TAIGA: A CONFIGURABLE RISC-V SOFT-PROCESSOR FRAMEWORK FOR HETEROGENEOUS COMPUTING SYSTEMS RESEARCH

Eric Matthews and Lesley Shannon

School of Engineering Science
Simon Fraser University
Burnaby, British Columbia
ematthew@sfu.ca, lshannon@ensc.sfu.ca

ABSTRACT

Heterogeneous computing architectures that combine high-performance multicore processors with custom hardware accelerators offer the potential to increase compute efficiency. In this paper, we discuss our goal of creating an open-source, configurable, multicore framework for heterogeneous computing systems research on Field Programmable Gate Arrays (FPGAs) using the RISC-V ISA and the types of research questions we believe it can be used to investigate. As the basis of this desired framework, we describe Taiga, our RISC-V 32-bit IMA (Integer, Mult/Div, Atomic) ISA soft processor. Its design has been specifically targeted at Intel and Xilinx FPGA fabrics to reduce resource usage and improve operating frequency to facilitate heterogeneous computing systems research. The processor design is both modular and extensible so as to facilitate the inclusion of new functional units (i.e. custom instructions) and for the eventual extension of the core to the full RISC-V ISA. It also includes necessary hardware infrastructure to support an O/S as well as replication for multicore architectures.

***Index Terms*—** configurable, extensible, soft-processor, RTL, variable-length pipeline, parallel execution-units, heterogeneous computing systems research, FPGAs

1. INTRODUCTION

Heterogeneous compute systems combine multiple processing elements with different Instruction Set Architectures (ISAs) (e.g. x86, GPGPUs, etc.) and/or custom hardware (HW) accelerators with the goal of providing better compute efficiency. The subset of heterogeneous systems that combine one or more processors with custom HW accelerators that are implemented on Field Programmable Gate Arrays (FPGAs) are of increasing interest with Intel's purchase of the second largest FPGA vendor and the fact that FPGAs are now commonly found in data centres.

This is a very active area, as we try to answer the question: How do we incorporate reconfigurable fabric and custom HW accelerators into the computing hierarchy? Even if we assume an infinite reconfigurable fabric able to implement all the custom HW accelerators for all of the applications executing on heterogeneous platforms at any given time, there are many interesting research questions to consider. For example, should we integrate HW accelerators as custom instructions into a CPU's datapath? Or, should they be more loosely integrated as their own independent compute units with extremely limited ISAs? While a mixture of these two options might make the most sense-depending on the complexity of the HW accelerator, how would we identify and support these varied levels of integration dynamically for unknown workloads at runtime?

For platforms running unknown multi-application workloads, there are challenges in selecting appropriate accelerators, and sharing them where possible. Simply recognizing that reconfigurable fabric is a fixed resource of a fixed size that must be shared between accelerators, and possibly applications, adds additional dimensions. From architecture to resource management (sharing and scheduling) to virtualization and security (guaranteed execution isolation of HW "threads" on reconfigurable fabrics), these types of heterogeneous computing systems offer many interesting and exciting research questions for the future.

In this paper, we discuss our goal of creating an FPGA-based, open-source[1], configurable, multicore framework for heterogeneous computing systems research using the RISC-V ISA and the types of research questions we believe it can be used to investigate. As the basis of our desired framework, we describe Taiga [1], our modular, extensible, RISC-V 32-bit IMA (Integer, Mult/Div, Atomic) ISA soft processor and how future work will extend it to create a multicore framework for heterogeneous computing systems research. Taiga's design has been specifically targeted at Intel and Xilinx FPGA fabrics to reduce resource usage and improve operating frequency to facilitate heterogeneous computing systems research. Unlike other RISC-V soft processors aimed at FPGAs, Taiga is designed to:

- support variable-latency execution of functional units,
- enable parallel execution of functional units (to increase Instruction Level Parallelism),
- ease integration of custom functional units into the CPU datapath,
- allow replication of functional units (traditional or custom), and
- support scratchpad *and* cache-based memory coherence models, and
- in the future, enable high performance processor functionality (out-of-order commits, etc.).

Taiga has been designed for the eventual extension of the core to the full RISC-V ISA and to enable its extension to the 64-bit ISA if desired. Its configurations also include hardware infrastructure necessary to support an O/S as well as replication for multicore architectures.

Although multiple other RISC-V processors currently exist [2–4], those that are specifically targetted at FPGA platforms [3, 4] are not aimed at systems level research and are not readily extensible to supporting an O/S or Symmetric/Asymmetric Multicore Configurations. The novelty of Taiga is that: 1) it is specifically designed as a CPU core for Symmetric and Asymmetric multicore configurations with an

[1] https://gitlab.com/sfu-rcl/Taiga

O/S, and 2) its unique soft processor datapath architecture comprises parallel-execution, variable-latency functional units increases ILP, making it a better suited soft processor architecture for research into the inclusion of new custom instructions into its datapath.

The remainder of this paper is organized as follows. Section 2 summarizes the works related to this paper, including relevant FPGA-mappable RISC-V architectures and existing soft-processor-based multicore architectures. Section 3 provides an overview of Taiga's architecture and Section 4 discusses how we envision it being extended to a multicore design and how it can be used in heterogeneous computing systems research. Finally, Section 5 concludes the paper, summarizing our plans for future work.

2. RELATED WORKS

Heterogeneous computing systems research is a very active area. In this section, we provide an overview of some of the frameworks and infrastructure that can be used to enable this research- particularly for those that wish to map their designs to FPGA fabrics. We summarize existing work on RISC-V architectures that can be mapped to FPGA platforms, highlighting why Taiga is more appropriate for heterogeneous systems based research on FPGAs. We also mention some of the other platforms for computing simulation and discuss why we are excited about the opportunities presented by an open-source, FPGA-based platform for heterogeneous computing systems research. Finally, we outline the existing soft-processor-based multicore platforms for FPGAs.

ORCA [3] and GRVI [4] both focus on optimizing the resource usage of their RISC-V processor implementations for FPGA-based platforms. However, neither include the key hardware support required for running an O/S, such as a memory management unit (MMU), translation lookaside buffers (TLB) and caches. Both architectures also rely on a fixed-pipeline architecture that limits the amount of ILP that can be extracted for non-vectorized software and they do not have a generic interface for incorporating custom HW accelerators as custom instructions. Taiga's parallel-execution, variable-latency functional units are designed to increase ILP for the datapath and readily support tight integration of custom instructions.

When complex out-of-order ASIC designs [5, 6] have been ported onto FPGAs, they did not map well to FPGA fabrics, which resulted in extremely large designs with low operating frequencies. The RISC-V Rocket processor [2] is currently the best soft-processor-based RISC-V implementation for systems research. It provides interfaces for custom hardware accelerators, supports multicore configurations and includes features for O/S support and other features common to modern CPUs (e.g. branch predictors and caches). However, the architecture is not aimed at FPGA-based platforms, meaning that key architectural features (e.g. the branch predictor) do not map well to the FPGA fabric or leverage the embedded cores in the fabric (e.g. the DSP units and LUT-RAMs) to reduce resource usage and increase operating frequency. As we discuss in Section 3, this results in a significantly larger and slower configuration when mapping onto an FPGA. Taiga [1] provides an additional opportunity for architectural researchers for heterogeneous systems as it supports an extensible modular interface for memory subsystems supporting both caches and scratchpad memory for instruction and data sources.

Software simulators (e.g. Simics [7], SimpleScalar [8], GEMS [9], GEM5 [10]) have provided common platforms for computing systems architectural researchers to investigate potential new architectural features. In fact, work has been done to extend software simulators to support reconfigurable fabrics as part of their simulation [11]. Our proposed framework is meant as a complementary alternative to these platforms: software simulation can provide the user with unlimited visibility to software execution but at speeds that are orders of magnitude slower than running the same system configurations using our proposed FPGA-based computing systems framework. While a soft-processor based platform does not provide the same visibility as traditional software simulators, it is possible to instrument FPGA-based computing platforms to provide detailed trace data that can provide researchers the types of insights required to understand key runtime interactions (e.g. memory accesses) [12].

There are multiple soft-processors with ISAs other than the RISC-V that are vendor agnostic [13–15], however, there only are only three multicore soft-processor architectures that are designed to boot a Linux kernel: 1) the RISC-V Rocket processor [2], the LEON3 [15], and the PolyBlaze [16]. However, neither the LEON3 nor Rocket processors are aimed at FPGA fabrics, making them both larger and slower than FPGA-targeted designs. Conversely, the PolyBlaze is designed for FPGA fabrics as it is based on Xilinx's MicroBlaze [17] CPU architecture. But it is also not not open source, as the MicroBlaze is proprietary. Furthermore, the Rocket processor, PolyBlaze and LEON3 all have fixed-pipelines, which are not well suited to tightly integrating custom instructions. We believe that the Taiga soft-processor targets a design space with specific opportunities. Its unique parallel-execution, variable latency functional unit design provides exciting architecture research opportunities. Furthermore, by targeting the design to FPGA fabric, it is able to significantly reduce resource usage while increasing operating frequency so that researchers will be able to instantiate large multicore configurations on an actual device and still have considerable room for HW accelerators. This increases the type and range of custom HW accelerators that can be investigated along with the levels of integration (i.e. tightly, loosely, or mixed).

3. OVERVIEW OF THE TAIGA PROCESSOR

Figure 1 presents a overview of the Taiga processor [1] and pipeline details. As each unit in Taiga has different latencies and throughputs, each execution unit has its latency (top number) and iteration interval-the rate at which a unit can start additional requests (bottom number)-included. For example, the multiplier unit has a latency of 2 cycles (top number) for an operation and is fully pipelined and able to start a new request every cycle (i.e. bottom number of '1'). Conversely, the divider is not pipelined. Its latency is dependent on whether there is an early exit corner case (e.g. overflow, divide by zero, the result of a previous div/rem operation). Otherwise, it takes an additional 32 cycles (for a total of 34) to complete.

As clearly shown in Figure 1, all the functional units have independent execution paths. As such, even though the processor has a scalar (single-issue) architecture, it is possible for multiple instructions to be in flight in the execution phase at any given time. This not only increases the ILP of the underlying architecture, but facilitates the addition of new functional units (e.g. an FPU or a custom instruction). By creating a generic, scalable interface to the decode/issue phase of the processor, it is easier to extend its architecture for new functional units. Section 4 will discuss how this architecture can be leveraged for heterogeneous computing systems research.

Taiga supports the 32-bit base integer instruction set with the Multiply/Divide and atomic operations extensions (RV32IMA) [1]. It is described using SystemVerilog and designed to be FPGA vendor

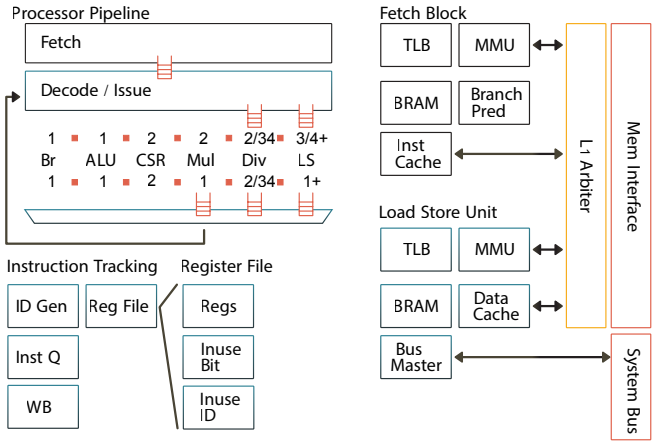

Fig. 1. Taiga Pipeline Structure and Overview. Latency and throughput information is provided for each of the execution units excluding the effect of input/output FIFOs. The number above each unit is the cycle latency for that given unit. The number below the unit is the rate, in cycles, that the unit can start additional requests. Multiple listings of numbers indicates that the latency can very. A variable latency is indicated with a number followed by a plus symbol.

Table 1. Full System (Processor, Bus with UART, memory interface) LEON3 and Taiga Configurations versus Rocket Processor (plus memory interface) Resource Usage Comparison

	LUTs	FFs	Slices	BRAMs	DSPs	Freq(MHz)
LEON3	6,704	3,640	2,042	12	4	75
Rocket	17,144	9,058	5,536	10	0	54
Taiga	3,998	2,942	1,371	10	4	104

agnostic where possible and leverage specific features where appropriate, supporting both Intel and Xilinx FPGAs. Along with selecting which vendor the processor is being mapped to, its configuration parameters allow the inclusion/exclusion of an: integer divider, integer multiplier, instruction cache, data cache, scratchpad (data and instruction), memory management unit, translation lookaside buffer, and branch predictor. The size and address range of the scratchpad is configurable as is the number and types of functional units supported and the type of bus. Both the data and instruction cache allow the user to select the number of lines, ways, and the number of words in a line. The data and instruction TLBs allow users to select the number of ways and depth. Finally, the user can select the number of entries in the branch predictor table.

Taiga's modular design allows users to easily extend or alter subcomponent functionality. For example, while both caches use a random replacement policy, the replacement policy subcomponent could easily be replaced with an alternate policy (e.g. Least Recently Used, clock, etc.). Similarly, the branch predictor simply uses a "previous branch" policy (i.e. predicting a jump to the same destination as the previous branch), which can also easily be replace by the user. In both cases, these polices have been chosen to optimize for resource usage and operating frequency as opposed to Instructions Per Cycle (IPC).

Table 1 summarizes the resource usage and operating frequency of the Taiga soft processor system along with a LEON3 and Rocket RISC-V processor in single core configurations [1]. The LEON3 and Rocket processors have been selected for this comparison as they are the only two open-source, soft-processors scalable to multicore configurations that are designed to boot an operating system. For this comparison, the configurations of all three processors have been set to be as close as possible. All three designs are mapped to a Xilinx Zynq Z7CZ020 FPGA on a zedboard for the resource and operating frequency comparisons. All three processors are configured with: 1) no scratchpad memories, 2) 8KB, 2-way instruction and data caches with random replacement policies, 3) MMU support, 4) integer multiply and divide support only (no Floating point units), 5) default branch predictors and 6) no local memory. The resource usage numbers for both the LEON3 and Taiga processor also include a system bus and a UART, whereas the Rocket resource usage numbers are only for the processor core and its memory interface. The final configuration difference is that we are using the 64-bit Rocket processor, whereas both the LEON3 and Taiga are configured as 32-bit processors. This difference is due to the fact that at the time of this submission, the 32-bit Rocket full system configuration (processor plus memory interface) could not be generated for compilation by the Vivado synthesis tools.

As can be seen from the results reported in Table 1, the Taiga processor uses considerably fewer resources and has a significantly higher operating frequency than both the LEON3 and Rocket RISC-V processors. Although the LEON3 features a different ISA than the Taiga processor, when configured as a 32-bit processor, it uses 1.5x more slices than the Taiga processor while operating at a 28% slower operating frequency. When combined with the limited range of multicore configurations (a maximum of four cores), we believe that this is more limiting for heterogeneous systems researchers. The Rocket RISC-V processor allows users to select any non-zero number of cores. Our future multicore versions of Taiga are planned to support up any non-zero configuration up to at least 8 cores (possibly more). Minimally, we allow at least 8 cores as previous work suggests that it may be possible to scale our system up to 8 cores without impacting its operating frequency [16]

Obviously, the Rocket RISC-V is expected to be larger than the Taiga processor due to its 64-bit ISA. However, simply having a wider data does not result in the more than 4x increase in resources. Instead, some of the additional area is due to more complex processor features (e.g. compressed instruction set) whereas other increases in resource usage are due to the poor mapping of specific architectural features (e.g. the branch predictor) to the fabric. Finally, the Rocket RISC-V does not utilize any of the FPGA's embedded DSP slices for arithmetic operations. This results in an additional increase in size and critical path delay for arithmetic operations.

Table 2. Taiga configuration comparisons. All Cache configurations assume a random replacement policy.

Configuration Details	LUTs	FFs	BRAMs	DSPs	Freq(MHz)
Minimal: 4KB scratchpad; Excludes: Div, Mult, caches, MMU, TLB, and branch predictor	1,551	971	1	0	123
Div and Mult: Div, Mult, and 4KB scratchpad; Exclude: caches, MMU, TLB, branch predictor and scratchpad	2,018	1,121	1	4	119
Scratchpad: Div, Mult, branch predictor, and 16KB scratchpad; Excludes: caches, MMU, and TLB)	2,159	1,193	5	4	115
Baseline Config (from Table 1): Div, Mult, 2-way 8KB D$ and I$, MMU, TLB, branch predictor, and no scratchpad	2,851	1,512	7	4	110
Everything: Div, Mult, 2-way 8KB D$ and I$, MMU, TLB, branch predictor, and 4KB scratchpad	2,937	1,514	8	4	104

Table 2 illustrates how the Taiga processor scales in resource usage for different configurations that might be desired by the user. For all configurations, the Taiga processor operating frequency is at least 104 MHz (as listed in Table 1). Taiga's minimal configuration configuration requires a scratchpad as the processor requires either a scratchpad or cache to act as a local memory interface with the system memory. The 4KB scratchpad requires a single BRAM and no DSP slices are needed without the divider and multiplier functional units. The Div and Mult configuration adds the divider and multiplier functional units to the minimal configurations. It increases the LUT utilization by 30% and requires an additional 4 DSP blocks to implement the new arithmetic units.

The scratchpad configuration increases the size of the scratchpad to 16KB and adds the branch predictor. This nominally increases the LUTs required (7%). This configuration requires 5 BRAMs, 4 for the 16KB scratchpad and 1 for the branch predictor table. The baseline configuration is the same as the one described in Table 1. The scratchpad has been removed and replaced with 2-way 8KB Data and Instruction caches, along with the addition of the MMUs and TLBs as well. This significantly increases the LUT (32%) and flipflop (27%) usage to support these additional units. The configuration also requires 7 BRAMs. Each cache requires two BRAMs, one for each way to allow parallel access of the ways, and one for the tag bank, for a total of 3 BRAMs per cache. The 7th cache is used by the branch predictor. The final configuration includes "everything"; it includes everything in the baseline configuration and adds in a 4KB scratchpad. This minimally increases the LUT usage (3%) and adds an additional BRAM for the scratchpad memory relative to the baseline configuration.

.

4. THE FUTURE: AN OPEN-SOURCE, MULTICORE FPGA-BASED PLATFORM FOR HETEROGENEOUS COMPUTING SYSTEMS RESEARCH

In this section, we first describe how the Taiga processor will be scaled to a multicore configuration. We then discuss how such an architectural framework can be used for heterogeneous multicore systems research.

As stated previously, we believe that our proposed processor framework is complimentary to the Rocket system. Taiga is currently designed to support the privileged instruction set (1.10), with current support for the 32-bit IMA (Integer, Mul/div, Atomic), and configurable FPU support in the near future to support the full "default" RISC-V configuration. With the privileged instruction set support Taiga will be capable of supporting Linux. As it is specifically targeted at Intel and Xilinx FPGA fabrics, it offers considerable resource usage and performance benefits for those that wish to actually map their designs to FPGAs - particularly for multicore configurations- as part of their research investigations. Based on our experience with the PolyBlaze system [16], we plan to scale the Taiga to the multicore architectural framework shown in Figure 2. Each Taiga

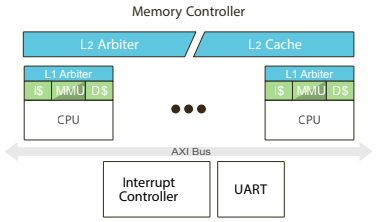

Fig. 2. Envisioned multicore RISC-V configuration

core will have its own instruction and data caches as well as a MMU. An interface, called the L1 arbiter, will act as a bridge to connect the CPU core to the L2 arbiter. The L2 arbiter acts provides a generic interface for each CPU core to the rest of the memory system. It ensures system-level coherency amongst the cores. Both the L1 and L2 arbiter designs are based on our previous PolyBlaze system designs [16]. Depending on the user's preference, the L2 arbiter can then be used to either connect the multicore system to a system-level L2 Cache (as shown in Figure 2) or directly to the main memory system. For Intel and Xilinx FPGAs, we expect that the user would likely map the L2 cache to the fabric's on-chip memory (BRAMs). However, the main memory system is expected to be stored off-chip in DDR (or flash). This will provide sufficient memory for a Linux kernel as well as the benchmark applications and their data sets.

Although the system will be designed assuming a symmetric multiprocessor (SMP) configuration, the proposed CPU and system-level architecture offer users the ability to investigate both tightly integrated HW accelerators (as "custom instructions") or loosely integrated HW accelerators that would act as independent compute units, or some mix of the two. In the case of custom instructions, researchers would be able to update the RISC-V GCC compiler flow to recognize the new operations and directly compile to the Taiga processor. Architecturally, the generic functional unit interface, shown from the Decode phase in Figure 1, allows researchers to easily include any number of any type of functional unit they wish. When combined with the RISC-V GCC support, this provides an automated flow for researchers to evaluate software application mappings to RISC-V processors with tightly integrated accelerators. This will allow researchers to focus more on how to leverage these custom instruction configurations in heterogeneous systems, where each processor has one or more different HW accelerators tightly integrated into its pipeline. This leads to interesting research questions, such as, how to map unknown workloads onto these multicore systems that comprise one or more threads that may wish to share these custom instructions (even between applications). Alternately, as the processor supports both caches and scratchpads, researchers can investigate how data is marshalled for custom instructions- should they simply fetch from the data cache or is a scratchpad more appropriate.

The proposed system-level architecture can also be used to loosely integrate HW accelerators into a heterogeneous compute architecture. In this case, the L2 arbiter, shown in Figure 2 provides a generic

interface for new compute units. A CPU core can be replaced by a custom HW accelerator with its own equivalent of the L1 Arbiter. Researchers can then investigate how the HW accelerator should marshall its data, using either a scratchpad or cache, while ensuring system-level coherency. If a cache is to be used, what type(s) of replacement policies would function well. Software questions such as scheduling threads to use HW accelerators as compute units and sharing these compute units between applications also become interesting. In short, these custom HW accelerators become services at the system-level that should be both scheduleable by the O/S and shareable between applications in an unknown workload at runtime.

Finally, these two styles of integration, tightly integrated custom HW instructions and loosely integrated custom HW compute units, need not be mutually exclusive. They are building blocks that allow computer architecture researchers a rich design space to explore for heterogeneous systems. Taiga's proposed scaling to a multicore system will provide researchers with all the necessary infrastructure to support both the hardware and software design space for this research.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we discussed the Taiga soft-processor core and future plans to scale it to a multicore processor that will support both symmetric and asymmetric configurations. We described why we believe that this new soft RISC-V processor will help to fill an important research infrastructure need as it is extended to a multicore architecture. Specifically, its unique soft-processor pipeline architecture supporting parallel execution of variable latency functional units is well suited to investigating tightly integrated custom instructions into processor architectures. Furthermore, it supports caches and scratchpads, allowing researchers to investigate alternate memory mechanisms for storing local data for custom instructions. It is designed to be both modular and extensible so that researchers can add new functional units (e.g. an FPU) and even new high performance features (e.g. out-of-order commits). Finally, the processor core has been specifically designed for FPGA fabrics to provide reduced resource usage and higher operating frequencies. This should ensure that it will scale well to larger multicore configurations while leaving ample reconfigurable fabric for custom instructions and loosely integrated custom hardware accelerators.

Our immediate future work will be to bring up a Linux kernel on the single core configuration and add floating point unit support to provide support for the full "default" RISC-V configuration. We will then scale the the Taiga core to an open-source multicore architecture supporting symmetric and asymmetric configurations, with individual processors allowing unique cache and custom instruction configurations for heterogeneous systems research. This infrastructure will allow us and others to investigate key research questions for programming, virtualization, resource management and architecture of heterogeneous computing systems.

6. REFERENCES

[1] E. Matthews and L.Shannon, "Taiga: A new risc-v soft-processor framework enabling high performance cpu architectural features," in *To appear in FPL*, 2017.

[2] K. Asanovi *et al.*, "The rocket chip generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016.

[3] "ORCA: RISC-V by VectorBlox." [Online]. Available: github.com/VectorBlox/orca

[4] J. Gray, "Grvi phalanx: A massively parallel risc-v fpga accelerator accelerator," in *FCCM*, May 2016, pp. 17–20.

[5] G. Schelle *et al.*, "Intel nehalem processor core made fpga synthesizable," in *The ACM/SIGDA Int'l Symp. on FPGAs*, 2010, pp. 3–12.

[6] F. J. Mesa-Martinez *et al.*, "Scoore santa cruz out-of-order risc engine, fpga design issues," in *(WARP), held in conjunction with ISCA-33*, 2006, pp. 61–70.

[7] P. S. Magnusson *et al.*, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[8] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[9] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[10] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[11] P. Garcia and K. Compton, "A reconfigurable hardware interface for a modern computing system," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*. IEEE, 2007, pp. 73–84.

[12] N. C. Doyle, E. Matthews, G. Holland, A. Fedorova, and L. Shannon, "Performance impacts and limitations of hardware memory access trace collection," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 506–511.

[13] U. of Cambridge, "The tiger mips processor," University of Cambridge, Tech. Rep., 2010. [Online]. Available: www.cl.cam.ac.uk/teaching/0910/ECAD+Arch/mips.html

[14] "Arm cortex-m1 processor," ARM Ltd. [Online]. Available: arm.com/products/processors/cortex-m/cortex-m1.php

[15] *GRLIB IP Core User's Manual*. [Online]. Available: www.gaisler.com/products/grlib/grip.pdf

[16] E. Matthews, L. Shannon, and A. Fedorova, "Polyblaze: From one to many bringing the microblaze into the multicore era with linux smp support," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 2012, pp. 224–230.

[17] *MicroBlaze Processor Reference Guide*, Xilinx Inc. [Online]. Available: xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug984-vivado-microblaze-ref.pdf