Design of a Pipelined and Parameterized VLIW Processor: ρ-VEX v2.0

Roël Seedorf, Fakhar Anjam, Anthony Brandon, and Stephan Wong

Computer Engineering Laboratory, Delft University of Technology
The Netherlands
{r.a.e.seedorf, f.anjam, a.a.c.brandon, j.s.s.m.wong}@tudelft.nl

Abstract. In this paper, we describe an improved design of a VLIW processor that aims to bridge the gap between application-specific and general-purpose processors. Our discussion focuses on the new design and implementation of the reconfigurable and parameterized VLIW processor called ρ-VEX by introducing pipelining and forwarding logic. Our processor is parameterized in the issue-width, the type and number of functional units, the size of the multi-ported register files, the size of its memories, and in the presence of forwarding logic. It can execute custom assembly operations or run profile-driven compiler optimized code. Our experimental results encompass resource utilization of our designs (including register file optimizations) and execution times on a Virtex-6 FPGA. The execution times are compared to its multi-cycle predecessor and the MicroBlaze for 6 selected benchmarks. The results show that our new design runs 6.64 times faster than the multi-cycle design, and that forwarding provides performance gains between 1.32 and 2.93 over the (unforwarded) implementation. Finally, our initial results demonstrate that our forwarded 4-issue ρ-VEX processor running at 150 MHz, performs between 1.04 and 2.72 times faster than the speed-optimized MicroBlaze processor running at 200 MHz.

1 Introduction

In the embedded systems market, we are clearly witnessing a trend in which application-specific circuitry is being replaced by general-purpose processing platforms. This is a clear departure from the past as embedded systems are traditionally inherent application-specific as they only need to support a limited set of applications or application domains. The current-day push for quicker and cheaper embedded systems means that the lengthy and "expensive" design cycles of the past (averaging 18 month) are no longer viable. Therefore, in the design of such systems, increasingly more general-purpose computing platforms are being utilized as they can be "re-used" over several product generations and thereby amortize their initial design cost over a much larger amount of products. For this purpose, two types of processors have become candidate as the core for the processing platforms. The first type are derivatives from out-of-order superscalar RISC processors that have been used. They are flexible in the sense that they can

be easily reprogrammed to support different applications, but they have several disadvantages. First, there is a lot of control overhead to correctly sequence the code execution on these processors leading to wasted power consumption. Second, to increase performance (by fully utilizing all available hardware) complex instruction fetch and decode mechanism are needed and in turn adding more to the power consumption. Third, in order to make them more power efficient, new instructions are commonly introduced, but this requires a large amount of effort in adapting the existing tools and compilers to take full advantage of these instructions. The second type are VLIW processors that have gained a foothold in embedded systems as they rely on compilers to schedule instruction execution and thereby alleviating the first and second disadvantages of RISC processors, therefore resulting in much more power efficient designs. However, they suffer from the same drawback as their counterparts in the general-purpose computing domain as different applications will not always result in the best resource(s) utilization. Consequently, we proposed a parameterized reconfigurable VLIW processor that tackles the problem of instruction under-utilization for VLIW processors when these run applications with less ILP than was anticipated during their design [7]. Instead of generating code that runs as efficient as possible on a specific fixed implementation, we reconfigure our processor organization depending on the applications requirements. For now, we mainly focus on the most obvious relation between the applications inherent ILP and the processors issue-width. We have chosen the VEX VLIW instruction set architecture (ISA) as it is supported by an industrial-strength parameterized compiler with a cycle-accurate simulator [3]. In this paper, we present further optimizations and extensions to the new designed ρ-VEX processor (v2.0) that includes pipelining and forwarding logic that should give better performance than the first ρ-VEX implementation. First, we compare its performance with the MicroBlaze softcore from Xilinx (both running at their maximum clock frequencies). Moreover, we extended the existing assembler to a full toolchain by including a linker and a loader attached to a VHDL generator. The latter is used to instantiate the instruction and data memories. Finally, we investigate the performance of our new design with an without forwarding logic.

The remainder of this paper is organized as follows. Section 2 describes the related work and motivates our research. Section 3 describes the the architecture and the organization of the ρ-VEX processor. Section 4 presents the parameterization of the ρ-VEX processor and discuses several optimizations to reduce its reconfigurable resources. Section 5 describes the experiments that were performed and presents the results. Finally, Section 6 concludes this paper.

2 Related Work and Motivation

The EPIC-Explorer is based on the EPIC architecture and presents a framework for simulating a parametrized System-on-Chip (SoC) with a main VLIW processor [4]. This framework allows a designer to assess the dissipated power, the number of execution clock cycles, and the area that an application requires for

execution on an instance of the SoC. The proposed framework has the ability to change the register file (RF) sizes or the number and type of functional units (FUs) that are supported, but it lacks the possibility to change the processor's issue-width. This has a considerable impact on the amount of ILP that the EPIC core can execute for a given application.

The Lx technology platform is a multi-cluster VLIW processor [5]. It incorporates a clustered VLIW processor that was developed as an industry collaboration between HP and STMicro. This delivered a scalable and customizable platform for image applications. Furthermore, a tool chain is utilized that is based on the Multiflow compiler supported by a simulator for performing design space exploration (DSE). The LX core implemented in a 0.25 micron technology, could run up to 400 MHz.

The first ρ-VEX design had a single cluster with 4-issue slots [7]. The processor is based on the VEX ISA and has a multi-cycle design implemented in a Xilinx Virtex 2 FPGA. The processor's has four processing stages and its purpose was to function as a programmable accelerator within the Molen framework [7]. Furthermore, it has the ability to extend the ISA with user custom operations that could be added to the ALU. Moreover, the project resulted in an assembler that created an instruction ROM. The design's main drawback was its multi-cycle micro-architecture as this limits the *temporal* ILP that can be executed because consecutive instructions finish sequentially.

Previous research showed that a VLIW processor can function as the processing engines within an embedded MPSoC [1]. For a given set of applications, the optimal size of the architecture of the processors must be investigated. Our work advocates the use of configurable issue-width processors that share a base instruction set called VEX [3]. After investigating this design, we concluded that the nature of its architecture limits the performance that can be attained for applications with high amounts of ILP [6]. Except for the size of its memories, the processor was not parameterized in its issue-width and the position of its functional units (FUs) were fixed. To address these problems, we propose a new design that is pipelined and parameterized. For a target application, this results in a convenient way to instantiate the processor with the aim of getting better instruction throughput. Moreover, we implemented a tool chain that consists of an assembler, linker, and a loader. This is utilized for conveniently generating the instruction and data memories.

3 ρ-VEX Architecture and Organization

The VEX ISA calls a encoded operation a syllablle, and it defines an instruction as a set of syllables [3]. In VEX operations are equivalent to 32 bit RISC instructions. Two sets of rules must be satisfied in the micro-architecture: (1) a set of *common* rules, and (2) a set of *specific* rules. The first set must be adhered to by every processor instance. This includes the ISA, the registers connectivity, the memory coherency, and the architectural state. *Specific* rules define different features between processor instances. These are the issue-width, the latency and

behavior of functional units (FUs), or the size of the register files (RF) and the memories. Both sets constrain the design space such that a processor architect can explore and determine the desired specifics of the target architecture. The 4-issue processor instance utilizes the instruction layout depicted in Figure 1. This figure illustrates how syllables are mapped to sets of FUs.

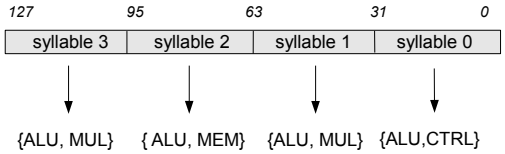

Fig. 1. Instruction Layout – Adapted from [7]

3.1 Processor Architecture

The ρ-VEX processor has a Load/Store Harvard architecture with multiple issue-slots. Its micro-architecture is pipelined with 5 stages. The processor has 4 types of FUs, namely: ALU, Multiplier, Load/Store, and Branch. More specifically, there are four 32-bit ALUs, two 16×32 bit Multipliers, one Load/Store Unit, and one Branch Unit (BU). In addition, the architecture defines and a 64×32-bit general-purpose register file, and a branch register file (BR) with 8×1-bit branch registers that store branch conditions, predicate values and the carries from arithmetic operations. The number of each type of FU, except for the BU's issue slot, is adjustable. This allows us to customize the processor based on the the target application(s) requirements. For example by changing the issue-width, we can adapt the number and the type of processor resources for applications with limited or more amounts of spatial ILP. There are 70 native VEX operations divided over 5 classes: 39 ALU operations, 11 multiplication operations, 9 memory operations, 9 control operations, and 2 inter-cluster operations. ALU operations are partitioned into arithmetic and logical operations. Our un-clustered architecture does not require inter-cluster move operations. We do support additional syllables called LONG_IMM and STOP. The former is used to support 32-bit immediate by using a second free syllable slot for the high immediate part. The STOP syllable triggers the end of an application's execution.

3.2 Processor Organization

The pipeline of all ρ-VEX processor instances consists of five stages: *Fetch*, *Decode*, *Execute 1*, *Execute 2*, and *WriteBack*. An overview is depicted in Figure 2. The micro-architecture has a multiple-issue pipeline. We define each pipeline issue as a pipe-lane and call the pipeline registers by the stages that they separate. In the following discussion, we are assuming a 4-issue organization.

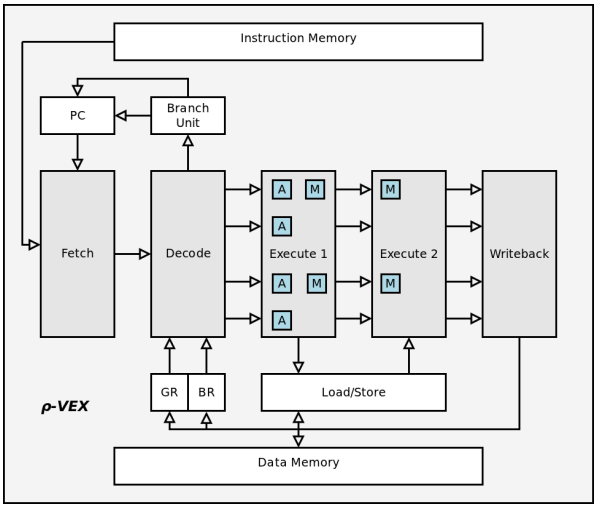

Fig. 2. ρ-VEX Pipeline Organization

3.3 Fetch Stage

This stage performs three micro-operations: Address Generation, Instruction Access, and Synchronization. The Address Generation Unit (AGU) computes the next instruction address that the program counter sends to instruction memory. Subsequently, the instruction are fetched and stored in the IF/ID pipeline register. The AGU contains a sequencer and the selection logic to choose between the program counter and the branch target. Upon decoding a branch syllable, the Branch Unit controls the selection of the branch target. For a taken branch, the desired instruction is fetched from the branch target and the Synchronization Unit flushes the fetched instruction. This unit provides the ability to halt the pipeline when the instruction stream ends, it flushes the pipeline, and signals subsequent stages to stop processing instructions.

3.4 Decode Stage

This stage receives a quad-word bundle and splits it into four 32-bit syllables. Each syllable is decoded in parallel to form an operation from which register identifiers are extracted to read the RFs. The read values will form the operands in the first Execution Stage. This stage contains the following components: the Decode Unit, the Branch Unit (BU), the General-Purpose Register File (GP), the Branch Register File (BR), and a Bypass Network. Moreover, it performs immediate generation. Pipe-lane 0 decodes with the Branch Unit all control operations. The Branch Unit receives the opcode, control information, and its source operands. Placing the BU here gives the micro-architecture a single-cycle miss-penalty. Furthermore, our *two-step* branch architecture decouples compare operations from the actual branch, which in collaboration with the Bypass Network, exposes a single-cycle branch latency to the compiler (see Figure 3). Upon

decoding a conditional branch syllable, the pre-stored condition from the branch register is read and controls the instruction selection directly in the AGU. This allows to prepare execution of multiple branches [3]. This stage also contains the pipeline controller distributed over each pipe-lane. With the Decode Unit, it generates control signals (micro-operations) for the pipeline stages. Due to its parallel and concurrent functionality, it is the most sophisticated stage, which differs from the multi-cycle design that has a separate controller with inter-stage signaling in each processing stage [7]. In the current design all stages execute their operations in parallel *and* concurrently. This allows it to cope with both temporal and spatial ILP of applications. As the register files store the architectural state of the processor, their designs are critical because the WriteBack (WB) stage and the Decode stage may exhibit read-after-write hazards.

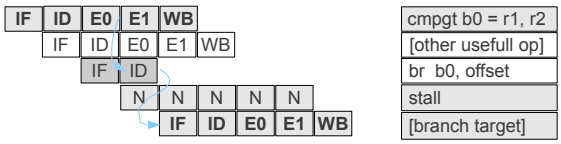

Fig. 3. Two-step Branch Architecture

3.5 Execute 1 Stage

This stage is where the functional units produce their results: 4 parallel ALU, 2 MUL Units in parallel with 2 ALU and 1 Load/Store Unit in parallel with 1 ALU. More specifically, ALU or MUL operations and the *effective* memory address are computed in parallel. In pipe-lane 0 the Branch Unit updated the link register in the second stage. This stage propagates control signals from the pipeline controller and the data that are required by subsequent stages. The operands and their opcodes arrive at every functional unit for execution. Except for the MUL units, these units produce results to be latched in E1/E2 pipeline register. The MUL Unit consits of two 32×16-bit multipliers that have 2 stages.

3.6 Execute 2 Stage

This stage receives the results of the FUs and does a pre-selection of the run-time values for the WB stage. The branch flag is placed on the ALU's carry output. In lanes 3 and 1, the result come from the MUL unit, which adds the partial results. As we have mapped the opcode values such that operation classes form disjoint sets among operation types, we test the opcodes of each operation to

determine which set it belongs, in order to pass the right functional unit's output. Memory operations are mapped to pipe-lane 2. The data memory address and the byte(s) position(s) indicate the location to read/write data. By asserting the Load/Store's control signals correctly, we read the target value from data memory and latch it in the E1/WB pipeline register with the other results.

3.7 WriteBack Stage

This stage controls committing the results of the previous stages to the register files. Basically, it receives results, addresses, and enable signals that indicate which are the target register file(s) in a lane. Registers in the Decode stage with RAW hazards benefit from Register File Forwarding [6]. In the current design, we perform this in an explicit way, which means writes to the register files happen on the rising clock-edge. There is forwarding logic within the register files that detects when the Decode Stage reads a register that is being updated by the WriteBack (WB) stage. When this happens this logic forwards the updated value to the outputs of the register file. For the forwarded instances similar external logic exists between the WriteBack stage and the operands of the first Execution stage, and between the operands of the Branch Unit and the ALU results.

3.8 Bypass Network

Our parameterized design can instantiate forwarded and un-forwarded instances of the base processor. The forwarded instances utilize a bypass network composed of two parts:

1. Bypass branch/link register results to the Branch Unit
2. Bypass FU results from E1/E2 or E2/WB register

The first part allows to reduce the issue-use latency of Branch Unit operands to a single clock-cycle. These operands can be a branch condition, or the updated link register, or the stack-pointer. This is the minimal latency since the Branch Unit resides in the second stage. Hence, the functional results for the Branch Unit come from the E1/E2 pipeline latch. Part two benefits the first execution stage, as the data path can be fully bypassed from both execution stages.

3.9 Implementation

We implemented the forwarded and un-forwarded 4-issue processor instances using Xilinx ISE version 12.4 on a Xilinx Virtex-6 XC6VLX240T FPGA. We present here the measurements that were performed on this FPGA (1FFG1156) for the ML605 board. The 2-issue core has 2 ALUs, 2 MULs, 1 Branch Unit and one Load/Store Unit. The implementation results for the target FPGA are presented in Table 3.

4 Parameterization and Optimizations

In this section, we present the parameterization of the ρ-VEX processor and discuss several optimizations to reduce its reconfigurable resources.

Parameterization

Utilizing the current design of the ρ-VEX processor, we generated different ρ-VEX processor instances. The following parameters of our current design can be changed:

1. Issue-width of the processors pipeline
2. Type and the location of functional units
3. Size of the GP and BR register files
4. Size of the data and instruction memory
5. Presence of forwarding logic

We generated 2-, 4-, and 8-issue ρ-VEX processors. The 2-issue processor has 2 ALUs, 2 MUL, 1 Branch Unit, and 1 Load/Store Unit. The 4-issue processor has 4 ALUs, 2 MULs, 1 Branch Unit. The register files in all these designs have 64 registers.

Table 1 presents the resource utilization for the 2-, 4-, and 8-issue ρ-VEX processors for the target FPGA. The numbers in braces in the table represent the percentage of resource utilization for the entire FPGA. All designs can achieve a 150 MHz frequency.

Table 1. Resource utilization for ρ-VEX

Issue-width	Slice Regs	Slice LUTs
2-issue	2609 (0%)	10714 (7%)
4-issue	3055 (1%)	23253 (15%)
8-issue	3941 (1%)	50474 (33%)

Optimizations

The optimizations mainly include new register file designs for the ρ-VEX processor to reduce the number of configurable resources for different processor instances. When a processor's issue-width increases, the number of read and write ports on the multi-ported RF has to increase accordingly. Generally, multi-ported memories in FPGAs are implemented using slices. With the aim of supporting multiple ports, the BRAMs are organized into banks and data is duplicated across various within each bank. For a RF with 4-write and 8-read ports, we need 32 BRAMs distributed across 4 banks with 8 BRAMs per bank. In order

to provide multiple read ports, multiple BRAMs are used within each register bank to store duplicate copies of the corresponding register subset. Basically, the number of write ports defines the number of banks and the number of read ports defines the number of BRAMs per bank. Table 2 presents the resource usage for the 2-, 4-, and 8-issue ρ-VEX processors with the BRAMs-based register files for the target FPGA. The general-purpose register file contains 64 registers.

Table 2. Resource utilization for ρ-VEX processors

Register File	Slice Regs	Slice LUTs	BRAMS
2-issue	586 (0%)	6375 (4%)	4 (1%)
4-issue	1046 (0%)	12899 (8%)	16 (4%)
8-issue	1868 (0%)	26252 (17%)	64 (15%)

5 Experimental and Preliminary Results

We ran 6 application kernels on the ρ-VEX processor. All measurements were performed on a Xilinx Virtex-6 XC6VLX240T FPGA, *Speed-grade -1* on a ML605 board. We compared the forwarded and un-forwarded instances of the 4-issue ρ-VEX processor and measured the execution times of the benchmarks to investigate whether forwarding provides performance benefits. This is not trivial as the forwarding logic is situated on the critical path of the pipeline. Hence, the clock frequency can degrade. The processor runs at 160 MHz, and the forwarded processor runs at 150 MHz. From the measured execution times, we compute the speedups that are given in Figure 4. Forwarding is indeed beneficial when more performance is required as it provides speedup for every kernel. We observe that the highest speedup is achieved for ADPCM benchmark and that the lowest achieved speedup is for CJPEG. Therefore, the speedup lies between 1.32 and 2.93, despite the slight degradation in clock frequency.

Finally, we compared the speed-optimized MicroBlaze to the fastest instance of the ρ-VEX processor (4-issue with forwarding). The MicroBlaze is executing code with the -O3 optimization and -in-lining flags turned on for the GCC compiler. The version 3.42 of the VEX compiler was utilized and the compiler flags were set to -O3 and -auto-inline. The MicroBlaze runs at 150 MHz and 200 MHz, while our 4-issue forwarded ρ-VEX runs at 150 MHz. The speedup results for the 6 benchmarks are depicted in Figure 5. We observe that the 4-issue forwarded design provides speedup in the range of 1.04 to 2.72. Evidently, the speedup of our ρ-VEX processor is higher when compared to a MicroBlaze running at 150 MHz, ranging between 1.38 to 3.63.

Table 3 shows the resource requirements for the different configurations of the MicroBlaze and three ρ-VEX instances. The results show that the 2-issue ρ-VEX processor requires 1.05 the amount of Slice Register resources, 2.75 times

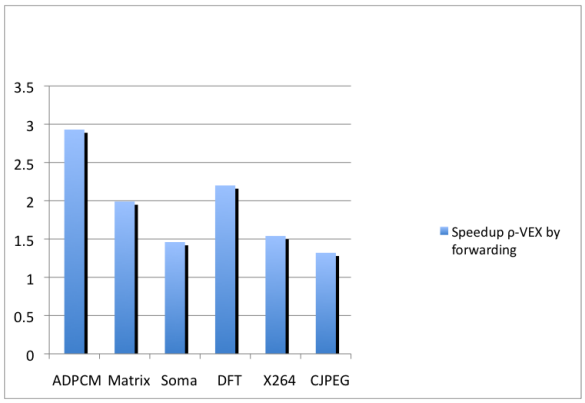

Fig. 4. Speedup of ρ-VEX processor due to forwarding logic

Table 3. Resource Utilization for MicroBlaze and ρ-VEX

Core	Slice REG	Slice LUT	DSP48E1	LUTRAM
2-issue	757	2684	0	352
4-issue	1367	6060	4	1408
fwd 4-issue	3274	19119	4	1408
MB speed	1041	1231	3	149
MB area	722	977	3	148

the amount of Slice LUTs, and 2.38 times the amount of LUTRAMs that the MicroBlaze requires. This core requires no DSP resources. The un-forwarded 4-issue ρ-VEX processor requires 1.31 the amount of Slice Register resources, 4.92 times the amount of Slice LUTs, and 9.45 the amount of LUTRAMs compared to the MicroBlaze. For the DSP resources the comparison factor is 1.33. The majority of the resources are consumed by the register files, which requires 2 write ports and 4 read ports for the 2-issue implementation, and 4 write ports and 8 read ports for the 4-issue implementation. These are larger as they exploit more ILP by avoiding excessive spills.

6 Conclusion and Future Work

We discussed the new design and implementation of the ρ-VEX processor (v2.0). By introducing pipelining and forwarding the performance is greatly improved over the initial multi-cycle ρ-VEX design. We also extended an existing tool chain to fully support our design. Furthermore, as this design targets an FPGA implementation, it requires optimizations of its register files. We observed that execution times decreases for each of the benchmarks when forwarding is selected. Therefore, forwarding can be selected when more performance is required. We observed that the speedup is in the range of 1.32 to 2.93 for the 6 benchmarks.

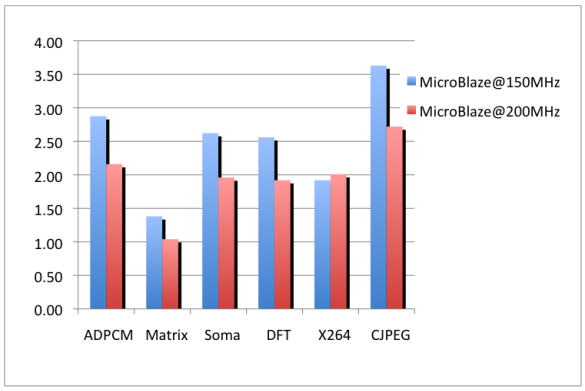

Fig. 5. Performance comparison with speed-optimized MicroBlaze

Finally, a performance comparison between the ρ-VEX processor clocked at 150 MHz and the MicroBlaze optimized for speed and clocked at 200 MHz was performed for the Xilinx Virtex-6 FPGA. The 4-issue forwarded ρ-VEX processor runs at 150 MHz and is faster than the speed-optimized MicroBlaze clocked at 200 MHz. For the 6 investigated benchmark kernels we measured a speedup in the ranging of 1.04 to 2.72.

Acknowledgment

This work is supported by the European Commission in the context of the ERA (Embedded Reconfigurable Architectures) collaborative project #249059 (FP7). The opinions expressed in this paper are of the authors only and in no way reflect the opinions of the European Commission

References

1. Kumar, A., Hansson, A., Huisken, Corporaal, H.: An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor System on Chip. In Proceedings of Design, Automation & Test in Europe Conference & Exhibition, DATE, pp. 1-6, (2007)
2. Lowney, P.G., Freudenberger, S.M., Karzes, T.J., Lichtenstein, W.D., Nix, R.P., O'Donell, J.S., Rutttenberg, J.C.: The Multiflow Trace Scheduling Compiler. Kluwer Academic Publishers, (1993)
3. Fisher, J., Faraboschi, P., Young, C.: Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools. Morgan Kaufmann (2004)
4. Ascia, G., Catania, V., Palesi, M., Patti, D.: EPIC-Explorer: A Parameterized VLIW-based Platform Framework for Design Space Exploration. University of Catania, In Proceedings of Design Automation Conference, DATE, vol. 2, pp. 940-943, (2005)

5. Faraboschi, P., Brown, G., Fisher, J.A., Desoli, G., Homewood, F.,: Lx: A Technology Platform for Customizable VLIW Embedded Processing. Hewlett Packard Laboratories, Cambridge MA, STmicroelectronics Cambridge MA., In Proceedings of the International Symposium on Computer Architecture, vol 27, pp. 203-213, (2000)
6. Seedorf, R.: Fingerprint Verification on the VEX Processor. Masters thesis Delft University of Technology (2010)
7. Wong, S., van As, T., Brown, G.: ρ-VEX: A Reconfigurable and Extensible Soft-core VLIW Processor. In Proceedings of International Conference on Field-Programmable Technology, FPT, pp. 369-372, (2008)