A Compression Instruction Set Design based on RISC-V for Network Packet Forwarding

Z Cao¹, Q Lv¹, Y Wang¹, M Wen¹, N Wu², C Zhang¹

¹ College of Computer, National University of Defense Technology, Changsha 410073, China
² NetForward Micro-Electronic Corporation, Chengdu 610031, China

Abstract. The multi-core processor of RISC for network packet forwarding has been limited by the on-chip storage space. As more and more cores are implemented in one chip, the storage resources allocated by each core on the chip become less and less, as well as the conflict of visiting RAM has become more prominent. Therefore, the use of more compact instruction size will reduce the number of visits, and get a higher instruction cache hit rate, thereby improving the performance of the application and energy efficiency. Based on the open source instruction set RISC-V, this paper proposes a compressed ISA for the network packet forwarding. It is proved by experiment that the new instruction set which customized by this method has higher compression efficiency and better system performance for the network packet forwarding applications.

1. Introduction
With the continuous development of network technology, the network speed of the current backbone generally can achieve to 40Gbps or more. At the same time, there is a very high demand for the speed on network packet forwarding of network equipment due to the rapid growth of network speed [1]. The current network equipment mainly uses ASIC and network processor. However, since the high cost and long design period by using ASIC, it has been used in the network equipment less and less. On the contrary, most of the network equipment manufacturers choose the network processor, because of its advantages of low cost and high flexibility. In order to improve the processing ability of network equipment, network manufacturers have to use the multi-core architecture [2,3,4], such as Alcatel-Lucent FP series [5], Marvell’s Xelerated series [6], EZchip NP Series [7], and so on. Until now, the number of cores can be reached to dozens, or even hundreds.

By implementing the network processor of the multi-core architecture, the processing capacity for packet forwarding has been improved. Meanwhile, it also brought some problems, for example, higher cost, less storage space for each core, higher energy dissipation, and other issues.

- **Cost problem:** Firstly, even if we put aside the complex and bad historical design problem of the existing mature business ISA (Instruction Set Architecture), we also need to consider the high usage fees and strict use restrictions. Secondly, some open source instruction sets, such as OpenRISC, will be slowly developed and lack of maintenance.

- **On-chip storage space problem:** the number of PEs (Processing Element, PE) in one chip increases rapidly, but the manufacturing process improves slowly, therefore, each PE gets less storage space, such as the sizes of instruction cache and data cache will be drastically reduced.
At the same time, the cache hit rate reduces, visit conflict increases and the system performance reduces as well.

- **Energy dissipation problem**: For most specific processor cores, launching the instruction stream and cache misses are two major reasons for energy dissipation [8]. The paper [9] pointed out that in an ordinary five-stage pipeline, the access and instruction cache misses may cause 40% of the energy dissipation, however main memory accesses and processor stalls will consume more energy than cache misses.

To solve the problems above, the design method of the ASIP (Application Specific Instruction set Processor) may be a good choice: customizing the hardware with the existing instruction set, such as FPGA or other programmable hardware, help separate the design from hardware implementation. With customization, it is easier to optimize the design of hardware and to improve the processing performance in network packet forwarding. In order to implement the customization, this paper proposes the following solutions:

- The open source instruction set RISC-V has obvious advantages: Firstly, the RISC-V instruction set allows anyone to use and modify it for free; Secondly, unlike the instruction set which has long history, there is no historical and compatible issues for RISC-V; Thirdly, modular design also makes it easy to add instructions and modify; Last but not the least, RISC-V is designed for the modern computing devices, such as cloud computing, data centers, network equipment and embedded devices, light quantitative and modular design bring us a simpler and controllable hardware implementation. Besides, the RISC-V contains extension sets of I (Integer), C (Compression), M (integer Multiple / Division), A (Atomic), F (single-float precision) and D (double-float precision). The I set is a complete collection, and the others are the extension sets. The C set is a superset of the I set to increase instruction stream density.
- Compressing instruction set can increase the instruction stream density, and thus it can reduce the number of fetching instructions, improve the instruction cache hit rate, and even the number of RAM access. This equates to increase the size of instruction cache, improve processing performance and reduce the energy dissipation.

Based on the RISC-V instruction set, this paper attempts to find out the frequency of each instruction by analyzing the applications in network packet forwarding domain, and then replace the instructions which have the highest frequency with some short word instructions, so the instruction density is increased, and thus to solve the problems which are caused by the multi-core architecture in network processor.

The contributions of this paper are summarized as follows:

- Test and analyze the applications in the network packet forwarding, and find out the common characteristics, and then provide the basis for customizing the instruction set in this field;
- Custom the RISC-V instruction set by cutting, adding and changing some instructions, so that high density instruction steam can be provided;
- Modify the compiler so as to support running and testing the customized instruction sets.
- The rest of this paper is organized as follows: section 2 describes the work of the compression instruction set. The third section describes how to design the compression instruction set for the network packet forwarding field. The fourth section describes the experiment. And the fifth section is the summary and future work.

2. Background
This section is going to introduce compressed ISA related work, such as the comparison between CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer), and then two existing compression methods and their typical cases. For the initial computer architecture, smaller storage space and expensive storage devices severely limit the size of the application instruction stream, so the CISC was originally designed to use the variable-length instructions: according to the frequency of various instructions occurred in different
applications, the higher the frequency, the shorter the code is used. At the same time, according to its operands or the different ways of addressing mode to shorten its encoding length, such as DEC VAX [10] and Intel X86 instruction set.

For the instruction sets that follow the RISC principle [11], they have always been to choose performance rather than code size. An unavoidable problem with loose instruction encoding is that the density of the code is smaller. Compared to the CISC instruction set, the program that uses the RISC instruction set will require more instructions to complete the same function. With the extensive use in embedded systems with the RISC instruction set, as well as the development of multi-core architecture, the demand for improving the code density is becoming increasingly important. To meet the demands, some RISC vendors have extended their instruction sets to support a subset of their base ISA’s functionality by using short instruction words, typically 16 bits. In general, there exist two compression methods: Statistical-based Compression and Dictionary-based Compression [12,13].

2.1. Statistical-based Compression
Statistical-based compression uses the frequency of singleton characters to choose the size of the codewords that will replace them. Frequent characters are encoded using shorter codewords so that the overall length of the compressed text is minimized. Huffman encoding of text is a well-known example. These short instructions have a straightforward mapping to one or more instructions in the base ISA. MIPS16 [14] and ARM Thumb [15] are notable examples of this approach. Thumb is actually a separate instruction set added to the standard ARM RISC instruction set, which consists of approximately 36 16-bit instructions, including basic additions, subtractions, loops, and jump instructions. The MIPS16e instruction set includes a set of 16-bit standard MIPS instruction sets as well. Thumb and standard ARM instruction sets, as well as the MIPS16e and standard MIPS instruction sets can not be mixed. It must be explicitly switched between the compression instruction and the standard instruction via a mode switch instruction. Switching not only increases the complexity of the hardware, generates additional switching code, but also reduces the system performance by switching them. By using the Thumb and MIPS16e instructions to replace the standard ARM and MIPS 32-bit instructions, you can reduce the size of most program code by about 20% to 30%.

ARM has targeted improvements and proposed Thumb-2 compression instruction set [16]. Unlike Thumb, Thumb-2 is a complete set of compression instructions that can completely replace the ARM standard instruction set, running 16-bit and 32-bit mixed code without switching. But this also increases the implementation of the hardware. In general, Thumb-2 provides less code compression efficiency, but less performance loss.

2.2. Dictionary-based Compression
Dictionary-based compression selects entire phrases of common characters and replaces them with a single codeword. The codeword is used as an index into the dictionary entry which contains the original characters. Compression is achieved because the codewords use fewer bits than the characters they replace. For example, replace frequent instructions or instruction sequences with an index into a dictionary, which stores the decompressed instructions. Though these schemes are effective at reducing code size, the dictionary lookup adds latency and offsets the energy reduction from compression.

IBM's CodePack [17] system uses the most complex code compression techniques. Unlike Thumb and MIPS16e, the CodePack system completely compresses the code as if it were running WinZip in PowerPC. CodePack parses and compresses the entire program, and the generated user code must be decompressed and executed in a compressed version. Despite its complexity, CodePack offers 20% to 30% of space savings, similar to other compression technologies. However, CodePack has some other effects. Since each compressed program has its own compact key, CodePack is essentially both a compression system and an encryption system. This also means that the compressed PowerPC
program is not binary code compatible. In addition, CodePack generates two key values for each program because the upper and lower 16 bits of the instruction are compressed separately. The variable-length instruction set has been used to improve code density for a long time, but since the original RISC ISA does not have enough opcode space for these unplanned compression instructions, they have been treated as a new ISA Development. RISC-V is designed to support compression instructions from the outside, leaving enough opcode space for RVC (RISC-V Compressed), so that it can be added as a simple extension along the basic ISA (along with other standard extensions). RISC-V compression instruction set is based on the statistical-based compression method. The idea of RVC is to reduce code size for embedded applications, improve performance and energy efficiency for all applications.

3. Customized Compression Instruction Set Design

3.1. Minimizing the Instruction Set

For the specific domain of network packet forwarding, the usual operation is to deal with the packet header, and the processing of each arrival packet is basically the same. In order to compress the scale of the application code in network packet forwarding, the first thing is to verify the distribution of different instructions in this field.

Unlike desktop applications and multimedia applications, there is no a standard unified benchmarking program to evaluate the performance of the network. Some of the popular test sets, such as Commbench [18], NpBench [19], NetBench [20], and EEMBC's network 2.0 [21], have different considerations and designs, and the evaluation conditions are not completely the same. Considering the specific field of network message forwarding in this paper, it refers to several above mentioned typical benchmarks, and extract some typical cases of network message forwarding as a test assembly in this experiment. The test assembly run independently on the target machine in the event of detachment from the operating system. This paper chooses four common applications in network packet forwarding field, including DRR, FRAG, RTR, and CRC. Details are shown in Table 1.

| Application | Content |
|-------------|---------|
| DRR         | QoS queue scheduling algorithm, by controlling the use of different types of packets to link bandwidth, so that different data streams get different levels of service |
| FRAG        | IP packet fragmentation technology |
| RTR         | Routing Table finding algorithm in three-layer Forwarding |
| CRC         | The cyclic redundancy check algorithm. calculating the checksum in the link layer |

Compile these programs with a complete RISC-V instruction set (G set, which contains the IMAFD sets), and then obtain the distribution of various instruction types. The distribution of the various types of instructions is shown in Table 2.

| Applications | I Set | F/D Sets | M Set | A Set |
|--------------|-------|----------|-------|-------|
| DRR          | 99.61%| 0.18%    | 0.21% | 0%    |
| FRAG         | 99.61%| 0.17%    | 0.22% | 0%    |
| RTR          | 98.85%| 0.97%    | 0.18% | 0%    |
| CRC          | 99.62%| 0.17%    | 0.21% | 0%    |

From the analysis results in Table 2, it shows that the application for network packet forwarding uses very few RISC-V extended instructions. Therefore, when designing ASIP for network packet forwarding, it is possible to consider replacing hard floating point element with soft floating point element, and so on, so as to achieve the purpose of minimizing the instruction set, and reducing the complexity of the hardware structure, as well as reducing development costs and improving the system performance.
3.2. Customize the compression Instruction Set

In RISC-V, RVC (RISC-V Compressed set) is a variable-length instruction set extension, and it is a superset of the RISC-V ISA, encoding the most frequent instructions in half the size of a RISCV instruction; the remaining functionality is still accessible with full-length instructions. RVC programs are 25% smaller than RISC-V programs, fetch 25% fewer instruction bits than RISC-V programs, and incur fewer instruction cache misses [9]. The 32-bit RVC instructions are shown in table 3.

| Inst[15:13] | Inst[1:0] |
|-------------|-----------|
| 00 | ADD4SPN | FLD | LW | FLW | Reserve | FSD | SW | FSW |
| 01 | ADDI | JAL | LI | LUI/ADD16SP | MISC-ALU | J | BEQZ | BNEZ |
| 10 | SLLI | FLDSP | LWSP | FLWSP | J[AL]R/MV/ADD | FSDSP | SWSP | FSWSP |
| 11 | >16 bits (Standard Instructions) |

According to the design in Section 3.1, even if the RISC-V extended instruction set has been removed and only the I collection has been retained, the I collection also contains a large number of instructions. Since the application for network packet forwarding has a relatively simple application of the desktop system Function, the use of the frequency of the instructions must be obviously different. In this section, it re-customs RVC based on the I instruction set, so that it has a higher compression ratio. After compiling the four applications (DRR, CRC, RTR and FRAG) in the test set according to the RV32I standard, the frequency distribution of each instruction and the number of the registers on this basis can be obtained. The results are shown in Figure 1 and Figure 2.

![Figure 1. The top 24 instructions in the highest frequency](image-url)
It can be seen from Figure 1 that the high frequency standard instructions used for network packet forwarding applications do not match the instructions defined by the RVC instruction set. In other words, if the application for network packets is forwarded only in accordance with the instructions defined by RVC Compression, the compression ratio would not be high. Due to the coding format, RVC instruction code number is less than RISC-V standard instruction set itself. How to effectively select the standard instruction to compress is worthy of studying. For one thing, adding compression instructions which are not defined in RVC but frequently used would bring a better performance in the specific application situation. For another, the compression instructions in original RVC which don’t provide good compression effect can be deleted or modified so that the final ISA could be more effective.

The following is re-compiling the RV32I based on the results of the distribution, and the design criteria are based on the following principles and facts:

- The repeatability of the registers in the instruction. This repetition is particularly noticeable in I-type instructions and R-type instructions for 3 operands.
- The locality of the register. Compression instructions because the instruction length is limited, so the register can also be limited, usually only 8. As can be seen in Figure 2, the use of registers is localized.
- Most of the immediate or address offsets are small.
- The use of special registers in registers, such as zero registers, ABI link registers, or ABI stack registers

According to the instruction usage frequency in Figure 1 and the register usage ratio shown in Figure 2, the list of RVC instructions to be customized is shown in Table 4.

**Table 4. Customized RVC.**

| Inst[15:13] | Inst[1:0] | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|-------------|-----------|-----|-----|-----|-----|-----|-----|-----|-----|
| 00          | ADDI4SPN  | LBU | LW  | SH  | Reserve | SB  | SW  | ANDI |
| 01          | ADDI      | JAL | LI  | LUI/ADDI16SP | MISC-ALU | J   | BEQZ | BNEZ |
| 10          | SLLI      | LBUSP | LWSP | SHSP | J[AL]R/MV/ADD | SBSP | SWSP | Reserve |
In Table 4, the bold texts represent the new compression instructions, while the underlined texts are instructions for the original instructions to be modified, and the rest are the original RV32C instructions.

3.3. Compiler Modification
Since the ISA has been modified, the compiler need to be migrated correspondingly to translate the high-level programs to target machine codes, and then the CPU can implement the functions. The process of machine code generation is shown in figure 3.

![Figure 3. The process of machine code generation](image)

Figure 3. The process of machine code generation

The GCC compiler translates the source code into an intermediate representation, making it easy for the compiler to optimize. After optimization, the compiler generates the corresponding assembly code based on the assembly instructions in the target platform machine description file and the GCC intermediate representation template. And then use the assembler to convert the assembly code into machine code.

With ISA modified, riscv.md, the machine description file, needs to be changed as well. A typical method is adding the compression instruction template, which allows the compiler to generate compression instructions. On the other hand, the assembler should be changed so that it can recognize the compression instructions and generate the corresponding machine code.

4. Experiment and Result
In order to obtain the real compression effect, this paper designs a complete hardware and software experimental environment. It acquires the relevant data, and evaluate the performance after applying the new compression instruction set through simulating the real network environment for message forwarding.

4.1. Hardware Experimental Environment
This paper adopts the high-performance hardware FPGA board and the single core supporting RVC, RVC-N and RVI. The specific configuration is as follows:

- Xilinx official board VC709: There is a high performance XC7VX690T Virtex-7 FPGA chip onboard, as well as SFP+ and UART interface. SFP+ converter, which achieves the conversion of the optical port and the electric port, is also used in the experiment.
The CPU core adopts the RISC-V SOC system written by Sergey Khabarov [22], and it is modified according to actual requirements. Since the original design is 64-bit architecture and does not support RVC, it is necessary to change it to 32-bit architecture, and provide support for RVIC or RVIC-N. The CPU is designed to support unaligned instruction storage.

4.2. Software Experimental Environment
The software experiment environment includes test assembly, PC test program, compiler, and so on.

- Network Test Assemblies: This paper chooses four common applications in network packet forwarding field, including DRR, FRAG, RTR, and CRC.
- Test software for sending and receiving: This experiment adopts network test software, which is provided by NetForward Micro-Electronic Corporation, to carry out sending and receiving test for the target and to simulate the normal network communication data.
- Compiler: For comparison, the RVIC-based test programs and RVIC-N-based test programs need be compiled. Therefore, compilers of different functions need to be configured.

4.3. Experimental Design and Results
This paper analyzes the differences between RVC-N and RVC instructions in the application of network message forwarding, from the aspects of static code compression rate, dynamic code compression rate (energy consumption), and performance and instruction failure rate.

4.3.1. Static Code Compression Ratio. Static code size impacts instruction memory costs and start-up overhead, and it correlates with instruction cache miss rates [9]. One of the goals in defining the RVC-N is thus to reduce static code size in network domain. In [9], the standard RVC and RISC-V, MIPS, ARM, MIPS16, Thumb, Thumb-2 and X86 were compared based on SPEC CPU2006 benchmark suit. The results show that static compression ratio of RVC ranks second only after Thumb and Thumb-2. In this paper, only the standard RVC and RVC-N are compared based on the network test set to determine that whether RVC-N in the field of network packet forwarding has a better effect.

![Figure 4. Comparison of static code compression ratio](image_url)
As shown in Figure 4, the code size of four test programs has been significantly reduced through the use of RVIC-N or RVIC code compression. The comparison results in Figure 5 also show that RVIC-N has a higher compression efficiency than RVIC.

4.3.2. Dynamic Code Compression and Energy Consumption. In an idealized processor model, dynamic code size equals to the amount of data fetched from instruction memory. Reducing dynamic code size can thus reduce instruction fetch energy, as fewer total bits are retrieved from instruction memory. There is a standard task in this experiment, and count the amounts of fetching instructions by using different compression instruction sets when the task is finished. By comparing the amounts, the result illustrate which compression instruction set has better performance. The comparison result is shown in Figure 6.

From figure 6, there is no big error between the dynamic code sizes, which is out of the expectation. By analyzing the design, it may be caused by the storage mode: the 32-bit instruction is stored on any 16-bit boundary.

4.3.3. Instruction Cache Missing Rate and Performance. Compression instructions usually will result in a smaller instruction cache failure rate. Because the instruction length is reduced, a Cache Line can save more instructions, or it means that more instructions can be contained in the same size instruction cache. Compression instructions can improve the hit rate. Meanwhile, it can improve system performance through reducing the number of CPU pauses for waiting for instructions. In order to find the role and performance of RVC in instruction cache failure rate, the instruction caches of different sizes is set and the mapping method is direct mapping mode. The instruction missing ratios of standard RVC and RVC-N are tested individually in the dynamic operation process.

Table 5. Missing ratios with different cache sizes

| Cache | CRC | DRR | FRAG | RTR |
|-------|-----|-----|------|-----|

Figure 5. Static compression ratio

Figure 6. Dynamic Code Size Comparison
| Size(Bytes) | RVIC-N | RVIC | RVIC-N | RVIC | RVIC-N | RVIC | RVIC-N | RVIC |
|------------|--------|------|--------|------|--------|------|--------|------|
| 256        | 0.22%  | 5.27%| 2.32%  | 4.59%| 3.94%  | 3.76%| 4.33%  | 9.45%|
| 512        | 0.21%  | 1.25%| 1.32%  | 4.58%| 2.52%  | 0.50%| 3.81%  | 5.47%|
| 1024       | 0.19%  | 0.23%| 0.30%  | 4.57%| 1.99%  | 0.22%| 1.53%  | 5.06%|
| 2048       | 0.13%  | 0.18%| 0.09%  | 0.05%| 0.17%  | 0.17%| 1.03%  | 2.30%|
| 4096       | 0.08%  | 0.11%| 0.06%  | 0.03%| 0.11%  | 0.08%| 0.56%  | 1.50%|
| 8192       | 0.05%  | 0.04%| 0.02%  | 0.01%| 0.04%  | 0.06%| 0.19%  | 0.92%|
| 16384      | 0.02%  | 0.02%| 0.01%  | 0.00%| 0.03%  | 0.01%| 0.03%  | 0.77%|

5. Conclusions
This paper proposes an approach to custom RISC-V compression instruction set, and presents a custom instruction set for network packet forwarding. By analyzing the applications for network packet forwarding, the frequently used RISC-V instructions can be found out. Based on the results, unnecessary RISC-V extended instruction sets are cut, so as to reduce the complexity of hardware design and improve the system performance. Besides, the frequently used instructions are added to the compression instruction set, and the low frequently used instructions are removed from the original compression instruction set. From the experimental results, the result shows that the modified RISC-V compression instruction has higher compression efficiency in the field of network packet forwarding. What’s more, it also can improve the instruction cache hit rate and system performance, and reduce the energy dissipation. This approach which customized the compression instruction set for network packet forwarding can also be used for other fields.

The current work of this paper is still relatively shallow. It just plays a fundamental role for the compression instruction set customization in the network packet forwarding field. In order to further enhance the effectiveness and expand its practicality, the future work is as follows: First, this article only uses four basic network applications as benchmarks to evaluate the distributions of instructions, which is lack of analysis of more comprehensive network applications. The following work is carrying out subsequent analysis of more network application testing procedures. Second, in the test of instruction cache hit rate, considering more different cache size of the configuration options and selecting a different mapping mode to implement the instruction cache will provide the basis for selection of parameters for the multi-core architecture. Last, the CPU core used in this paper is non-commercial mature product, and the ROM and RAM it used are implemented by on-chip resources, which differs from the real SOC systems. The next step is considering the introduction of off-chip ROM and RAM, such as FLASH and DDR chips.

6. Acknowledgments
The authors gratefully acknowledge supports from National Key Research and Development program under No.2016YFB1000401 and No.2016YFB1000403; National Nature Science Foundation of China under NSFC No.61502509 and 61402504. This work was also supported by NetForward Micro-Electronic Corporation.

7. References
[1] Zhang X, Sun Z and Zhang M 2006 An Efficient Packet Scheduler for Modern Network Processors: Guarantee Load Balancing and Packet Ordering IEEE 2006 Workshop on High Performance Switching and Routing Ponza Poland June 7-9
[2] Peyravian M, Calvignac J, Sabhikhi R 2003 IBM—PowerNP Network Processor http://www.ibm.com
[3] Intel IXP1200 Network Processor Family - Hardware Reference Manual 2001 http://www.intel.com
[4] Cisco ASR 1000 Series Aggregation Services Routers http://www.cisco.com
[5] FP3: Breakthrough 400G network processor http://www.alcatel-lucent.com/products/fp3
[6] Marvell Introduces New Xelerated Network Processors and Traffic Management Solutions for the Mobile Internet http://www.marvell.com/

[7] EZchip NP-5 Product Brief http://www.ezchip.com

[8] Katevenis M, Sherburne R, Patterson D, and Sequin C 1984 The RISC II Micro-Architecture Journal of VLSI & Computer Systems vol1, p138–152

[9] Waterman A 2011 Improving Energy Efficiency and Reducing Code Size with RISC-V Compressed Technical Report No. UCB/EECS-2011-63

[10] Strecker W 1978 VAX-11/780—A Virtual Address Extension to the DEC PDP-11 Family In AFIPS Spring Conference

[11] Patterson D, Sequin C 1981 RISC I: A Reduced Instruction Set VLSI Computer In Proceedings of the 8th Annual International Symposium on Computer Architecture (ISCA ’81 IEEE Computer Society Press), pp443-457

[12] Lefurgy C, Bird P, Chen I, and Mudge T 1997 Improving Code Density Using Compression Techniques. In Proceedings of the 30th Annual International Symposium on Microarchitecture p 194–203

[13] Li T, Wu J, Lam S K, Srikant R and Lu X 2010 Selecting Profitable Custom Instructions for Reconfigurable Processors J. Syst. Architect. Special Issue on Hw/Sw Co-Design 56 p 340-351

[14] MIPS 2010 Technologies Inc MIPS64 Architecture for Programmers Volume IV-a: The MIPS16e Application-Specific Extension to the MIPS64 Architecture http://www.mips.com/products/architectures/mips64/

[15] ARM Ltd. 2008 ARMv6-M Architecture Reference Manual http://infocenter.arm.com/help/topic/com.arm.doc.ddi0419c/

[16] Phelan R 2003 Improving ARM Code Density and Performance ARM Limited

[17] Lefurgy C, Piccininni E and Mudge T 1999 Evaluation of A High Performance Code Compression Method International Symposium on Microarchitecture (Micro-32 Proceedings IEEE Xplore) p 93-102

[18] Wolf T and Franklin M 2000 CommBench - A Telecommunications Benchmark for Network Processors IEEE International Symposium on PERFORMANCE Analysis of Systems and Software (IEEE), pp154-162

[19] Lee B K and John L K 2003 NpBench: A Benchmark Suite for Control Plane and Data Plane Applications for Network Processors International Conference on Computer Design Proceedings. (IEEE), pp226-233

[20] Mangione-Smith B, Memik G and Mangione-Smith W H NetBench: A Benchmarking Suite for Network Proc. Int’l Conf. Computer Aided Design, pp39-43

[21] Dinhtrong T T, Bieman J M 2005 The FreeBSD Project: A Replication Case Study of Open Source Development IEEE Transactions on Software Engineering vol.31 (6), pp481-494

[22] https://github.com/sergeykhbr/riscv_vlog