TAIGA: A NEW RISC-V SOFT-PROCESSOR FRAMEWORK ENABLING HIGH PERFORMANCE CPU ARCHITECTURAL FEATURES

Eric Matthews, Lesley Shannon

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

ABSTRACT

Recently, there has been an increased focus on integration of reconfigurable fabric with modern processors. However, existing soft-processors are optimized to leverage older FPGA fabrics, focus primarily on resource minimization and have fixed-pipeline designs that limit the scope for tightly integrated hardware accelerators.

In this work, we present Taiga: a RISC-V, 32-bit, soft-processor architecture supporting the RISC-V Multiply/Divide and Atomic operations extensions (RV32IMA) designed to support Linux-based shared-memory systems. The processor design is highly configurable and features a standardized interface for functional units allowing for ease of integration of new functional units. Despite a more complex pipeline, our design uses approximately 33% fewer slices while clocking 39% faster than a LEON3 based system built on a Xilinx Zynq X7CZ020.

1. INTRODUCTION

Process technology improvements have allowed FPGA manufacturers to both increase the capacity of modern FPGAs *and* the functionality of many of their core components. Both Intel and Xilinx FPGAs now include wider LUTs, more complex DSP units and LUT based RAMs. Many of the constraints that existed when soft-processors were originally designed have changed/no longer exist due to these architectural changes and increases in capacity.

Soft-processors can also fill an important role in heterogeneous systems architecture research, such as how to integrate hardware (HW) accelerators into the system design. However, most existing soft processors focus on resource minimization, with fixed-pipeline architectures that do not reflect modern processor designs. This design constraint can significantly impact Instruction Level Parallelism (ILP) for designs that try to integrate HW accelerators into the CPU pipeline.

Our objective is to facilitate research into heterogeneous processor systems and high performance architectural features for soft-processors. We believe that FPGAs provide a unique opportunity for integrating HW accelerators with processors for embedded and data centre based computing. As such, we need an open source processor that can readily support the integration of functional units with variable latency and have the capability to leverage parallel resources. *Our goal is to leverage the changes in FPGA technology to produce a soft-processor that features higher ILP and comparable operating frequencies to commercial/existing soft-processors with the same configurations without significantly increasing area.*

Our first step towards that goal is Taiga: a highly configurable open source[1] RISC-V processor framework for general purpose soft-processor computing, heterogeneous systems research and soft-processor architectural research. Taiga is designed to enable high-performance processor features that map efficiently to FPGA fabrics and be readily extensible for investigations into heterogeneous processor systems. In this paper we introduce the Taiga processor, provide an overview of its pipeline structure, its execution unit interface and provide resource and frequency comparisons against existing similarly featured soft-processor systems.

The remainder of the paper is structured as follows. Section 2 discusses related works on soft-processors and designing for high performance soft-processors. Section 3 presents an architectural overview of the Taiga processor and Section 4 presents a resource and frequency comparison of the processor architecture. Finally, Section 5 concludes the paper.

2. BACKGROUND

Existing commercial soft-processors, like the NIOSII [1], MicroBlaze [2], and ARM Cortex-M1 [3], are fixed-pipeline designs optimized primarily for resource usage and secondly frequency. While the MicroBlaze and NIOSII both have the infrastructure to support an Operating System (OS), they are not open source. This prevents their use for architectural exploration and limits their use in developing heterogeneous systems that support their fixed mechanisms for HW accelerator integration. The LEON3 processor [4], an ASIC design that can be built for FPGAs, also features a fixed-

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

pipeline. While it is open source, it is not optimized for FPGAs and its fixed pipeline reduces its flexibility for architectural exploration.

Some researchers have performed investigations into FPGA considerations for soft-processors. Wong et al. investigated CMOS versus FPGA considerations [5], and Yiannacouras et al. explored the design space for fixed-pipeline architectures [6]. However, little design-space exploration has been performed for soft-processors with multiple parallel execution units.

There has long been interest in more complex processor architectures for FPGAs, but existing ASIC processors [7, 8] do not map well to FPGAs. Previous work investigated out-of-order superscalar design elements for FPGAs, such a reorder buffer [9] and an instruction scheduler [10]. Other work studied increasing the performance of fixed-pipeline processor designs through techniques such as runahead execution [11]. However, there is a large design space in between fixed-pipeline designs and large, out-of-order superscalar designs.

FPGAs are an ideal platform to investigate heterogeneous processor systems, but to fully investigate a range of accelerator integration opportunities, it should be possible to have tightly coupled designs. While both the MicroBlaze and NIOSII support HW accelerators, their fixed-pipeline designs limit instruction level parallelism as their pipelines are not designed to leverage variable latency operations. The CHIMAERA system [12] demonstrates the type of tightly-integrated accelerators that could be explored with a processor design that features parallel execution units.

2.1. RISC-V processors

There are an increasing number of RISC-V based processors; however, we are currently aware of only two RISC-V processors designed for FPGAs, GRVI [13] and ORCA [14]. Both of these processors have different design goals from Taiga. They focus on small, and in the case of GRVI extremely small, resource footprints. Additionally, neither processor implements TLBs/MMUs or caches, all of which are needed to properly support an OS. The Rocket processor [15], which implements the 64-bit version of the RISC-V ISA along with all current extensions, can be built for an FPGA and features OS support. However, it is designed firstly for ASICs and is quite large on an FPGA. The purpose of Taiga, is to provide the functionality to support an OS, facilitate the integration of HW accelerators and enable research into soft-processor high performance architectural features.

3. THE TAIGA PROCESSOR

Taiga is a 32-bit RISC-V processor, supporting the 32-bit base integer instruction set along with the Multiply/Divide extension and the atomic operations extension (RV32IMA).

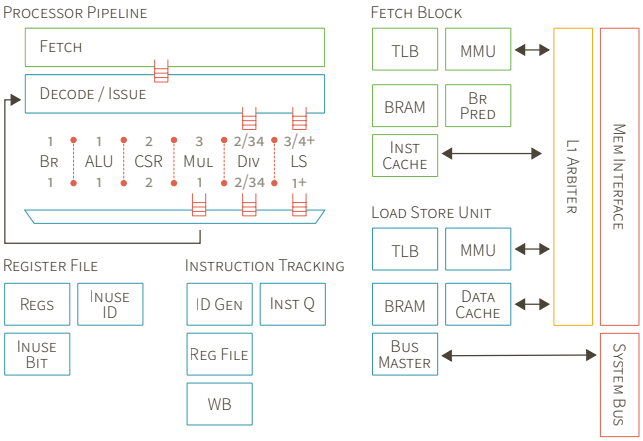


Fig. 1. Taiga Processor Pipeline Details and Components. For Execution units, the upper number indicates the cycle latency for an instruction for that unit. The lower number is the throughput expressed as one instruction every X cycle(s). A slash indicates there are two possible paths, and a plus indicates a minimum latency.

The processor is written in SystemVerilog, is highly configurable and is designed to be largely FPGA vendor agnostic, supporting both Intel and Xilinx FPGAs. Caches, TLBs, choice of bus standard, and inclusion of multiply and divide operations are all configuration options.

With a focus on facilitating research into both heterogeneous processor systems and high performance soft-processors, Taiga's pipeline is structured as multiple independent execution units. The execution units have no restrictions on latency. Taking this design approach we can have a standardized execution unit interface allowing for the ease of integration of new functional units into the processor. Additionally, this approach enables us to leverage more Instruction Level Parallelism (ILP). The pipeline depth for the fetch logic and *Load Store unit (LS unit)* are designed with consideration for the TLBs and caches, which are needed for OS support in shared-memory systems.

A detailed diagram of Taiga's pipeline, execution unit latencies/throughputs, and major components is presented in Figure 1. As each execution unit has different pipeline depths and behaviours, we have listed both the latency for an instruction issued to each unit (the upper number), and the sustained throughput rate for the unit, expressed as one instruction every (lower number) of cycle(s). (FIFO buffering is not taken into account for these numbers.) For example, the Multiply (Mul) unit can accept/complete a multiply instruction every cycle but has a three cycle latency. The Divide (Div) unit is not fully pipelined and can only process one request at a time. As such, it has a minimum delay of two cycles if the divide is one of: overflow, divide by zero, or is reusing the result of a previous div/rem operation. If the div/rem instruction is not one of those special cases, the division algorithm starts at takes an additional 32 cycles. Details of the ALU, Branch unit (BR), branch predictor, Mul unit,

Control Status Registers (CSR), TLBs and MMUs are not presented here due to space limitations.

3.1. Pipeline Structure

In order to leverage the parallel execution units, Taiga features a highly decoupled design with fetch logic able to move independently of the issue logic and execution units. Additionally, Taiga's parallel execution units all operate independently of each other allowing for each unit to have its own optimized pipeline. As different execution units have different latencies, for strict commit ordering, an instruction queue tracks the order in which instructions are issued and controls the select logic for the writeback mux.

Details of the Fetch and Load Store Unit (LS unit) pipelines are presented in Figure 2. Both the fetch logic and LS Unit support multiple memory interfaces: BRAMs for local memory, caches for access to shared-memory and a peripheral bus for the LS Unit. Compared to fixed-pipeline designs, the FIFOs for the Fetch and LS Units allow for isolation of the cache control signals from the rest of the processor's pipeline. Additionally, the LS Unit input FIFO enables other instructions to be issued to other execution units when small bursts of load/store instructions are executed.

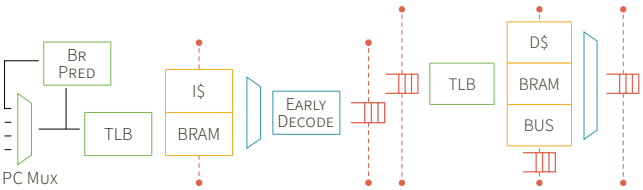
3.2. Execution Unit Interface

All execution units within Taiga share the same control signal interface with the issue logic and the writeback logic to facilitate the integration of new execution units. As the interface is fixed, the primary effort in integrating a new unit is in implementing the new decode logic to support the unit and any custom forwarding paths. Every unit has a *ready* and *new request* control pair for the issue logic and a *done* and *accepted* set of control signals for the writeback mux. These signals can be connected to FIFOs, in the case of the Load Store Unit, or to combinational logic in the case of the ALU, which does not register its output (effectively utilizing the writeback mux as forwarding logic). As long as an execution unit can not generate an exception, no additional control signals are required.

Most units share the same control logic for determining when source operands are ready. As such, the process of adding a new unit can begin by duplicating the logic for an existing unit. The Load Store Unit is an exception as it supports load-to-store forwarding internally, and thus, for its issue logic an instruction can be issued, when there is a valid load-to-store sequence, even when operands are not ready at the issue stage.

4. TAIGA RESOURCE USAGE AND FREQUENCY COMPARISON

In this section, we evaluate the resource usage and operating frequency of Taiga compared to other open source processors with similar capabilities on a zedboard (Xilinx Zynq X7CZ020 FPGA). We then further investigate critical paths within the processor and how they change under different configurations.



(a) Fetch Unit (b) Load Store Unit

Fig. 2. High-level Fetch Unit and Load Store Unit Pipelines and Components. (The dotted lines indicate boundaries between unit pipeline stages)

Table 1. Full System (Processor, Bus with UART, memory interface) Resource Usage Comparison

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

For comparison purposes against other processors (Rocket RV64IMAC and LEON3), the configuration is is constrained by what the Rocket processor supports, thus no scratch memory and a cache configuration supported by Rocket's virtually-accessed physically tagged caches. All systems were built with 8KB 2-way instruction and data caches with random replacement policies, MMU support, Mul/Div support and no local memory. In the case of the LEON3 and Taiga, the systems also have a UART on the data bus. All three systems are built for the zedboard and interface to memory through the ARM processor. A comparison of resource usage and frequency is presented in Table 1.

The Rocket processor is a 64-bit processor and is thus expected to be larger due to a wider data path, however, that cannot account for the four times size increase. Some of the additional area is due to more complex processor features, and some of it is due to the fact that some design aspects, such as their branch predictor do not map well to FPGAs. The LEON3 processor is the closest comparison for an open source 32-bit general purpose processor and is about 1.5 times the size of Taiga (in slices), while Taiga clocks approximately 39% faster, this despite Taiga featuring a more flexible pipeline design.

Table 2 presents three different system configurations for Taiga with only processor configuration options changing between the different versions. The first is a *minimal* configuration, where the multiply, divide, TLBs/MMUs, and caches (along with atomic operations) have been removed from the system. The second configuration is the *comparison* system used above in the LEON3 and Rocket compar-

Table 2. Taiga system configuration comparison

	LUTs	FFs	Slices	BRAMs	DSPs	Freq(MHz)
minimal	2,614	1,592	906	9	0	111
comparison	3,998	2,942	1,371	10	4	104
full	4,079	2,959	1,394	32	4	103

ison. The third configuration is a *full* configuration with a 4-way 16KB data cache and 32KB of local memory shared by the Fetch and Load Store Units.

A full system configuration is about 1.5 times larger than the minimal configuration. We have also built the system for an Xilinx Virtex UltraScale+ XCVU9P, where the maximum frequency for the processor is 254MHz. Additionally, Taiga can be built for Intel parts and supports the Avalon bus. It is also possible to build Taiga for older FPGAs that do not support LUT RAMs such as the Cyclone IV FPGA on the DE2-115 board. However, this will result in much higher resource usage as Taiga has been designed to leverage LUT RAMs extensively in its design. A system without caches built and tested on a DE2-115 board required 6,344 LEs and 4,519 FFs. This highlights how design considerations vary over FPGA generations.

In terms of the main functional blocks of Taiga, the Load Store Unit is the largest at about 20% of the slices of the design, the next largest block is the main decode and issue logic at about 17% of the design. The CSR unit constitutes about 11% of the design resources, the Fetch Unit about 9% and the divider about 9% as well. The register file, instruction flow control, and writeback mux constitute approximately 20% of the slices for the design, with the remainder split between the other execution units and the TLBs/MMUs.

In the current design of Taiga, on the (Xilinx Zynq X7CZ020, the maximum operating frequency is, depending on the configuration, within the range of 103MHz to 111MHz. In the *minimal* configuration, the critical path is the write-back data path. In the *comparison* system, it is also through the write-back data path and in *full*, the critical path is through the data TLB to the data tags. The *full* system has a larger more associative data cache than the *comparison* leading to higher fanout for the translated address signals. When building for the UltraScale+ part (*full*), the critical path becomes the tag hit logic between the tag and data stages. In this case, the critical path changes as the logic inside the TLBs has a shorter delay due to the longer carry chains in the UltraScale+ FPGAs.

5. CONCLUSIONS

In this paper, we presented Taiga, a 32-bit, RISC-V RV32IMA processor for general purpose soft-processor computing, heterogeneous systems research and soft-processor architectural research. We found that while Taiga features variable-latency parallel execution units, it has comparable or better frequency and resource usage compared to other similarly featured soft-processors.

6. ACKNOWLEDGEMENTS

The authors would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) (STP 380968-09, RGPIN 341516-2011), Xilinx Inc. and Altera Corp. for supplying resources and/or funding for this project.

7. REFERENCES

[1] *Nios II Gen2 Processor Reference Guide*, Altera Corp. [Online]. Available: altera.com/en_US/pdfs/literature /hb/nios2/n2cpu-nii5v1gen2.pdf

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

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

[4] *GRLIB IP Core User's Manual*, Cobham Gaisler AB. [Online]. Available: gaisler.com/products/grlib/grip.pdf

[5] H. Wong, V. Betz, and J. Rose, "Comparing fpga vs. custom cmos and the impact on processor microarchitecture," in *FPGA*, 2011, pp. 5–14.

[6] P. Yiannacouras *et al.*, "The microarchitecture of fpga-based soft processors," in *CASES*, 2005, pp. 202–212.

[7] G. Schelle *et al.*, "Intel nehalem processor core made fpga synthesizable," in *FPGA*, 2010, pp. 3–12.

[8] 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.

[9] M. Rosire, J. l. Desbarbieux, N. Drach, and F. Wajsbrt, "An out-of-order superscalar processor on fpga: The reorder buffer design," in *DATE*, March 2012, pp. 1549–1554.

[10] H. Wong, V. Betz, and J. Rose, "High performance instruction scheduling circuits for out-of-order soft processors," in *FCCM*, May 2016, pp. 9–16.

[11] K. Aasaraai and A. Moshovos, "Sprex: A soft processor with runahead execution," in *2012 International Conference on Reconfigurable Computing and FPGAs*, Dec 2012, pp. 1–7.

[12] Z. A. Ye *et al.*, "Chimaera: a high-performance architecture with a tightly-coupled reconfigurable functional unit," in *ISCA*. ACM Press, 2000, pp. 225–235.

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

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

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