelism can be run on the larger issue-width core for better performance, and applications with more coarse-grain (data level) parallelism can be run on the multiple 2-issue cores with the data divided among the cores for faster execution.

The remainder of the paper is organized as follows. Related work is discussed in Section II. Section III describes the VEX system. The $\rho$-VEX VLIW processor is briefly introduced in Section IV. Section V presents the design of our run-time adjustable/reconfigurable CMP. Results are discussed in Section VI. Finally, conclusions are presented in Section VII.

## II. Related Work

In literature, the available softcore VLIW processors [2][3][4][6][7][5] either do not have a complete toolchain, or they are based on some proprietary softcore processors (Xilinx MicroBlaze or Altera NIOS-II), which are not open-source. We utilized the $\rho$-VEX VLIW processor [8], which is open-source and has a complete toolchain (compiler, simulator, and assembler). Softcore multiprocessor systems as found in literature are mostly based on either the Xilinx MicroBlaze [10][11][12][13] or the Altera NIOS-II softcore processors [14][15]. The main drawback of all these designs is that they are utilizing proprietary softcores which are not open-source. In addition, the NIOS-II and MicroBlaze are single issue-width

[1]In this paper, "instruction" in ILP refers to RISC instructions. These are similar to the operations in the VLIW. When referring to VLIW instruction, this will be explicitly mentioned.

processor cores and cannot exploit ILP as can be exploited by a VLIW processor core. Since these multiprocessor systems do not have multi-issue cores, they do not have the ability to change their issue-width composition for an incoming application at run-time.

*TRIPS* [16] is a reconfigurable architecture that enables the available out-of-order processing cores and the on-chip memory system to be configured and combined in different modes for instruction, data, or thread-level parallelism. TRIPS implements a custom ISA and microarchitecture, and relies heavily on compiler support for scheduling instructions to extract ILP. *Smart memories* [17] is a reconfigurable architecture capable of merging in-order RISC cores to form a VLIW machine. The two configurations are not ISA-compatible, and the VLIW configuration requires specialized compiler support. *Core fusion* [18] provides mechanisms to combine small out-of-order cores to make larger issue-width cores at run-time and does not require specialized compiler support. Core fusion is implemented in a simulator. On the other hand, we provide a design in which small 2-issue in-order cores can morph at run-time to make a larger issue-width core and hence can handle instruction and data level parallelism. Our architecture sticks to a single ISA, and requires no specialized compiler support. We implemented our design in an FPGA, where we have the additional advantage of tuning our 2-issue base core as well as the whole CMP compared to an ASIC design.

## III. The VEX System: ISA and Toolchain

The VEX stands for VLIW Example [19]. The VEX is developed by Hewlett-Packard (HP) and STMicroelectronics. The VEX instruction set architecture (ISA) is a 32-bit clustered VLIW ISA that is scalable and customizable to individual application domains. The VEX ISA is loosely modeled on the ISA of the HP/ST Lx (ST200) family of VLIW embedded cores [1]. Based on *trace scheduling*, the VEX C compiler is a parameterized ISO/C89 compiler. A flexible programmable machine model determines the target architecture, which is provided as input to the compiler. A VEX software toolchain including the VEX C compiler and the VEX simulator is made freely available by the Hewlett-Packard Laboratories [20].

## IV. The $\rho$-VEX VLIW Processor

The $\rho$-VEX is a configurable (design-time) open-source VLIW softcore processor [8]. The ISA is based on the VEX ISA [19]. Different parameters of the $\rho$-VEX processor, such as the number and type of functional units (FUs), number of multiported registers (size of register file), number and type of accessible FUs per syllable, width of memory buses, and different latencies can be changed at design time. Figure 1 depicts the organization of a 32-bit, 2-issue $\rho$-VEX VLIW processor implemented in an FPGA. The $\rho$-VEX processor consists of *fetch*, *decode*, *execute,* and *writeback* units. The fetch unit fetches a VLIW instruction from the attached instruction memory, and splits it into syllables that are passed on to the decode unit. Here VLIW instructions are decoded and register contents used as operands are fetched from the register

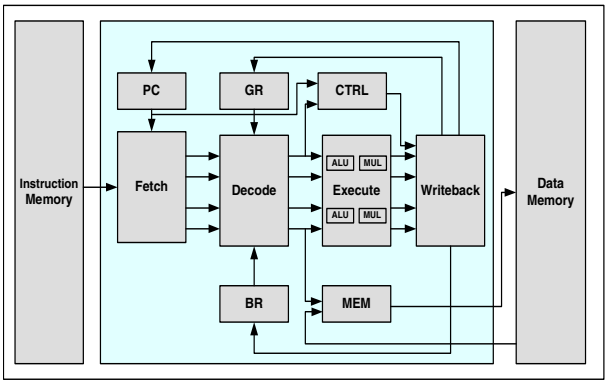


Figure 1. 2-issue $\rho$-VEX VLIW processor

file. The actual operations take place in either the execute unit, or in one of the parallel *branch or control* (CTRL) or *load/store or memory* (MEM) units. *Arithmetic logic unit* (ALU) and *multiplier* (MUL) operations are performed in the execute unit. All jump and branch operations are handled by the CTRL unit, and all data memory load and store operations are handled by the MEM unit. All write activities are performed in the writeback unit to ensure that all targets are written back at the same time. The write targets could be the *general register* (GR) file and/or the *branch registe*r (BR) file. The instruction and data memories for the processor are implemented with BRAMs. The $\rho$-VEX processor supports reconfigurable operations, as the VEX compiler supports the use of custom VLIW operations via pragmas inside the application code. The implementation of custom operations for our CMP design is out of scope of this paper.

## V. Run-time Adjustable CMP Design and Implementation

In this section, we present the design and implementation details of our run-time adjustable CMP. We utilize the 2-issue $\rho$-VEX VLIW processor as the base core for our design. Our CMP is composed of eight (8) such cores. These cores can be combined or split at run-time to form larger or smaller issue-width cores respectively. Figure 2 depicts the general view of our run-time adjustable CMP system.

At maximum, four 2-issue cores can be combined together to form an 8-issue core. Following are the possible configu-

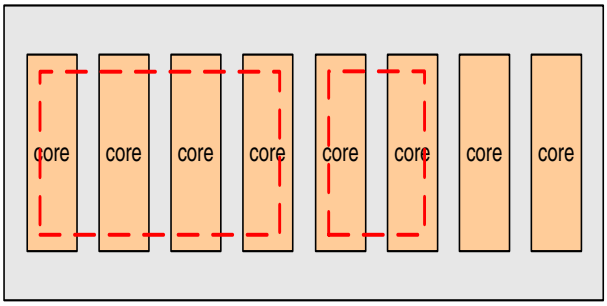


Figure 2. General view of the run-time adjustable CMP

rations with these eight cores: (1) two 8-issue cores, (2) one 8-issue and two 4-issue cores, (3) one 8-issue, one 4-issue and two 2-issue cores, (4) four 4-issue cores, (5) three 4-issue and two 2-issue cores, (6) two 4-issue and four 2-issue cores, (7) one 4-issue and six 2-issue cores, and (8) eight 2-issue cores. Each 2-issue core has two ALUs and two multiplier units (MULs). There is also a load/store or memory unit (MEM) as well as a branch or control unit (CTRL). A 4-issue core has double the resources of a 2-issue core except that only one of the CTRL units is utilized when two 2-issue cores are combined. An 8-issue core has double the resources of a 4-issue core except that only one of the CTRL units is utilized when four 2-issue cores are combined.

### A. Mechanism of Issue-width Adjustment

Each of the eight 2-issue cores has an input signal called *run*. When the *run* signal for a core is at logic 'one', the core starts fetching its VLIW instructions. When this signal is at logic 'zero', the program counter (PC) for that core is stopped. The *run* signal for a core is also utilized to clock gate the source clock for the core. If any of the 2-issue cores is not executing any application, it can be taken to a lower power mode by gating off its source clock, and hence, the dynamic power of that core is reduced resulting in a reduced total power consumption of the system. When a core finishes its execution, it raises its *done* signal. The *done* signal is utilized in order to schedule new code on a core. Another group of external signals called the *issue_ctrl* controls the organization and issue-width of the cores. For example, when the *issue_ctrl* signal is at logic 'zero', the system behaves as eight 2-issue cores. Different cores can be combined or split by utilizing this signal. The *issue_ctrl* signal controls the PCs for cores that are combined to form a larger issue-width core and drives them in lock-step. It additionally controls the organization of the register files. In order to group or ungroup certain cores to change the issue-width, the selected cores are first stopped utilizing their *run* signals. In the next cycle, the *issue_ctrl* signal is changed to adjust the issue-width of the resulting core/cores. In the next cycle, the *run* signals are asserted and the cores start fetching their VLIW instructions. The signals *issue_ctrl*, *run*, and *done* can be controlled and monitored by higher level controller or scheduler, which are out of scope of this paper and therefore, not discussed here.

We divided our CMP system into two sections namely *frontend* and *backend*. The frontend consists of the modules which need reconfiguration/adjustment in order to combine or split the issue-slots. It includes the fetch and decode units, and the multiported general-purpose register (GR) file and the branch register (BR) file. The backend consists of the writeback and execution units which do not require reconfiguration/adjustment when multiple 2-issue cores are combined to form larger issue-width cores or split the larger issue-width cores back to the base 2-issue cores.

### B. Frontend

The frontend of the CMP requires reconfiguration/adjustment for changing the issue-width of the cores. The signals *issue_ctrl* and *run* are utilized to combine or split the issue-slots. Here we discuss all modules of the frontend.

*1) Fetch Unit:* A 2-issue fetch unit simply splits the incoming long instruction into two syllables (instructions (32-bit) for individual execution units), and then passes them to the decode unit. Therefore, multiple 2-issue fetch units can be stacked together to form a combined fetch unit to behave like a larger issue (up to 8-issue) fetch unit. Each 2-issue fetch unit has a program counter (PC). The only sub-unit of a fetch unit that needs to be altered/reconfigured is the PC. If multiple fetch units are combined to form a larger issue-width core, only one of the PC is running and other PCs in that specific core are stopped. The signal *issue_ctrl* is utilized for this purpose.

*2) General-Purpose Register File:* The VEX ISA specifies a 32-bit 64-element multiported general-purpose register (GR) file for a multi-issue VLIW core. For 2-issue, 4-issue, and 8-issue cores, we require register files with 2-write-4-read (2W4R) ports, 4-write-8-read (4W8R) ports, and 8-write-16-read (8W16R) ports, respectively. We implemented our register file with BlockRAMs (BRAMs) based on the design presented in [21]. We designed our register file such that a single register file can handle an 8-issue core or two 4-issue cores or one 4-issue and two 2-issue cores or four 2-issue cores at the same time. The register file is depicted in Figure 3.

The register file is designed utilizing the 18 Kbytes embedded BRAMs. Each BRAM is configured in simple dual port (SDP) mode with 1 read and 1 write port. In order to provide multiple ports, the BRAMs are organized into multiple banks and data is duplicated across various BRAMs. In this design, the number of write ports defines the number of banks and

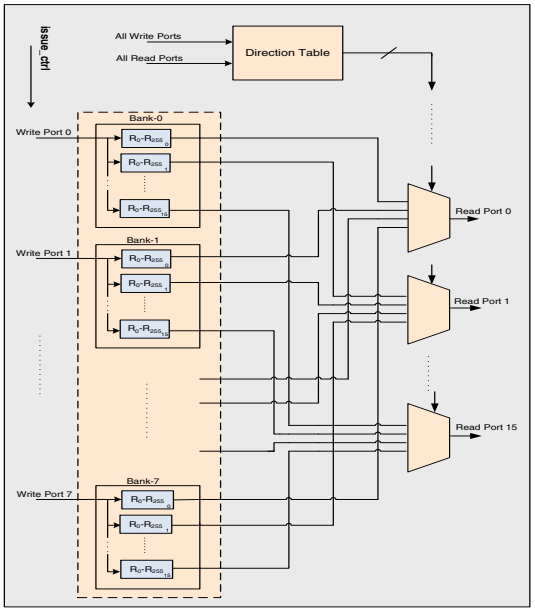


Figure 3. 256 × 32-bit 8W16R ports register file

the number of read ports defines the number of BRAMs per bank. The register file has 8 write and 16 read ports utilizing 128 BRAMs each providing a 256×32-bit register. Table I presents the utilization of this register file for different types of cores it can serve. Each port has 6-bit address to access 64 registers, but each BRAM has 8-bit address to provide 256 registers. The signal *issue_ctrl* is utilized inside the register file to generate the $7^{th}$ and the $8^{th}$ bits for the BRAMs. We utilize two such register files in our design as we have a total of eight 2-issue cores.

*3) Branch Register File:* The VEX ISA specifies a 1-bit 8-element multiported branch register (BR) file for a multi-issue VLIW core. For 2-issue, 4-issue, and 8-issue cores, we require branch register files with 2W2R ports, 4W4R ports, and 8W8R ports, respectively. Since the size of the branch register file is small, it is implemented utilizing the FPGA's slice registers and slice look-up tables (LUTs) instead of BRAMs. We implemented a 32×1-bit register file with 8 write and 8 read ports. Utilizing the *issue_ctrl* signal, we share the register file among the configured cores. The utilization of this register file is similar to that of the general-purpose register file. We utilize two such register files in our design as we have a total of eight 2-issue cores.

*4) Decode Unit:* Multiple 2-issue decode units can be stacked together to form a decode unit for a larger issue-width processor. The branch/CTRL unit which calculates the offset and the branch target addresses is included in every 2-issue decode unit, but only one branch unit is working when multiple 2-issue decode units are combined. Each 2-issue decode unit decodes its own long instruction (64-bit) and raises high its own *done* signal when the last VLIW instruction in the program (*STOP* instruction) is executed and the last result is written back. When a core is configured as a larger issue-width core, the combined decode unit provides only one branch unit and one *done* signal. The other *done* signals are tied to logic low. The signal *issue_ctrl* controls this mechanism.

### C. The Backend

The backend is that part of the system which remains fixed and does not change when the issue-width is changed. Eight 2-issue writeback units are stacked together. Each writeback unit can serve a 2-issue core and multiple writeback units can make a writeback unit for a larger issue-width core. Each lane of the writeback unit can write to its corresponding port on the general-purpose register file and/or the branch register file. Since these register files can handle the processor issue-width by themselves, the writeback unit does not need to take care of that, and hence, does not need reconfiguration/adjustment in order to combine or split the issue-slots. Backend consists of all the execution units. There are 16 ALUs, 16 MULs, and 8 MEM units which make up the backend. Every issue-slot has an ALU and a MUL unit and every two issue-slots have a MEM unit. The backend is the most resource-hungry part of our CMP.

Table I
UTILIZATION OF THE 8W16R PORTS REGISTER FILE

| Processor Type | Write Ports | Read Ports | Registers |
|---|---|---|---|
| One 8-issue | 0 - 7 | 0 - 15 | 0 - 63 |
| Two 4-issue | 0 - 3<br>4 - 7 | 0 - 7<br>8 - 15 | 0 - 63<br>64 - 127 |
| One 4-issue and two 2-issue | 0 - 3<br>4 - 5<br>6 - 7 | 0 - 7<br>8 - 11<br>12 - 15 | 0 - 63<br>64 - 127<br>128 - 191 |
| Four 2-issue | 0 - 1<br>2 - 3<br>4 - 5<br>6 - 7 | 0 - 3<br>4 - 7<br>8 - 11<br>12 - 15 | 0 - 63<br>64 - 127<br>128 - 191<br>192 - 255 |

## VI. RESULTS AND DISCUSSION

### A. Implementation Results

There are a total of eight 2-issue cores which can be combined to form larger issue-width cores. At maximum, four 2-issue cores can be combined together to form an 8-issue core. From these eight 2-issue cores, two 8-issue cores or a combination of 8-issue + 4-issue, 8-issue + 4-issue + 2-issue, 8-issue + 2-issue, or 4-issue + 2-issue cores can be built. The instruction and data memories are implemented with BRAMs. Each 2-issue processor or any larger issue-width processor in our design can run up to a maximum clock frequency of 107 MHz on a Virtex-6 *XC6VLX240T-1FF1156* FPGA available on the Xilinx *ML605* development board. Table II presents the resource utilization for our run-time adjustable CMP for the same FPGA. The values in the brackets in the last row in Table II represent the percentage resource utilization of the whole FPGA.

We tested our CMP with different applications. We utilized two benchmark suites. One benchmark suite represents the *MiBench* benchmark [22]. The other benchmark suite is a collection of different applications/kernels mostly of processing intensive types. For presentation purpose in this paper, we simply call the second benchmark suite *Benchmark 2*.

### B. Benchmark 1: MiBench

The MiBench benchmark suite [22] is a set of freely-available embedded programs for benchmarking purposes. It consists of different embedded applications divided into six suites with each suite targeting a specific area of the embedded market. The six categories are Automotive and Industrial Control, Consumer Devices, Office Automation, Networking, Security, and Telecommunications. All the programs are available as standard C source code. MiBench is portable to any platform that has compiler support. We took 8 applications representing different MiBench suites. We

Table II
RESOURCE UTILIZATION FOR THE RUN-TIME ADJUSTABLE CMP

| Module | Slice Registers | Slice LUTs | DSP48E1s | 18 kB BRAMs |
|---|---|---|---|---|
| Register File | 2 × 820 | 2 × 5887 | 0 | 2 × 128 |
| Backend | 2 × 692 | 2 × 8108 | 2 × 112 | 0 |
| 8-cores CMP | 5720<br>(1%) | 31267<br>(21%) | 256<br>(33%) | 256<br>(31%) |

executed these applications with three different configurations of our processor system, i.e., 8-issue, 4-issue and 2-issue cores. Figure 4 depicts the execution cycles for the three types of the processor systems. The execution cycles are normalized to that for an 8-issue core.

### C. Benchmark 2

Benchmark 2 is a self-made benchmark suite consisting of the following 9 embedded applications: Finite impulse response (FIR) filter, integer division, factorial of a small number, factorial of a large number, Floydwarshall graph, matrix transpose, matrix multiplication, integer square root, and a discrete Fourier transform (DFT) kernel. We executed these applications with three different configurations of our processor system, i.e., 8-issue, 4-issue and 2-issue cores. Figure 5 depicts the execution cycles for the three types of the processor systems. The execution cycles are normalized to that for an 8-issue core.

### D. Performance Analysis

We consider two use cases in order to show the effectiveness of our system.

*1) Use case 1:* In this case, the application is such that its data cannot be easily divided to be able to run on more than one cores. This scenario corresponds to applications with large instruction level parallelism (ILP) such as FIR filter, matrix multiplication, matrix transpose, DFT kernel, factorial, square root, bitstring, secure hash algorithm (SHA) etc. Generally, these kernels are part of some larger applications like *H.264/MPEG* audio/video, etc., and these kernels are executed multiple times while the application is running. Therefore, running such applications/kernels on a larger issue-width core can provide more performance compared to a smaller issue-width core. By combining multiple 2-issue cores to form a larger issue-width core (4-issue or 8-issue), we can exploit the available ILP in a better manner. Figure 6 depicts the instructions per cycle (IPC) for these applications. It can be observed that running these applications/kernels on a larger issue-width core can improve the performance of these applications/kernels. On the other hand, running these applications on a 2-issue core can benefit from the lower power consumption as the other 2-issue cores can be taken to a lower power mode, if these are not executing any other application.

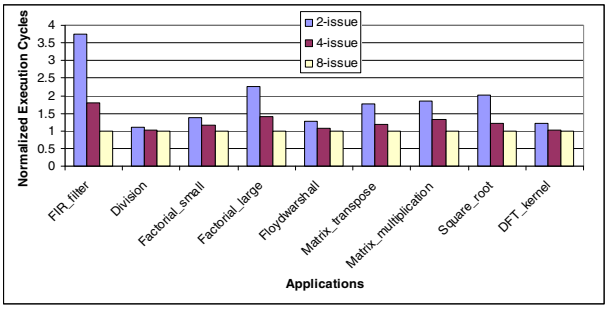


Figure 5. Normalized execution cycles for Benchmark 2

*2) Use case 2:* In this case, the application is such that its data set can be easily divided and can be run on more than one cores. This scenario corresponds to applications with large date level parallelism (DLP) such as the *Rijndael* encryption/decryption algorithm and the adaptive differential pulse-code modulation (ADPCM) encode/decode application. A matrix multiplication program can also get benefit by running on multiple independent cores with its data divided among the cores.

As a test case, we took the Rijndael algorithm. It is one of the most widely used encryption/decryption algorithms in cryptography. The Rijndael algorithm takes an input data of 128 bits and a key of 128, 196 or 256 bits and produces an encrypted output data of 128 bits. For decryption the same key is utilized as was used in the encryption process. We utilized a 128 bits key version of this algorithm. We encrypted and decrypted back a text of 2048 bytes using the Rijndael encryption/decryption algorithms. Initially, we run the application as a whole with 2048 bytes on a single 8-issue core. We then run the same application on two 4-issue cores with the data divided among the two cores. Each core encrypts/decrypts its own 1024 bytes of data. In the third experiment, we run the same applications on four 2-issue cores providing 512 bytes of data to each core. The individual encrypted/decrypted data can later be combined. Figure 7 depicts the execution cycles for the three types of the processor systems. The execution cycles are normalized to that for the four 2-issue cores. It can be observed from Figure 7 that applications with larger data level parallelism execute faster when run on multiple smaller issue-width cores with

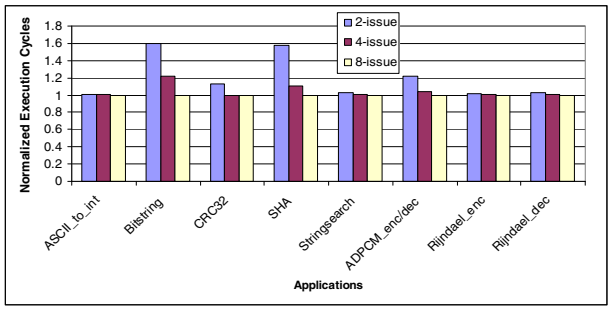


Figure 4. Normalized execution cycles for Mibench benchmark

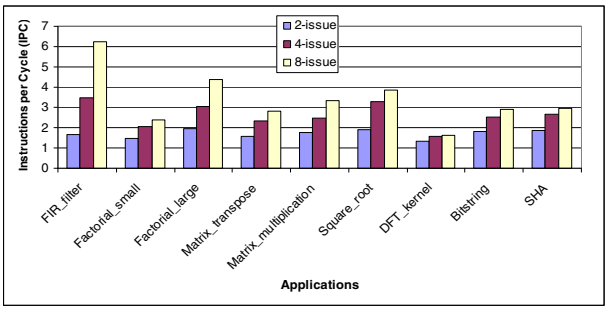


Figure 6. IPC for the higher ILP applications

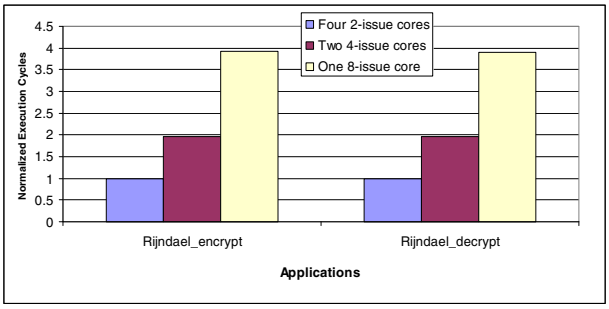


Figure 7. Normalized execution cycles for Rijndael algorithms

the input data divided among them compared to running the application on one larger issue-width core with all the input data.

### E. Power Analysis

We calculated the dynamic power consumption in our CMP. We designed our CMP such that each core clock is run by a separate controlled clock buffer [23]. By controlling each clock buffer, we can start and stop the clock running each core. The clock to a core is controlled by its *run* signal. If the *run* signal for a core is at logic low, the clock to that core is gated off. We utilized the Xilinx *XPower Analyzer* tool for the power calculation for the Xilinx Virtex-6 *XC6VLX240T-1FF1156* FPGA. We found that turning a 2-issue core off can reduce the dynamic power consumption of the whole system by about 9%. Hence, if any of the 2-issue cores is not executing an application, it can be turned off and the system can be taken to a lower power mode.

## VII. Conclusions

In this paper, we presented the design of a run-time adjustable chip multiprocessor (CMP) implemented in an FPGA. The processor instruction set architecture (ISA) is based on the VEX ISA. Our processor design is dynamically adjustable/reconfigurable. The processor has eight 2-issue cores, each of which can be run independently. If not in use, each core can be taken to a lower power mode by gating off the source clock. Multiple 2-issue cores can be combined at run-time to form larger issue-width VLIW cores. The CMP can target a variety of applications, and requires no additional programming effort or specialized compiler support. We showed that applications with larger instruction level parallelism (ILP) performed better when run on a larger issue-width core. While applications with larger data level parallelism (DLP) performed better when run on multiple 2-issue cores with the data distributed among the cores.

## References


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

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

[3] M. Koester, W. Luk, and G. Brown, "A Hardware Compilation Flow for Instance-Specific VLIW Cores", in *18th International Conference on Field Programmable Logic and Applications (FPL '08)*, pp. 619 - 622, 2008.

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

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

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

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

[8] S. Wong, T. van As, and G. Brown, "$\rho$-VEX: A Reconfigurable and Extensible Softcore VLIW Processor", in *IEEE International Conference on Field-Programmable Technologies (ICFPT '08)*, pp. 369 - 372, 2008.

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

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

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

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

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

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

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

[16] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore, "Exploiting ILP, TLP, and DLP wiith the Polymorphous TRIPS Architecture", in *International Symposium on Computer Architecture (ISCA '03)*, pp. 422 - 433, 2003.

[17] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart Memories: A Modular Reconfigurable Architecture", in *International Symposium on Computer Architecture (ISCA '00)*, pp. 161 - 171, 2000.

[18] E. Ipek, M. Kirman, N. Kirman, J.F. Martinez, "Core Fusion: Accommodating Software Diversity in Chip Multiprocessors", in *ACM SIGARCH Computer Architecture News*, pp. 186 - 197, vol. 35, Issue 2, 2007.

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

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

[21] C.E. LaForest and J.G. Steffan, "Efficient Multi-ported Memories for FPGAs", in *18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '10)*, pp. 41 - 50, 2010.

[22] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite", in *4th IEEE International Workshop on Workload Characterization (WWC '01)*, pp. 3 - 14, 2001.

[23] Xilinx, Inc. 2010. User Guide UG362: Virtex-6 FPGA Clocking Resources, http://www.xilinx.com.