A VLIW Softcore Processor with Dynamically Adjustable Issue-slots

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*—In this paper, we present a very long instruction word (VLIW) softcore processor implemented in an FPGA. The processor instruction set architecture (ISA) is based on the VEX ISA. The issue-width of the processor can be dynamically adjusted. The processor has two 2-issue cores, 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. The two 2-issue cores can be combined at run-time to form one larger 4-issue core. Applications/kernels with larger instruction level parallelism (ILP), such as matrix multiplication, FFT, DFT, etc., can be run on the larger 4-issue core to exploit the available ILP. Applications with more data level parallelism (DLP), such as AES encryption/decryption, ADPCM encode/decode etc., can be run on the two 2-issue cores with the data divided among the two cores. We utilize the Xilinx partial reconfiguration flow to implement our design. The size of the partial bitstreams to combine the two 2-issue cores to one 4-issue core or split vice versa is 59 kbytes. The minimum time required to reconfigure the processor or adjust the issue-slots are 0.893 ms and 0.148 ms for the Xilinx Virtex-II Pro and Virtex-4 FPGAs, respectively.**

I. Introduction

Field-programmable gate arrays (FPGAs) have become a widely used tool for rapid prototyping, providing software-like flexibility and hardware-like performance. To exploit instruction level parallelism (ILP), a very long instruction word (VLIW) processor can be utilized to increase the performance beyond a single issue-width processor [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 VLIW processor is implemented in an application-specific integrated circuit (ASIC), its issue-width is fixed at design time. Therefore, the issue-width of the processor can not be adjusted after fabrication to suit a different set of applications. 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 parameter of a 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.

In [8][9], we presented the design and implementation of a parameterized and extensible softcore VLIW processor called ρ-VEX. In this paper, we extended this design-time reconfigurable processor to support run-time partial dynamic reconfiguration. After the processor is implemented in an FPGA, it can adjust its issue-width while other circuits in the FPGA keep running. Only a small partial bitstream is loaded to adjust the issue-slots. The processor has two 2-issue cores which can be run independently and these can be combined to form a larger 4-issue core or split vice versa. Hence, before an application starts executing, the machine organization can be changed to match the application. Applications/kernels with more fine-grain (instruction level) parallelism such as a discrete fourier transform (DFT) kernel can be run on the larger 4-issue machine for better performance, and applications with more coarse-grain (data level) parallelism such as an encryption algorithm with a specific amount of data can be run on the two 2-issue processors in a single instruction multiple data (SIMD) fashion for faster execution.

The remainder of the paper is organized as follows. The related work is discussed in Section II. Section III describes the VEX system and the ρ-VEX VLIW processor. Section IV presents the design of our dynamically reconfigurable VLIW processor. The implementation details and application development framework are presented in Section V. Results are discussed in Section VI. Finally, conclusions are presented in Section VII.

II. Related Work

Spyder [2] appeared as the first softcore VLIW processor. The provided toolchain was not complete and the processor was not run-time reconfigurable. The VLIW processors presented in [3] are instance-specific implementations and hence do not represent more general VLIW processors. An FPGA-based design of a softcore VLIW processor based on the ISA of the Altera NIOS-II soft processor is presented in [4]. The compilation scheme consists of a Trimaran [10] as the front-end and the extended NIOS-II as the back-end. Due to the licensed Altera NIOS-II, this VLIW design is not much flexible and not open-source. Additionally, the design is not run-time reconfigurable. In [5], a modular design of a VLIW processor

is reported. Certain parameters of the processor architecture could be altered in a modular fashion. The lack of a good software toolchain and the absence of parametric extensibility limited the use of this architecture. In [6], the architecture and micro-architecture of a customizable softcore VLIW processor are presented. Additionally, tools are discussed to customize, generate, and program this processor. The limitation is the absence of a compiler. A VLIW processor with reconfigurable instruction set is presented in [7]. In this case, a reconfigurable unit is coupled to a VLIW processor. The co-processor can be configurable for any custom instruction. We are different from this design in the sense that we do not couple a reconfigurable co-processor. We can add a custom unit to the general data path of our processor at design time and reconfigure the issue-slots at run-time. In [8][9], we present the rationale and the design and implementation of an open-source softcore VLIW processor. This processor is design-time parameterized and now we have extended it to make its issue-width run-time reconfigurable/adjustable.

III. The Base Processor System

A. The VEX system: ISA and Toolchain

The VEX stands for VLIW Example [11]. 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 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 [12].

B. The ρ-VEX VLIW processor

The ρ-VEX is a configurable (design-time) open-source VLIW softcore processor [8]. The ISA is based on the VEX ISA [11]. Different parameters of the ρ-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 ρ-VEX VLIW processor implemented in an FPGA. The ρ-VEX processor consists of *fetch*, *decode*, *execute,* and *writeback* stages/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. In the decode unit, instructions are decoded and the register contents used as operands are fetched from the register file. The actual operations take place in either the execute unit, or in one of the parallel *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

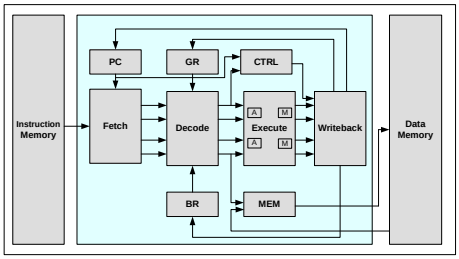

Figure 1. 2-issue ρ-VEX VLIW processor

operations are handled by the MEM unit. All write activities are performed in the writeback unit to ensure that all targets are written back at the same time. The different write targets could be the *general register* (GR) file, *branch registe*r (BR) file, *data memory* or the *program counter* (PC). Additionally, the ρ-VEX processor supports reconfigurable operations, as the VEX compiler supports the use of custom instructions via pragmas within an application code. The instruction and data memories for the processor are implemented with BRAMs. The ρ-VEX processor is utilized as a base processor in the design of our dynamically reconfigurable processor.

IV. Dynamically Reconfigurable Processor Design

In this section, we present the design of our reconfigurable processor. Figure 2 depicts the issue slots in our processor system. Each 2-issue machine has two ALUs and two MULs. There is also a MEM unit as well as a CTRL unit. A 4-issue machine has double the resources of a 2-issue machine except that only one of the CTRL units is utilized when the two 2-issue cores are combined. A signal called as the *issue_ctrl* helps in controlling the issue-width of the processor. When this signal is low, the two 2-issue cores can be utilized independently. When this signal is high, the two 2-issue cores are combined and they behave like a single 4-issue machine with double the resources of a 2-issue machine. If any of the two 2-issue cores is not utilized by an application, it can be taken to a lower power mode by turning off its source clock. In this way, the dynamic power consumption of that core is reduced resulting in a reduced total power consumption of the system. The signal *power_ctrl* is utilized for gating off the source clocks of the two 2-issue cores for power control. The signals *issue_ctrl* and *power_ctrl* can be controlled by some higher level controller or scheduler, which will also handle the reconfiguration process.

Each core (2-issue or 4-issue) consists of different units, namely *fetch*, *decode*, *execute*, and *writeback* as depicted

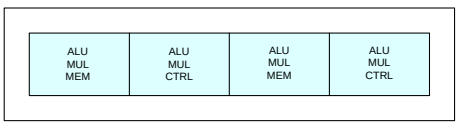

Figure 2. Execution units in different issue-slots

in Figure 1. For the purpose of making the processor run-time adjustable, we divided these units into two sections, namely *frontend* and *backend*, as depicted in Figure 3. The frontend consists of the modules which needs reconfiguration/adjustment in order to combine or split the issue-slots. It includes the fetch, decode, writeback, general-purpose register (GR) file and the branch register (BR) file. The backend consists of the execution units which do not require reconfiguration when switching from two 2-issue cores to one 4-issue core or vice versa.

A. Frontend

The frontend of our reconfigurable processor is reconfigured/adjusted at run-time for changing the issue-slots of the processor. It consists of the fetch, decode, and writeback units, and the multiported general-purpose register file and the branch register file. Out of these five modules, the decode unit is the only module that is reconfigured by loading a partial bitstream to switch between two 2-issue cores to one 4-issue core or vice versa. The other four modules are controlled by the *issue_ctrl* signal, and they do not require a partial bitstream to reconfigure/re-adjust. This is done in order to minimize the number of resources to be reconfigured and hence minimize the size of the partial bitstream. This resulted in reduced configuration time as well as reduced memory storage for the partial bitstreams.

1) Fetch Unit: A 2-issue fetch unit simply splits the incoming long instruction into two syllables (instruction (32-bit) for individual execution unit), and then passes them to the decode unit. Therefore, two 2-issue fetch units can be stacked together to form a combined fetch unit to behave like two independent 2-issue fetch units or one 4-issue fetch unit. Each 2-issue fetch unit has a program counter (PC). The only sub-unit of the fetch units that need to be altered/reconfigured is the PC. If the combined fetch unit is to be utilized as two independent 2-issue fetch units, the two program counters are running independently. If the combined fetch unit is to be utilized as a single 4-issue fetch unit, only one of the program counters is running and the other one is stopped. The *issue_ctrl* signal 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 processor. In the case of a 2-issue core we require a register file with 2-write and 4-read (2W4R) ports, while for a 4-issue core we require a register file with 4-write and 8-read (4W8R) ports. The register file is one of the most resource consuming module of the processor when it is implemented utilizing the FPGA's slice registers [8]. We implemented our register file utilizing BlockRAMs (BRAMs) based on the design presented in [13]. We need two 2-issue register files when both the cores are utilized as independent 2-issue cores or one 4-issue register file when the two 2-issue cores are combined to form one 4-issue core. We designed our register file in a manner that a single register file can handle a 4-issue core or two 2-issue cores at the same time. The register file is depicted in Figure 4.

The register file has 4-write and 8-read ports utilizing 32 BRAMs each providing 128 registers of 32-bit each. If the system is configured as a 4-issue core, all of the ports and the lower 64 registers are utilized. If the system is configured as two 2-issue cores, half of the ports are utilized by one core and the second half by the other core. The lower 64 registers are utilized by one core and the upper 64 by the other core. Each port has 6-bit address to access 64 registers, but each BRAM has 7-bit address to provide 128 registers. The signal *issue_ctrl* is utilized inside the register file to generate the 7^{th} bit for the BRAMs. By using this mechanism we avoided the register files to be reconfigured by loading the partial bitstreams and hence reduced the size of the partial bitstreams required to alter the organization of the processors. Table I presents the resource utilization for our register file for the Xilinx Virtex-II Pro *XC2VP30* FPGA.

3) Branch Register File: The VEX ISA specifies a 1-bit 8-element multiported branch register (BR) file for a multi-issue VLIW processor. For a 2-issue core, we require a branch register file with 2-write and 2-read (2W2R) ports and for a 4-issue core, we require a branch register file with 4-write and 4-read (4W4R) ports. Since the size of this register file is small,

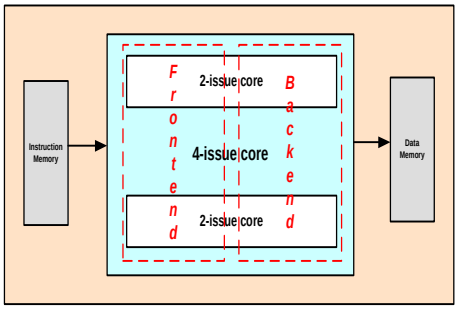

Figure 3. Frontend and backend of the reconfigurable processor

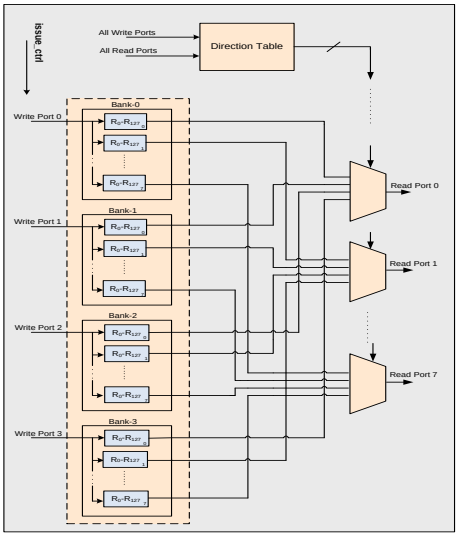

Figure 4. 128 × 32-bit 4W8R ports register file

Table I
RESOURCE UTILIZATION FOR THE 4W8R PORTS REGISTER FILES

Register File Design	Slices	BRAMs
64×32-bit 4W8R Ports (slice-based)	8594	0
128×32-bit 4W8R Ports (BRAMs-based)	1060	32

it is implemented utilizing slice registers instead of BRAMs. We implemented a 16×1-bit register file with 4W4R ports. Utilizing the *issue_ctrl* signal, we partition the register file among the configured cores. When the system is configured as one 4-issue core, all of the ports and the lower 8 registers are utilized for the branch register file. When the system is configured as two 2-issue cores, half of the ports and the lower 8 registers make the branch register file for one core and the other half ports and the upper 8 registers make the branch register file for the second core. The signal *issue_ctrl* handles this mechanism. The branch register file requires 81 Virtex-II Pro *XC2VP30* FPGA slices.

4) Writeback Unit: The writeback unit has four lanes, which can be utilized for a 4-issue core or two 2-issue cores. Each lane can write to its corresponding port on the general-purpose or branch register files. Since these register files can handle the processor issue-width by themselves, the writeback unit does not need to take care of that. The writeback unit only takes care of the PC. If the system is configured as a single 4-issue core, lane0 of the writeback unit writes to the only PC of the system. If the system is configured as two 2-issue cores, lane0 of the writeback unit writes to the PC of the first core, and lane3 writes to the PC of the second core. The signal *issue_ctrl* handles the mechanism to switch between the two cases, and hence there is no need to reconfigure/re-adjust the writeback unit by loading a partial bitstream.

5) Decode Unit: The decode unit is reconfigured by loading a partial bitstream. Separate bitstreams are utilized to switch the two 2-issue cores to one 4-issue core or vice versa. The branch/CTRL unit which calculates the offset and the branch target addresses is included in the decode unit. The decode unit for the 4-issue core has only one CTRL unit, while the decode unit for the two 2-issue cores has two CTRL units. When the system is configured as two 2-issue cores, the configured decode unit provides two 2-issue decode units with separate CTRL units. Each decode section decodes its own long instruction (64-bit) and raises high its own *done* signal when the last instruction in the program (STOP instruction) is executed and the last result is written back. When the system is configured as a single 4-issue core, the configured decode unit provides only one 4-issue decode unit with one CTRL unit. This 4-issue decode unit decodes the incoming long instruction (128-bit) and raises high the first *done* signal when the last instruction in the program (STOP instruction) is executed and the last result is written back. The second *done* signal is this case is tied to logic low. The decode unit for the two 2-issue cores with two CTRL units requires 355 Virtex-II Pro *XC2VP30* FPGA slices, while that for one 4-issue core with one CTRL unit requires 293 slices. According to the Xilinx *early access partial reconfiguration (EAPR)* design methodology [14], we have to allocate area for the largest size module, and hence the size of the partial bitstream would be according to the size of the largest module (355 slices not 293 slices). Further details are given in sections V and VI.

B. The Backend

The backend of our reconfigurable processor does not require re-adjustment when the issue-width is switched from 2-issue to 4-issue or vice versa. In terms of the EAPR design methodology, it is a part of the *static* region of the design. The backend consists of execution units. There are 4 ALUs, 4 MULs, and 2 MEM units. There are 4 lanes. Each lane has an ALU and a MUL unit. Each of lane1 and lane3 has a MEM unit. When the system is configured as two 2-issue cores, lane0 and lane1 constitute the execution units for the first core and lane2 and lane3 constitute the execution units for the second core. When the system is configured as a 4-issue core, all of the four lanes constitute the execution units for the 4-issue core. In this case, all of the ALU, MUL, and MEM units can be utilized by the 4-issue core. Figure 2 depicts these lanes with the available execution units. The backend is the most resource-hungry part of the processor. It requires 8265 Virtex-II Pro *XC2VP30* FPGA slices.

V. DYNAMICALLY RECONFIGURABLE PROCESSOR IMPLEMENTATION

In general, partial dynamic reconfiguration is utilized to time share a specific area of a chip in order to reduce the total resource requirement of a design. Our purpose of utilizing the partial dynamic reconfiguration is not to reduce the required area; rather we utilized it for the purpose of changing the functionality of a system at run-time and for making a better utilization of the available resources based on the incoming application. Therefore, we made the reconfigurable region to be as small as possible, such that the size of the partial bitstreams could be as small as possible. We utilized the Xilinx EAPR methodology [14] for designing our reconfigurable processor system with adjustable issue-slots. We split our design into two regions, namely *static* and *reconfigurable* regions. Figure 5 depicts the static and dynamically reconfigurable regions of our design. The processor system consists of the frontend, backend, instruction memory, data memory, and a universal asynchronous receiver transmitter (UART) module. Except the decode unit in the frontend, all other modules in the frontend, backend, memories, and UART are placed in the static region as they do not need partial dynamic reconfiguration. The decode unit in the frontend is placed in the reconfigurable region.

The static and reconfigurable regions are connected utilizing *bus macros*. In this design, we have a total of 920 signals that need to cross the boundary between the static and reconfigurable regions and hence a total of 115 bus macros are required to provide the communication between the two regions.

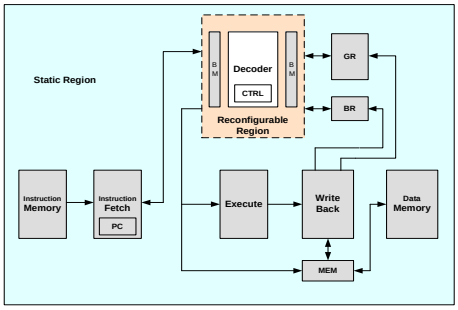

Figure 5. Static and reconfigurable regions

Using the EAPR design methodology, partial bitstreams for the decode units are generated and can be downloaded to the FPGA using the Xilinx iMPACT tool. A reconfiguration controller can be implemented and can be directed by some higher level scheduler to reconfigure the issue-slots of the processor in order to provide the best performance for the incoming application, by utilizing the internal configuration access port (ICAP) in the FPGA.

A. Application Development Framework and Testing

To optimally utilize our processor, we present an application development framework. An application program written in C is compiled with the VEX compiler to generate the VEX assembly. This assembly is passed through the ρ-VEX assembler [8] to generate the ρ-VEX binaries/executables. Since the VEX is a VLIW processor system, separate binaries are generated for the 2-issue and 4-issue machines. For testing purposes, we utilize an UART module, which is part of our processor system. The program binaries can be loaded from a personal computer into the instruction memories of the processor implemented in an FPGA utilizing the UART module. The same UART is utilized to transmit results of a program execution in the FPGA processor to a personal computer that can be visualized for testing and debugging purposes.

VI. Results and Discussion

A. Implementation Results and Performance Analysis

The reconfigurable region has 920 signals crossing the boundary between the static and reconfigurable regions and hence requires 115 bus macros. The reconfigurable region is constrained to a total of 560 slices, in order to contain all the required resources and provide space for the bus macros. Table II presents the resource utilization for our design. The reconfigurable region includes only the decode unit of the frontend section. The static region includes rest of the frontend, the backend, the instruction and data memories, and the UART. The processor is running at 50 MHz but can run correctly up to a maximum of 70 MHz in a Virtex-II Pro *XC2VP30* FPGA.

The partial bitstream size for the reconfigurable region is 59 kbytes, and is about 24 times smaller than the full bitstream size which is about 1415 kbytes. The width of the ICAP in the Virtex-II Pro and Virtex-4 FPGAs is 8 bits and 32 bits, respectively. The maximum frequency for the ICAP in the Virtex-II Pro and Virtex-4 FPGAs is 66 MHz and 100 MHz, respectively. The minimum time needed to switch between two 2-issue cores to one 4-issue core or vice versa is 0.893 milliseconds for a Virtex-II Pro device, and 0.148 milliseconds for a Virtex-4 device. This value does not include the time needed for accessing the memory in which the bitstreams are placed. It is the time needed for the SelectMAP or ICAP to configure the device. At 50 MHz clock, these reconfiguration times translate to a total of 44650 clock cycles for a Virtex-II Pro device and 7400 clock cycles for a Virtex-4 device.

We executed different applications on our reconfigurable processor. We considered two scenarios, one that could better exploit a 4-issue core and other that could better exploit the utilization of two 2-issue cores.

1) Application Scenario 1: In this scenario, the application is such that its data can not be easily divided to be able to run on more than one cores. This scenario corresponds to applications/kernels with large ILP such as a *matrix multiplication* program or a *discrete fourier transform (DFT)* kernel. Generally, these kernels are part of some larger applications like MPEG video, etc., and these kernels are repeated many times while the application is running. Therefore, running such applications/kernels on a larger issue-width core can provide more performance as compared to a smaller issue-width core. Hence, in our case we can combine the two 2-issue cores to form one 4-issue core and exploit the available ILP. We created a C program that does a 100-by-100 matrix multiplication. We executed the same program on a 4-issue and a 2-issue core. Figure 6 depicts the normalized performance for the two types of cores. In this figure, *Issue_2_1* means the application is running on one of the two 2-issue cores, and *Issue_4* means that the application is running on the combined 4-issue core. We executed a DFT kernel on both a 4-issue and a 2-issue core. Figure 6 depicts the normalized performance for the two types of cores. It can be observed from this figure 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 core can be taken to a lower power mode, if it is not executing any application.

2) Application Scenario 2: In this scenario, 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 *advanced encryption standard (AES)* encryption/decryption

Table II
Resource utilization for the reconfigurable processor

Region	Slices
Reconfigurable Region	560
Static Region	9944
Bus Macros	230

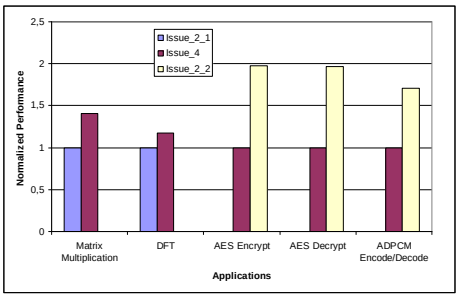

Figure 6. Normalized performance

and the *adaptive differential pulse-code modulation (ADPCM)* encode/decode. The AES algorithm is one of the most widely used encryption algorithms in cryptography. The AES algorithm takes an input data of 128 bits and a key of 128, 196 or 256 bits and produces an encrypted output data of 128 bits. For decryption the same key is utilized as was used in the encryption process. We utilized a 128 bit key version of the AES algorithm. We encrypted and decrypted a text of 1024 bytes. For the single 4-issue core, the C program for the encryption and decryption are compiled and assembled with the input data of 1024 bytes. For the two 2-issue cores, the input data is split into two sets each of 512 bytes. Each core is provided its own data set and the same program for encryption/decryption runs on it. Figure 6 depicts the normalized performance for the two types of cores. In this figure, *Issue_2_2* means the application is running on both of the two 2-issue cores, and *Issue_4* means that the application is running on the combined 4-issue core. It can be observed from Figure 6 that the two 2-issue core system completed the execution of the whole application in almost half the time compared to the single 4-issue core system.

The ADPCM audio encoding/decoding technique is utilized for the encoding/decoding of audio (voice) in voice over internet protocol (VoIP) telephony applications. We utilized an application that encoded and then decoded 10000 samples. We run the application on the single 4-issue core. We then divided the data into two parts and run the same application on both the two 2-issue cores with the divided data sets. Figure 6 depicts the normalized performance for the two types of cores. It can be observed from Figure 6 that the two 2-issue core system completed the execution of the whole application in almost half the time compared to the single 4-issue core system.

B. Power Analysis

We calculated the dynamic power consumption for our reconfigurable processor design utilizing the Xilinx *XPower Analyzer* tool. We executed a small program that computes the 45^{th} element of the Fibonacci series on a 4-issue core and a 2-issue core with the source clock for the other 2-issue core tied to logic low. We run these experiments at 50 MHz. We found that turning one of the two 2-issue cores off can reduce the dynamic power consumption of the system by up to 42%. Hence, if one of the two 2-issue cores is not running some 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 and implementation of a run-time reconfigurable/adjustable softcore VLIW processor. The processor has two 2-issue cores, 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. The two 2-issue cores can be combined at run-time to form one larger 4-issue core or split vice versa. We showed that applications/kernels with larger ILP achieved better performance when run on the larger 4-issue core. While applications with larger data level parallelism (DLP) showed better performance when they were run on the two 2-issue cores with the data divided among them. We utilized the Xilinx partial reconfiguration methodology to implement our design. The size of the partial bitstreams to combine the two 2-issue cores to one 4-issue core or split vice versa is 59 kbytes. The minimum time required to reconfigure the processor or adjust the issue-slots are 0.893 ms and 0.148 ms for the Xilinx Virtex-II Pro and Virtex-4 FPGAs, respectively.

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] 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.

[6] 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.

[7] 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.

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

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

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

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

[13] C.E. LaForest, 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.

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