A Catalog and In-Hardware Evaluation of Open-Source Drop-In Compatible RISC-V Softcore Processors

Carsten Heinz*, Yannick Lavan†, Jaco Hofmann‡, Andreas Koch§
Embedded Systems and Applications Group
TU Darmstadt
Darmstadt, Germany
{*heinz, †lavan, ‡hofmann, §koch}@esa.tu-darmstadt.de

***Abstract*—With the increasing popularity of RISC-V in the academic and industrial world, an ever growing number of open-source implementations of the instruction set have become available. However, it is not an easy task to compare the cores to one another, as they employ different interconnects, build systems and so on.**

This work presents an open-source catalog of RISC-V cores for use on FPGAs. All of these cores have been wrapped as drop-in compatible processing elements and can be used either standalone, or integrated into the TaPaSCo SoC composition framework. By using TaPaSCo, details of the bitstream generation flow and user-space interfaces are abstracted away, allowing the user to focus on the needs of the concrete applications when exploring the RISC-V landscape.

All of the catalog's cores have been synthesized for a number of hardware platforms, and are evaluated against each other using state-of-the-art embedded processor benchmarks such as Dhrystone, Embench and CoreMark. The results show that the cores have a huge degree in performance variability. The slowest cores achieve less than 100 MHz on large UltraScale+ devices, while better FPGA-optimized cores run in excess of 500 MHz. Accordingly, the benchmarks show a wide spread of performance ranging from less than 0.5 CoreMark/MHz up to over 2.5 CoreMark/MHz.

***Index Terms*—RISC-V, FPGA, soft-core**

I. Introduction

The rising popularity of RISC-V in the industrial and academic space resulted in a plethora of open-source RISC-V implementations. While all of these cores should follow their respective RISC-V ISA version, all features of the cores not specified by the ISA are left open to the respective designers. Accordingly, a wide range of different interfaces and configurations is used to add peripherals to the core, access memories or control execution. One processor might use separate AXI interfaces to access instruction or data memories, while another core uses a single Wishbone bus to access both.

This heterogeneity results in practical problems when using and exploring the core landscape. Someone interested in using an open-source RISC-V core in their designs will find it difficult to choose the right processor for their specific use-case. The cores come with a variety of supported toolflows, and will almost certainly not work out-of-the-box with the platforms available to the user. Often, the cores come with only one example project for a single FPGA platform that might not always be a suitable target. Porting the cores to a better suited target platform is tedious and error prone work, which is not feasible to perform for all designs that might be interesting.

This paper contributes the following to solve some of these problems:

1) A generalized catalog of open-source softcore processors for easy drop-in use on Xilinx FPGAs.
2) Integration of the softcores into the accelerator framework TaPaSCo [1] for one-click bitstream generation with a given core.
3) The scripts to package any additional cores in a similar manner.
4) A thorough evaluation of all presented cores with regards to FPGA resource usage and performance over a range of state-of-the-art microprocessor benchmarks.

In Section II, RISC-V is briefly introduced and the a variety of available open-source RISC-V cores are presented. Afterwards, the cores suitable for packaging are chosen. The packaging of the cores and the integration into TaPaSCo is detailed in Section III. Using the packaged cores, evaluation results for a variety of FPGA based platforms are presented in Section IV.

II. Overview of RISC-V and Soft Processors

Introduced in 2010, the RISC-V instruction set architectures (ISA) has found increasing support in the industrial as well as academic worlds [2]. Developed at UC Berkeley, the architecture presents an interesting target for new developments and innovation in the processor design world. Compared to other architectures, RISC-V provides a number of distinct advantages which makes it a prime target for development: (1) The open source nature allows any interested party to develop their own chips based on the ISA without the prohibitive license costs incurred by competing ISA. (2) While the development process of the ISA is open source, the major parts of the ISA are already frozen making it an interesting target for

978-1-7281-1957-1/19/$31.00 ©2019 IEEE

software developers as well. (3) Additional functionality is available through a set of extensions which are also frozen after stabilization. (4) The modularity makes the ISA suitable for high performance as well as low-power chips. Specialized application processors with dedicated accelerators are also supported.

These advantages led to widespread community adoption and a wide range of supporters, organized in the RISC-V foundation. The toolflow based on GCC and LLVM is reaching maturity and complex software, such as various Linux distributions, is already available. Accordingly major players in the industry as well as innovative start-ups participate in the ecosystem. The hard disk drive and flash manufacturer Western Digital is actively developing high-volume Flash controllers based on the RISC-V architecture [3]. To increase adoption of RISC-V, they furthermore released their SweRV design as open-source under the Apache 2.0 license. The fabless semiconductor company SiFive is a new company that focuses on developing cores based on the RISC-V ISA adapted to a specific customers needs and ranging from small in-order processors up to large, multi-core, high performance, out-of-order designs [4]. Outside of the industrial domain, many projects fueled by academia or even hobbyists emerged. The creators of RISC-V at UC Berkeley themselves developed a number of core designs such as the BOOM processor [5]. Another interesting project is GRVI Phalanx which implements 1680 GRVI RISC-V cores onto a single Xilinx Virtex UltraScale+ VU9P FPGA [6]. Many more cores exist and are under active development [7].

As the targets of the various development projects span a wide range, not all cores are suitable for discussion in context of this paper. This paper is highly focused around *open-source FPGA-optimized* cores that are suitable for integration with accelerators. Thus, five main points have to be fulfilled by the cores: (1) Be compatible with the TaPaSCo Processing Element (PE) scheme for interfacing accelerators. (2) Be available to researchers and preferably open-source. (3) Be suitable for implementation on FPGA targets. (4) Support recent ISA versions. (5) Be 32 Bit ISA compatible.

The first requirement is highly important as the existing TaPaSCo framework is used for bitstream generation of the complete system-on-chip. Fortunately, TaPaSCo imposed only very few rules on the processing elements it will integrate. The main requirement is that the core uses AXI Memory-Mapped interfaces for its memory accesses. Again, most of the cores available already fulfill this requirement. Furthermore, most of the cores fulfill the second requirement as well. This excludes some proprietary commercial offerings, however, such as the SiFive family of processors. The third requirement should be fulfilled by most cores that fulfill the first requirement. However, some aspects of ASIC-targeted designs, such as memory generator usage, might make them infeasible for efficient FPGA implementation. As to the fourth requirement, the cores have to support recent versions of the ISA so they can be targeted by the latest versions of the software toolchain. Lastly, we restrict the evaluation to cores with 32 Bit ISA for better comparability (even though a small-number of 64b capable open-sources cores exist).

Following these selection criteria, these seven cores have been considered for integration and evaluation in this study: Taiga [8], VectorBlox Orca [9], Western Digital Swerv [10], Bluespec Inc. Piccolo [11], Bluespec Inc. Flute [12], SpinalHDL VexRiscv [13] and PicoRV32 [14]. Details of the specific cores can be found in Table I. All of the cores support a current version of the ISA and are available under various open licenses.

Some cores fulfilling our criteria were not considered for evaluation due to a number of reasons. Some of the more popular cores, e.g., Ariane [15] and SHAKTI E-Class [16], require a complex build infrastructure, or have too-limited external interfaces, making it hard to package them into the TaPaSCo infrastructure. However, the selected cores already cover a wide range and provide a good insight into the diverse landscape of RISC-V processors.

A. Description of the Selected Cores

a) Piccolo and Flute: The two cores are developed by Bluespec, Inc. and are coded in Bluespec System Verilog. Several configuration options are available to enable a range of instruction set extensions, and it is possible to switch between 32 and 64 bit as seen in Table I. Piccolo is targeted for embedded systems and thus has a short pipeline of three stages. Flute is designed for higher performance and uses a five stage pipeline.
b) Orca: A RISC-V core from VectorBlox Computing Inc. The pipeline has 5 stages, but can be reduced to 4. It can be used in conjunction with a proprietary vector unit for higher performance.
c) PicoRV32: A size-optimized RISC-V core from Clifford Wolf. Due to its small size, high frequencies can be achieved. The standard feature set is minimalistic, but can be extended through configuration options.
d) SweRV: Western Digital's RISC-V core for integration into embedded systems. It features the most complex pipeline in our selection, having 9 stages and a dual-issue execute stage. The configuration options are restricted to the memory.
e) Taiga: The core from Simon Fraser University, Canada is designed primarily for FPGA usage. The pipeline features parallel execution units for an improved IPC. It provides TLBs and MMUs to support operating systems such as a full-scale Linux.
f) VexRiscv: The core written in SpinalHDL is highly flexible. For configuration, a plugin system is used. It can be used as a small core for embedded systems, but also configured up to a Linux running system with MMU support.

B. Configuration Parameters

Many RISC-V cores come with a wide variety of configuration options. For our evaluation, we used the same configuration of a core throughout all benchmarks. For better comparability of the compute performance, all cores are configured without

TABLE I
PROPERTIES OF SELECTED RISC-V CORES.

Core	License	Language	ISA	Pipeline Structure
Flute	Apache 2.0	Bluespec SystemVerilog	RV32/64I[A][C][D][F][MSU]	5-stage in-order
Orca	BSD	VHDL	RV32I[M]	4 / 5 stages
Piccolo	Apache 2.0	Bluespec SystemVerilog	RV32/64I[A][C][D][F][MSU]	3-stage in-order
PicoRV32	ISC	Verilog	RV32I[M][C]	1 stage
SweRV	Apache 2.0	SystemVerilog	RV32IMCZ	9-stage dual-issue
Taiga	Apache 2.0	SystemVerilog	RV32IMA	variable
VexRiscv	MIT	SpinalHDL	RV32I[M][C][A]	2 to 5+ stages
Proprietary	Proprietary	unknown	Proprietary 32 Bit RISC core optimized for target FPGA	3, 5, or 8-stage

caches. This provides a fair comparison between all cores including the cores without support for caches. As the further configuration process depends on the core, we present them individually for each core.

a) Piccolo and Flute: Both are based on the same build system and thus have the same options. We used the configuration `RV32ACIMU` without support for floating point and privilege levels. Caches were disabled manually.

b) Orca: The pipeline has the default length of 5 stages and caches are kept disabled.

c) PicoRV32: Configuration with barrel shifter, fast multiplier and divider.

d) SweRV: Default configuration without data and instruction caches. Version 1.1 of the core is used.

e) Taiga: Configuration with multiplier, divider and AXI bus, disabled data and instruction caches.

f) VexRiscv: Custom configuration with AXI4, multiplier and divider, without data and instruction caches.

III. SYSTEM-ON-CHIP INTEGRATION

TaPaSCo is a System-on-Chip generator for FPGAs developed at TU Darmstadt and available under LGPL [1]. Using TaPaSCo a designer of accelerators can easily generate bitstreams for a variety of platforms without having to deal with the low-level platform integration work, dealing with PCIe interfaces or off-chip-memory controllers. Additionally, TaPaSCo can automatically find the throughput-optimal clock frequency for a given design, or even optimal performance configurations based on user-supplied metrics. Apart from the bitstream generation aspects, TaPaSCo helps with the software integration as well by providing an API and Linux kernel drivers for the supported platforms. The driver and API are generalized and can be used without user code changes on any of the platforms.

These aspects make TaPaSCo very suitable to host the exploration performed in this work, as all cores will operate in the same environment. Getting the required evaluation results is as easy as packaging the cores in a suitable format, and expending cluster compute time at synthesis and place & route. The basic computation elements of TaPaSCo are called processing elements (PE) and have a so-called T-Shape of three groups of ports: Usually a PE has an interface for *control* that is attached to a host (e.g., x86 or ARM CPU), an interface to the *off-chip memory* provided by the selected FPGA, and some additional *signalling* such as interrupts to indicate task completion. As the RISC-V cores described in Section II-A do not fit this shape directly, they are packaged with lightweight wrappers to conform to the interface conventions.

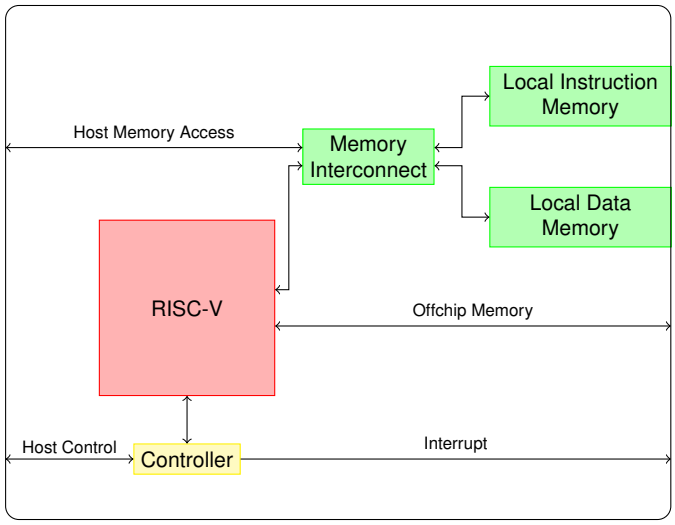

Fig. 1. Basic design of a RISC-V Core packaged as a TaPaSCo Processing Element (PE). Execution is managed by the Controller. The program is loaded into the local instruction memory which is mapped into the memory space of the RISC-V core. The core can signal the end of processing by raising an interrupt to the host.

First, a controller is added that is responsible for managing the execution of the processor. After loading a new bitstream, all processors it contains are initially held frozen. The controller is then responsible to awaken each core once a program has been loaded and the program execution started, both performed via TaPaSCo API calls. Additionally, the controller is used to indicate task completion to the host via an interrupt. To this end, the RISC-V core uses memory-mapped registers to indicate program completion, after which the controller then issues an interrupt to the host and sends the core to sleep again, awaiting further tasks. The program to be executed is stored in a local instruction memory that is exclusively used by this RISC-V core. The local memory can be accessed by the host and can directly be loaded with a program before starting execution. Additionally, the core has some local memory for data. Shared memory with other cores on the device is realized through a connection to the larger off-chip memory. Optionally, an L2 cache can be inserted in front of the off-chip memory

through a TaPaSCo option to improve memory performance. The schematic design of a TaPaSCo compatible packaged RISC-V core is shown in Figure 1.

Apart from the hardware aspects, software for the cores has to be compiled as well. GCC 9.1.1 RISC-V from May 2019 is used, with optimization level `O2` and the architecture specification `rv32im` to generate code for all of the evaluated cores. Custom linker scripts are used to specify the memory structure of the custom PE. The generated `ELF` files are stripped with `objcopy` to generate a raw binary representation that can directly be loaded through the corresponding local memory API calls of TaPaSCo.

To execute a program on a given RISC-V, the standard TaPaSCo API is used. A TaPaSCo job is created and launched, having the current program as a parameter. TaPaSCo will make sure the program binary is copied to the correct memory and starts execution through the controller. Additional parameters can be used to supply extra data, located either in the PE-local data memory or in the shared off-chip memory, or to copy back data to the host after execution. Finally, the controller records the exact execution time in cycles and can thus be used as to benchmark a TaPaSCo compute job.

Note that the RISC-V ISA actually specifies a `mcycle` register for this purpose, but this might not always be implemented, or (for some cores) does not count correctly.

A. Challenges

In practice, the integration process described above was not always as straight forward. All of the cores do have slightly separate addressing schemes. Some have separate ports for instruction and data accesses, while others employ a shared port. Another issue comes with the varied use of HDLs. Especially SystemVerilog proved to be a major headache when used in Vivado. Many of these cores required wrappers in traditional Verilog before Vivado could process them. Some of them even required changes to their code, as Vivado did not support all of the features of SystemVerilog used in these cores. Even correctly wrapped cores had some issues, as IP-XACT packaging did not work flawlessly depending on the *naming conventions* used for the AXI signals. Lastly, even the packaged cores were not always working flawlessly. Not all core developers are using a clear release schedule, e.g., Git-Flow, to ensure their master repository always contains consistent states of the code. This led to broken commits which were not entirely obvious before running code on them. Another gripe was with the range of configuration options offered by some cores, as some options simply did not function correctly, even though they were supported by the code.

B. Local Memory Interface

The local memory is constructed as a Harvard architecture with separate instruction and data memories. This is a result of the dual-ported BlockRAM memories available on FPGAs. We use one port of a memory for external access, the second port is connected to the RISC-V core. As most cores have separate instruction and data buses, we can avoid interconnect logic between the core and memory and thus reduce latency with this architecture.

The preferred interfaces for our architecture are AXI4 connections to instruction memory and to data memory. AXI4 Lite, as used, e.g. in Orca behaves similar to a full AXI4 connection for our purposes. We use a custom, latency optimized BlockRAM controller to allow single cycle memory accesses via AXI. Taiga uses non-AXI local memory interfaces for instruction and data, which can be mapped to direct connections to the underlying BlockRAM. As an exception to the rest, PicoRV32 only has a single AXI interface, and does require an intermediate interconnect to interface with local memory and other peripherals.

IV. Evaluation

The evaluation of the selected cores focuses on two main aspects: (1) How well does the core work on a given FPGA, considering resource usage and achieved frequencies? (2) What is the absolute and relative (per MHz) performance of the processors?

Additionally, we describe problems faced when evaluating the cores. These might include, e.g., obvious inefficiencies when targeting FPGAs, or undocumented behaviour, both of which are always a nuisance when working with complex external IP blocks.

For comparison with a core from outside the RISC-V world, a proprietary softcore highly optimized for the target FPGA was integrated into TaPaSCo as a golden reference. It supports a configurable pipeline length, which we use to select a performance-optimized 5 stage micro-architecture, as well as a frequency-optimized 8-stage pipeline.

A. Hardware

Thanks to using TaPaSCo automation for the evaluation, the main limiting factor for the evaluation were the available core-hours on our compute cluster for mapping the hardware designs to the target platforms. Utilizing the TaPaSCo Design Space Exploration (DSE) feature, all cores are evaluated for four different target architectures: (1) Xilinx VC709, Virtex 7 PCIe based, (2) Xilinx AU250, Virtex UltraScale+ PCIe based, (3) Xilinx ZCU102, Zynq MPSoC based, and (4) Xilinx PYNQ, Zynq 7000 based. To give a good picture of the overall design, the evaluation measures (1) maximum frequency for a single core and the (2) resource utilization of a core.

a) Single Core Performance: With only one core, the designs already show a vastly different suitability for FPGA applications. As shown in Figure 2 the fastest cores in terms of maximum frequency are the PicoRV32 and VexRiscv. PicoRV32 even achieves frequencies slightly better than the highly optimized 8-stage proprietary core.

b) Resource Utilization: For most of the FPGAs used, high clock frequencies were the primary optimization goals. However, for some applications, the optimal configuration might be a *smaller* core using fewer FPGA resources and only

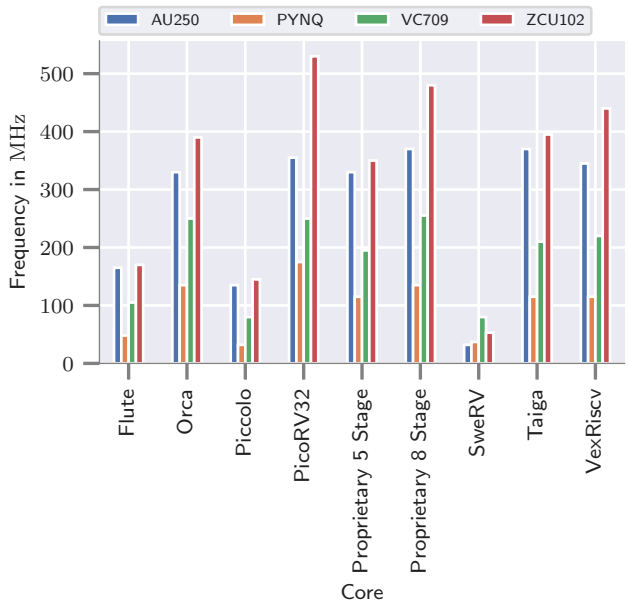

Fig. 2. Highest frequency achieved on the selected FPGA development platforms for a variety of open-source RISC-V cores. Note that the frequency itself is not a direct indicator for the performance, as certain cores might use longer critical paths to increase IPC.

achieving a *lower frequency*, but which results in superior *overall* performance, as it can be replicated more often for parallel processing in a many-core architecture. Table II shows the resource usage on all evaluated platforms. The numbers show only the PE logic (core, controller and local memory), and exclude the rest of the TaPaSCo infrastructure. BlockRAM count is converted to RAMB36 equivalents. Overall, the different platforms show a similar resource distribution.

As a result of the complex pipeline, SweRV has the largest resource utilization. Orca has the smallest utilization followed by VexRiscv, Taiga and PicoRV32. Piccolo and Flute have a resource utilization in between. Surprisingly, the different pipeline architectures of Piccolo and Flute influence the resource utilization only marginally.

B. Performance

The previous section detailed how the cores map for the various FPGA targets. Some cores might use higher frequencies to gain performance at a lower IPC, while others utilize more resources and longer critical paths to achieve a higher IPC. This section combines the results of the previous section and uses a number of widely-used benchmarks to measure application-level performance. All measurements are done in actual hardware using the target boards. When the FPGA target platform lacks an integrated CPU (e.g., the PCIe-based boards), a host machine using an AMD Ryzen 1600 CPU is used. All numbers shown do not include transfer times and are solely based on the execution times of the benchmarks of the different softcores. In the benchmarks, all data resides inside the local instruction and data memories and no accesses to external memory are required. Therefore, the results do not depend on offchip memory and are the same for all platforms.

Three benchmarks have been selected to give a broad performance overview. Dhrystone is the oldest benchmark used here and gives a basic idea of a given core's performance either as DMIPS or as DMIPS/MHz. The test is relatively simple and parts of it can be optimized away by a modern compiler. This requires some care to avoid getting unreliable results. Secondly, CoreMark from the EEMBC family is used. Lastly, Embench, a relatively new benchmark aiming to improve upon the previous benchmarks is evaluated.

a) Dhrystone: is one of the most commonly used benchmarks for comparing processor performance. The result of the benchmark is given in absolute DMIPS, or normalized for the clock frequency as DMIPS/MHz. The benchmark is relatively simple to run on any processor but, partly due to its age, does not give a very accurate representation of a processors performance. These days Dhrystone cannot be used to precisely compare performance across different test setups. The compiler used and the optimizations applied have a huge influence on the resulting DMIPS. Thus, the Dhrystone numbers reported here should not be compared with results captured in other target platform / compiler environments. But due to the long tradition of using it to indicate performance, we have included it here.

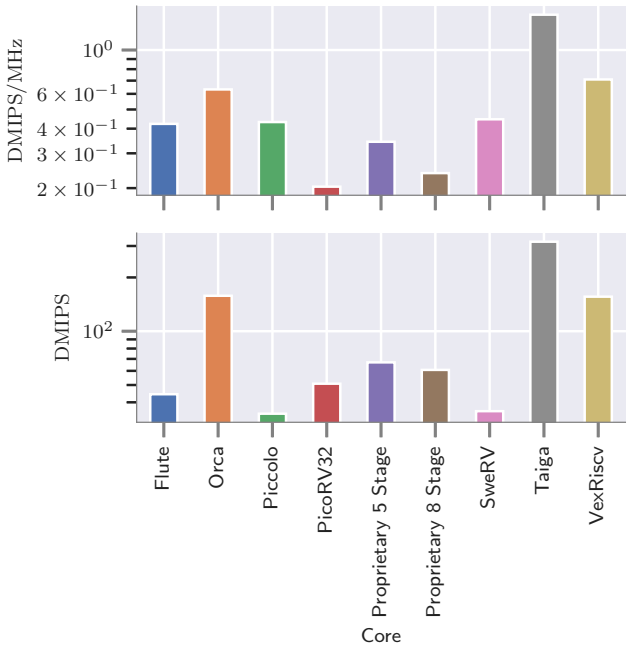

Fig. 3. Results for running the Dhrystone benchmark on the VC709 platform, both in absolute terms and normalized for the frequency. Dhrystone is a relatively inexact benchmark, as modern compilers can heavily optimize certain aspects of the benchmark away.

In Figure 3 the results of running Dhrystone on a VC709 is shown as absolute and relative performance. The best performance by far is achieved by Taiga, which wins this

TABLE II
RESOURCE USAGE OF RISC-V CORES ON ALL EVALUATED PLATFORMS.

		Flute	Orca	Piccolo	PicoRV32	SweRV	Taiga	VexRiscv	Proprietary 5-Stage	Proprietary 8-Stage
AU250	LUTs	14208	3822	14635	4037	30075	4515	3793	2929	3977
	Registers	11037	3139	10616	3341	16237	3239	3328	2765	4281
	Block RAM Tile	36	32	34.5	32	62	34	33	36	37
	DSPs	15	4	15	4	4	4	4	0	0
PYNQ	LUTs	13537	3917	13324	4143	30340	4504	3777	2902	3993
	Registers	11343	3100	10454	3363	16261	3176	3305	2787	4239
	Block RAM Tile	34.5	32	33.5	32	62	34	33	36	37
	DSPs	15	4	15	4	4	4	4	0	0
VC709	LUTs	13381	4019	13512	4249	30399	4576	3795	2997	4150
	Registers	11320	3115	10956	3296	16283	3207	3329	2779	3989
	Block RAM Tile	34.5	32	33	32	62	34	33	36	37
	DSPs	15	4	15	4	4	4	4	0	0
ZCU102	LUTs	13178	3882	13466	4233	30128	4565	3906	2932	4057
	Registers	11050	3115	10630	3345	16267	3216	3378	2766	4136
	Block RAM Tile	34.5	32	33	32	62	34	33	36	37
	DSPs	15	4	15	4	4	4	4	0	0

benchmark by a large margin in front of VexRiscv and Orca. The other cores are closer together in relative performance but none of them reach even close to 100 DMIPS on the VC709. However, they are very competitive with the highly optimized proprietary core. PicoRV32 has the lowest relative performance.

b) EEMBC Coremark: EEMBC is an organization that develops benchmarks for a wide variety of embedded microprocessor applications. They offer benchmarks for domains such as automotive, security, and networking in a variety of configurations such as single- and multi-core. CoreMark [17] is a variant aimed at all kinds of single-core microprocessors. It improves upon Dhrystone in a number of ways. For example CoreMark avoids API calls, to avoid measuring the quality of the C standard library, instead of the processor. Additionally, CoreMark tries to use more realistic code instead of the rather exotic code in Dhrystone. However, the benchmark is still considered to be a *synthetic* benchmark, and does not consist of real applications. The results of the benchmarks are combined to a single score called CoreMark, and can also be presented as CoreMark/MHz.

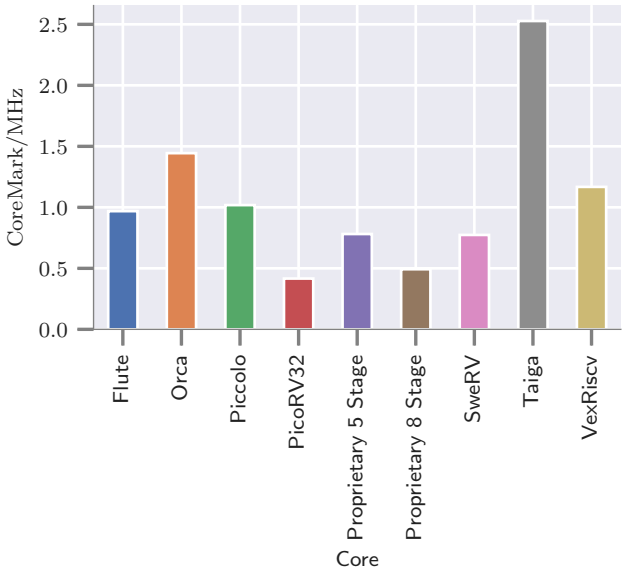

Fig. 4. CoreMark (normalized by frequency) achieved when running EEMBC CoreMark on the different cores, each running at optimal frequency for a VC709 (See Figure 2). CoreMark has been developed as a replacement for Dhrystone and aims to alleviate some of the problems such as over reliance on API calls and the susceptibility to compiler optimization's.

The results in Figure 4 show that Taiga is again leading, with a benchmark value of 2.53 CoreMark/MHz. It is followed by Orca with 1.4 CoreMark/MHz, VexRiscv with 1.2 CoreMark/MHz and Piccolo and Flute with 1.0 CoreMark/MHz. SweRV is comparable to the proprietary 5-stage core. Again, PicoRV32 has the lowest performance.

c) Embench pre 0.5: Embench [18] is a relatively new addition to the microprocessor benchmark world. Published in June 2019 by a group annoyed by the existing embedded benchmark environment, and also involved in the RISC-V foundation, its main goal is to replace Dhrystone and CoreMark as the defacto standard benchmarks with a better solution. The benchmark consists of multiple *actual* applications from of a variety of domains and does not include synthetic programs writen solely for the purpose of benchmarking. The current suite consists of: `aha-mont64`, `crc32`, `cubic`, `edn`, `huffbench`, `matmul-int`, `minver`, `nbody`, `nettle-aes`, `nettle-sha256`, `nsichneu`, `picojpeg`, `qrduino`, `sqlib-combined`, `slre`, `st`, `statemate`, `ud`, `wikisort`. Compared to the other benchmarks, Embench specifies exactly how the final score should be derived. Each benchmark's runtime is normalized for a `CPU_MHZ` value that is used to make sure the benchmark runs sufficiently long even on very fast processors. The normalized runtime is then compared to a reference platform consisting of a PULP RI5CY core using GCC 10.0.0, and then reported as a

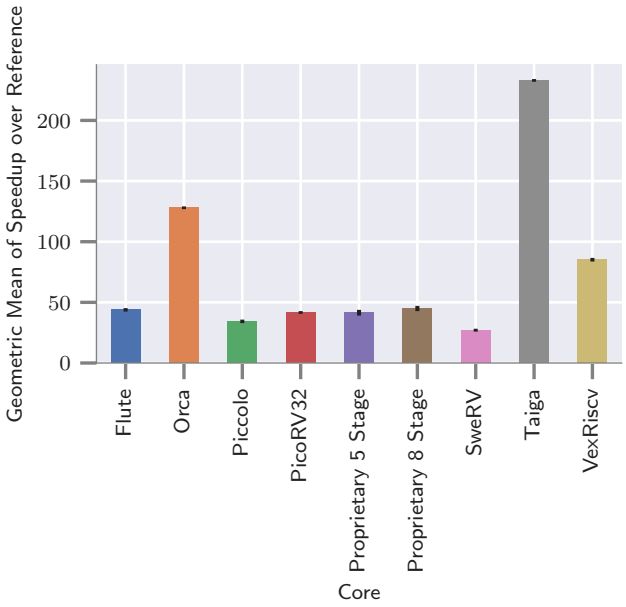

Fig. 5. Results for Embench reported as speedup over the reference platform as specified by the Embench manual. Embench is a relatively new benchmark and has been developed to better mimic embedded system usage, compared to Dhrystone and CoreMark. The benchmark is actively endorsed by the RISC-V foundation.

speedup or slowdown relative to this platform. Additionally, a geometric mean is used to combine the results of the different benchmarks into a single score. Further insight is provided through the geometric standard deviation of the runtimes. Apart from the performance evaluations, Embench also has support for code size evaluations. As the same binary is used for all cores in our evaluation, code sizes have not been considered here.

The results in Figure 5 do not contain data for the benchmarks `edn`, `slre` and `wikisort` on the proprietary standard cores, as these benchmarks failed to execute correctly on these cores.

The highest score is again achieved by Taiga and is followed by Orca and VexRiscv. The other cores are similar or slower than the highly optimized proprietary cores.

Note that the focus of this study was the evaluation of the cores in a *plug-in fashion* within the common TaPaSCo framework, as shown in Figure 1. Some of the cores can reach significantly higher performance when operating from custom memory structures. As an example, SweRV can profit immensely when executing from its own *Instruction Closely Coupled Memory* (ICCM), yielding CoreMark/MHz improvements on the order of 5x over those shown in Figure 4. However, program loading would then have to be performed using a special version of the OpenOCD, instead of the general-purpose memory-mapped loader used in TaPaSCo for all cores.

V. Conclusion and Outlook

Thanks to the open-source nature of RISC-V, development of RISC-V cores has been taken up by both academia as well as industry. This has resulted in a wide range of cores that have not been easy to compare thus far due to their different interfaces and stand-alone implementations. This paper enables that comparison by running all of the cores in the same environment, and also making them easily accessible as accelerator PE using the general TaPaSCo API. The resulting packaged cores can be evaluated and used on any platform supported by TaPaSCo. The packaging scripts are released as open-source on https://github.com/esa-tu-darmstadt/tapasco-riscv.

Most surprisingly, some of the open-source cores provide much higher performance than the highly optimized proprietary cores, with only very limited area overhead (e.g., Taiga and Orca compared to the proprietary 8-stage core).

There is still potential to improve upon this study. One important item would be the integration of the *debugging* facilities supported by the cores. Currently, debugging is mainly done through simulation, or "printf-debugging". This extension will be much easier once more cores adopt the RISC-V debugging interfaces, instead of the current custom-made solutions. Additionally, support for peripherals such as UART controllers should also be integrated.

Also, the debugging facilities such as OpenOCD could be repurposed to allow the loading of binaries into custom memories, such as the SweRV ICCM, to allow far higher execution speeds.

Acknowledgment

This research was funded by the German Federal Ministry for Education and Research (BMBF) with the funding ID 01 IS 17091 B.

References

[1] J. Korinth, J. Hofmann, C. Heinz, and A. Koch, "The TaPaSCo open-source toolflow for the automated composition of task-based parallel reconfigurable computing systems," in *International Symposium on Applied Reconfigurable Computing (ARC)*, 2019.

[2] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA," *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, vol. 116, 2011.

[3] Western Digital, *Western digital delivers new innovations to drive open standard interfaces and RISC-V processor development*, https://www.westerndigital.com/company/newsroom/press-releases/2018/2018-12-04-western-digital-delivers-new-innovations-to-drive-open-standard-interfaces-and-risc-v-processor-development, Dec. 4, 2018. (visited on 07/10/2019).

[4] SiFive, *About - SiFive*, https://www.sifive.com/about. (visited on 07/10/2019).

[5] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[6] J. Gray, "GRVI phalanx: A massively parallel RISC-V FPGA accelerator accelerator," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2016, pp. 17–20.

[7] M. Gielda, *RISC-V cores and SoC overview*, https://github.com/riscv/riscv-cores-list. (visited on 06/30/2019).

[8] E. Matthews and L. Shannon, "TAIGA: A new RISC-V soft-processor framework enabling high performance CPU architectural features," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2017, pp. 1–4. DOI: 10.23919/FPL.2017.8056766.

[9] VectorBlox Computing Inc., *ORCA*, https://github.com/VectorBlox/orca. (visited on 08/13/2019).

[10] Western Digital Corporation, *SweRV*, https://github.com/chipsalliance/Cores-SweRV. (visited on 06/27/2019).

[11] Bluespec, Inc., *Piccolo*, https://github.com/bluespec/Piccolo. (visited on 10/05/2019).

[12] ——, *Flute*, https://github.com/bluespec/Flute. (visited on 10/08/2019).

[13] SpinalHDL, *VexRiscv*, https://github.com/SpinalHDL/VexRiscv. (visited on 10/24/2019).

[14] C. Wolf, *PicoRV32 - a size-optimized RISC-V CPU*, https://github.com/cliffordwolf/picorv32. (visited on 09/26/2019).

[15] F. Zaruba, *Ariane RISC-V CPU*, https://github.com/pulp-platform/ariane. (visited on 07/15/2019).

[16] IIT Madras, *Shakti E-Class processor*, http://shakti.org.in/e-class.html. (visited on 07/28/2019).

[17] S. Gal-On and M. Levy, "Exploring coremark a benchmark maximizing simplicity and efficacy," *The Embedded Microprocessor Benchmark Consortium*, 2012, https://www.eembc.org/techlit/articles/coremark-whitepaper.pdf (visited on 07/24/2019).

[18] J. Bennett, *Embench: Open benchmarks for embedded platforms*, https://github.com/embench/embench-iot. (visited on 07/10/2019).