BYOC: A "Bring Your Own Core" Framework for Heterogeneous-ISA Research

Jonathan Balkind
jbalkind@princeton.edu
Princeton University, USA

Katie Lim
katielim@cs.washington.edu
University of Washington, USA

Michael Schaffner
schaffner@iis.ee.ethz.ch
ETH Zürich, Switzerland

Fei Gao
feig@princeton.edu
Princeton University, USA

Grigory Chirkov
gchirkov@princeton.edu
Princeton University, USA

Ang Li
angl@princeton.edu
Princeton University, USA

Alexey Lavrov
alavrov@princeton.edu
Princeton University, USA

Tri M. Nguyen
tri_nguyen@hms.harvard.edu
Harvard Medical School, USA

Yaosheng Fu
yfu@nvidia.com
NVIDIA, USA

Florian Zaruba
zarubaf@iis.ee.ethz.ch
ETH Zürich, Switzerland

Kunal Gulati
f20150353h@alumni.bits-pilani.ac.in
BITS Pilani, India

Luca Benini
lbenini@iis.ee.ethz.ch
ETH Zürich, Switzerland
Università di Bologna, Italy

David Wentzlaff
wentzlaf@princeton.edu
Princeton University, USA

Abstract

Heterogeneous architectures and heterogeneous-ISA designs are growing areas of computer architecture and system software research. Unfortunately, this line of research is significantly hindered by the lack of experimental systems and modifiable hardware frameworks. This work proposes BYOC, a "Bring Your Own Core" framework that is specifically designed to enable heterogeneous-ISA and heterogeneous system research. BYOC is an open-source hardware framework that provides a scalable cache coherence system, that includes out-of-the-box support for four different ISAs (RISC-V 32-bit, RISC-V 64-bit, x86, and SPARCv9) and has been connected to ten different cores. The framework also supports multiple loosely coupled accelerators and is a fully working system supporting SMP Linux. The Transaction-Response Interface (TRI) introduced with BYOC has been specifically designed to make it easy to add in new cores with new ISAs and memory interfaces. This work demonstrates multiple multi-ISA designs running on FPGA and characterises the communication costs. This work describes many of the architectural design trade-offs for building such a flexible system. BYOC is well suited to be the premiere platform for heterogeneous-ISA architecture, system software, and compiler research.

CCS Concepts. • **Computer systems organization → Multicore architectures**; *Interconnection architectures*; **Heterogeneous (hybrid) systems**; • **Networks** → *Network on chip.*

Keywords. heterogeneous-ISA; manycore; research platform; architecture; open source; RISC-V; x86; SPARC

ACM Reference Format:
Jonathan Balkind, Katie Lim, Michael Schaffner, Fei Gao, Grigory Chirkov, Ang Li, Alexey Lavrov, Tri M. Nguyen, Yaosheng Fu, Florian Zaruba, Kunal Gulati, Luca Benini, and David Wentzlaff. 2020. BYOC: A "Bring Your Own Core" Framework for Heterogeneous-ISA Research. In *Proceedings of Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20).* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3373376.3378479

ASPLOS'20, March 16–20, 2020, Lausanne, Switzerland

ACM ISBN ISBN 978-1-4503-7102-5/20/03…$15.00
https://doi.org/10.1145/3373376.3378479

1 Introduction

Heterogenous systems and processors are becoming ubiquitous [18, 22–25, 29, 31, 54]. Driven by the need to gain further efficiencies at the end of Moore's Law [38], building chips with mixtures of different cores, accelerators, GPUs, GPGPUs, TPUs, microarchitectures, and potentially even

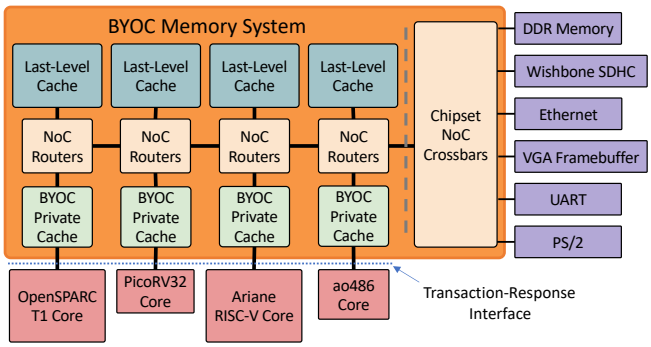

Figure 1. A BYOC system connecting supported cores and platform-agnostic peripherals.

different ISAs is becoming the standard. One example of this heterogeneity can be seen in cell phone processors; for instance, the latest Apple A12 processor contains two different processor cores and 42 accelerators [19, 48].

It has become common for heterogeneous processors to integrate devices such as GPUs, TPUs, and other accelerators, all working under the direction of a set of general-purpose cores. In the world of general-purpose compute, processor cores have emerged as reusable Intellectual Property (IP) blocks, with different, ISA-dependent features such as compact code density, low power operation, security extensions, high-throughput vector processing, and/or number of architectural registers. As a result of this, heterogeneous-ISA processors are emerging both as products [20] and as a hot topic of active research [16, 46, 52–54].

While heterogeneous-ISA processors provide significant opportunities for optimisation, exploiting the unique features of their distinct ISAs (and microarchitectures) to improve energy efficiency, performance, or security, there exist many interesting system-level challenges that need to be explored and solved with such systems. In particular, there is difficulty integrating cores of different ISAs, there is significant, ISA-specific, platform-level baggage that is needed to run existing operating systems and applications, and many of the existing platforms are so loosely coupled that they look more like traditional distributed systems than highly optimised heterogeneous-ISA systems.

Ideally, connecting cores of different ISAs to build a heterogeneous-ISA processor would not require significant architectural changes, with the ISA acting as "just another interface". However, existing research on heterogeneous-ISA systems has been hindered for multiple reasons. First, the direct comparison of cores of different ISAs requires a system in which those cores can be integrated. There are no existing hardware platforms which support this level of pluggability and therefore, prior work has had to rely on high level simulations. A framework to compare and characterise cores of different ISAs would require a simple, standard interface for connecting new cores, independent of their architectural or microarchitectural details. It should also enable all cores to connect as equals in order to enable apples-to-apples comparisons across differing design points.

Second, existing systems which feature cores of multiple ISAs typically do not offer hardware-coherent shared memory, instead they rely on slower and less convenient core-to-core communication mechanisms. Such platforms only offer either message passing [11] for communication or provide software-enabled coherence [29] which both incur significantly higher communication overheads. Next, existing heterogeneous-ISA research systems require significant runtime support for their secondary cores [24], or they limit the type of threads that can execute on them [29], treating the general-purpose cores more like accelerators which need bespoke application support. Finally, previously built systems frequently are not open source or feature significant components which are closed. This limits the opportunity for researchers across the stack to make modifications to facilitate research such as changes to improve performance, energy efficiency, or security.

In this paper, we present BYOC (shown in Figure 1), a cache-coherent, manycore, research framework built to enable heterogeneous-ISA research which can be simulated, realised on FPGA (at ~100MHz), or taped out in silicon (at >1GHz). BYOC is designed with a "Bring Your Own Core" architecture which supports the connection of cores with different ISAs and microarchitectures, and the integration with specialised hardware accelerators and domain specific accelerators. In addition, **BYOC is fully open source**[1] (hardware, software, and firmware) and is built using industry standard hardware description languages (Verilog and SystemVerilog).

To enable this "Bring Your Own Core" architecture, we developed a new cache interface termed the "Transaction-Response Interface" (TRI) in conjunction with a local "BYOC Private Cache" (BPC) that can work as a private L1 or L2 cache. TRI is lightweight, handshaked, and supports multiple outstanding memory transactions per core. With the cache coherence protocol implemented in the BPC and last-level cache (LLC), the core is decoupled from most details of the cache coherence protocol. This makes connecting a new core to BYOC a simple matter of implementing a transducer from the core to TRI, regardless of the core's ISA.

Our design of BYOC extends the OpenPiton manycore research platform [7], which used only the SPARC V9 ISA. In the process of developing the TRI and adding support for RISC-V and x86 cores in BYOC, we developed a number of world's first prototypes. These include JuxtaPiton (or Oxy) [27, 28], the world's first open-source, general-purpose, heterogeneous-ISA processor, and OpenPiton+Ariane [8], the first open-source, SMP Linux-booting RISC-V manycore. In this paper, we characterise both Oxy and a prototype system known as Dicho which has 64 bit SPARC V9

[1]GitHub repository: https://github.com/bring-your-own-core/byoc

(OpenSPARC T1) and 64 bit RISC-V (Ariane) cores both booting Linux and coherently sharing memory. These prototypes operate at high speed on FPGA to enable at-speed operating system (OS) and system software research.

We believe we have identified a pluggable, scalable, and performant design point for the MESI-based cache coherence protocol and the interface from the cores to BYOC. So far we have been able to connect **ten different cores** of varying microarchitectures and **four different ISAs** to BYOC via TRI transducers and support the mixing and matching of cores to build heterogeneous-ISA processors.

Our contributions include:

- The creation of an open-source, cache-coherent, memory system and framework explicitly designed to support cores of multiple general-purpose ISAs.
- The design of the Transaction-Response Interface (TRI) which enables taking a core with a minimal coherence interface and turning it into a manycore processor.
- The BYOC framework which supports both heterogeneous-ISA designs and heterogeneous architectures with accelerators and domain specific processors.
- A detailed evaluation of inter-core communication cost using the BYOC framework running on an FPGA.
- The first open-source, general-purpose, heterogeneous-ISA processor.
- The first open-source, SMP Linux-booting RISC-V manycore.

2 Background

2.1 Heterogeneous Architectures

Multicore and manycore systems can exhibit varying degrees of heterogeneity. The most common design point is heterogeneous microarchitecture, homogeneous ISA. Previous research [22, 23] showed performance and energy efficiency benefits by scheduling applications on cores that best match applications' demands. Due to its energy efficiency benefits, this architecture has become common in mobile processors, such as ARM big.LITTLE [5] and Apple's A12[19].

Another design point is heterogeneous microarchitecture, overlapping ISA where cores share some instructions but not others. This design is seen in work by Venkat et al. [52], which constructs an ISA based on x86 and then does a simulation-based design space exploration of systems with cores implementing subsets of their extended x86 ISA.

A third design point in this space is heterogeneous microarchitecture and disjoint ISAs where cores in the system do not have any common instructions. Venkat et al. [54] went on to explore this design point with ARM Thumb, x86, and Alpha cores in design space exploration. Due to the lack of BYOC, they had to rely on gem5 and McPAT.

In both the overlapping ISA and disjoint ISA works, they do not run an OS or fully model their memory system in evaluation. They do find performance and energy efficiency benefits over heterogeneous microarchitecture alone.

This prior work shows the wide range of design points possible in heterogeneous systems research, so any framework needs to be flexible enough to support all of these configurations. Additionally, prior work is either closed-source IP, which is not easily extensible, or high-level simulation, which does not support full-stack systems. BYOC is able to support these different levels of heterogeneity with all types of cores connecting as peers, enabling apples-to-apples comparisons for different hardware configuration. It is also completely open-source and is a hardware implementation capable of supporting full-stack research.

2.2 Heterogeneous-ISA Software Support

Heterogeneous-ISA architectures comes with challenges for compilers, runtimes, and operating systems. DeVuyst et al. [16] built a compiler and runtime support for dynamic migration between cores in heterogeneous-ISA systems. They were able to migrate binaries during runtime between cores, limiting total performance loss to under 5% even when migrating every few hundred milliseconds. A key to achieving this performance, however, was the availability of hardware shared memory, so they performed their experiments in simulation.

Prodromou et al. [46] tackle heterogeneous-ISA scheduling. They apply machine learning techniques to predict the ISA affinity of different application regions.

The researchers behind Popcorn Linux have also explored building a compiler that enables runtime migration as well as OS support for heterogeneous-ISA systems [11, 31]. They used their multikernel model to investigate a potential OS design for a heterogeneous-ISA system by compiling a copy of their kernel for each ISA. For evaluations, they used a hardware x86-ARM system where the cores were connected over PCIe. Because the system did not have hardware shared memory, the migration of binaries during execution was expensive due to the overhead of copying state.

Lin et al. built the K2 OS [29], an OS which assumes multiple coherence domains where cores in different domains do not have coherent memory. Their hardware has cores in the different domains of different ISAs. Using modified Linux kernels, they run a main kernel in one domain and a shadow kernel in another and replicate state between them using software coherence, which incurs overhead.

One of the main takeaways from prior work is that software systems running on heterogeneous-ISA architectures rely on shared memory to achieve good performance. Thus, it is crucial that any framework targeting heterogeneous-ISA systems is able to provide shared memory.

2.3 Heterogeneous-ISA Platforms

Some heterogeneous-ISA platforms are coming into being, but they are typically built in such a way that little cores are connected as "guests" in a big core's environment, rather

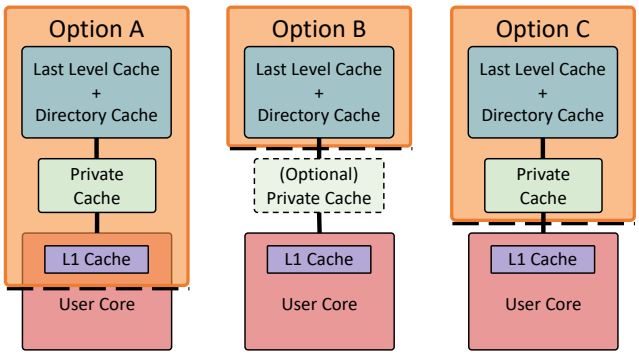

Figure 2. Three options for providing a core interface from a cache-coherent memory system. The thick dashed line indicates the interface point.

than being a first-class citizen. Most commonly this means there is a lack of coherent shared memory, no caches for the little cores, or the little cores do not feature full, OS-capable platform support. HERO from the PULP Platform [24] and the OMAP4 used as the platform for the K2 OS [20, 29] are some such examples. Other examples of heterogeneous-ISA platforms include lowRISC [12] and Celerity [15] which pair RV64 cores with RV32 cores.

2.4 Heterogeneous-ISA Security

Varying the ISA of a system can provide potential security improvements. Instruction set randomisation has the goal of making different cores appear to have different and secret instruction sets, emulating a heterogeneous-ISA system, so attackers must craft a custom exploit for each machine. Barrantes et al. [9] use dynamic binary translation to prevent code injection attacks by randomising code at load time and then unscrambling instructions as the translator runs them. Sinha et al. [50] build this randomisation into a hardware prototype to prevent code reuse attacks.

HIPStR [53] leverages a hardware heterogeneous-ISA setting to defend against code reuse attacks by randomly migrating execution of a binary between x86 and ARM cores. This forces any potential attackers to craft heterogeneous-ISA ROP attacks, which are extremely hard to execute. They still maintain performance benefits of heterogeneous-ISA systems due to the low overhead of migration.

3 Choosing an Interface Point

One challenge in building a cache-coherent framework designed to connect arbitrary cores is where to interface. Attaching new cores should be straightforward, and internal network-on-chip (NoC) and coherence protocol details should be sufficiently hidden. BYOC is a tile-based manycore by default, so the inclusion of a distributed, shared LLC is our first assumption, as it ensures scalability with core count. Next, we assume at least one level of local, private cache (which should be write-back to reduce traffic to the LLC). With these assumptions, our options (shown in Figure 2) are:

A The framework provides the L1 cache, L2 private cache (if included), and LLC. The user connects with the framework at an L1 cache interface.

B The framework provides LLC only. The user rewrites their L1 and/or private L2 for coherence and connects at the NoC or LLC cache interface.

C The framework provides the private cache and LLC. The user connects the L1 to the private cache interface.

Option A. The framework provides all cache levels (either two or three), thereby decoupling the user's core from the coherence protocol. The L1 cache must be designed to be suitable for integration into cores with a wide variety of microarchitectures and ISAs with low overhead, while maintaining the user's core's performance. The framework must also implement the parts of the coherence protocol that must be handled in the private cache.

One failing of this design point is that L1 caches are typically tightly interwoven with the processor instruction front-end and load-store unit (LSU), and hence the implementation can vary significantly between microarchitectures. One complication for Option A is the tagging and indexing strategy for the supplied L1 cache. There is a lot of variance based on microarchitecture as to whether a core's L1 caches will be virtually or physically indexed and virtually or physically tagged. An L1 supplied with the framework must support all these modes and thus must also interact with the core's TLBs. This renders Option A a difficult and impractical choice.

Option B. Option B requires the user of the framework to understand many details of the coherence protocol, including atomic operations and how instructions and data are kept coherent. It may also require heavy modifications to the last private cache level. Cache coherence protocols are inherently complex. The OpenPiton P-Mesh cache coherence system [7, 45] that we base our work on handles around 30 different message types. Writing compatible private caches would introduce both significant design and validation effort. While the designer gains control over architectural details at the core, it may lead to community fragmentation in how to rewrite the L1 (and L2) caches.

Option C. Given the downsides of Options A and B, we chose Option C. Like Option A, it can decouple the core and L1 cache from the coherence protocol. This requires a carefully written and highly configurable interface to the framework's local, private cache. However, unlike Option A, the framework does not need to deeply understand the core's microarchitecture. Given that the BYOC framework-provided local, private cache is primarily designed to be used as a second layer of cache beyond the L1, it also does not have the L1's, tight, single-cycle turnaround performance requirements.

Option C also wins over Option B as the framework provides a canonical implementation for the local, private cache's

Core to BPC	BPC to Core	Subset
Load	Load response	Load/store
Store	Store acknowledgement	Load/store
Instruction miss	Instruction return	Instruction
	Invalidation request	Invalidation
Atomic	Atomic response	Atomic
Outbound Interrupt	Inbound Interrupt	Interrupt

Table 1. TRI message types sent to and from the BPC.

part of the coherence protocol. The user can benefit from this validation and use the framework without having to understand the coherence protocol – provided that the interface to the local, private cache is sufficiently decoupled. If a user wishes to modify the local, private cache, it is part of the framework and so they may do so.

4 BYOC Architecture

BYOC is a cache-coherent manycore framework designed to support the connection of cores of a variety of ISAs and microarchitectures. To decouple cores from the cache-coherence protocol, BYOC provides a local, private cache known as the "BYOC Private Cache" (BPC) and a distributed, shared last-level cache (LLC) which implement the coherence protocol. New cores (including the four cores detailed in Table 3) connect their write-through L1 caches to the BPC using our new "Transaction-Response Interface" (TRI). The TRI handles load, store, and instruction requests from the core, invalidations to the core, and offloads atomic requests from the core to the BPC or LLC as applicable. BYOC also provides a chipset for the connection of peripherals and non-coherent accelerators, which is automatically generated based on an XML description of the system.

4.1 Transaction-Response Interface

Having chosen option C, we needed to specify a suitable interface to the BPC. Our Transaction-Response Interface (TRI) evolved over several years in which we integrated several different cores with the BYOC framework. Those cores serve both as useful tests and extensions of BYOC and as reference implementations of TRI transducers to the BPC. The goal is to make it possible to take any single-core system and turn it into a cache coherent multi- or manycore design with a rich collection of peripherals and platform support while minimising the effort required for the end user.

TRI has two parts: one from Core-to-BPC and one from BPC-to-Core. Both directions are valid-then-ready handshaked [51]. In the Core-to-BPC direction, a request provides a message type (shown in Table 1), a naturally-aligned physical address, the request size, its cacheability, the L1 cache way to replace (if any), and the data (for stores or atomics). As described in Section 4.2, the core may specify a "memory thread ID", used for hardware threads or as MSHRs entry IDs for a core to support more outstanding memory transactions.

In the BPC-to-Core direction, primarily used for responses, a request has a message type, the L1 ways to be invalidated, the request's cacheability, the memory thread ID, and data.

4.1.1 TRI Functionality Subsets. TRI is designed to be modular, offering subsets of the functionality depending on the chosen core's needs. For instance, a core can implement the load/store subset comprising the load and store messages, enabling swift system integration and single-core bring up. If the core has no L1 cache, then it is already coherent and ready to be a multicore.

For full coherence support with L1 caches, the invalidation subset is also needed. This includes providing the L1 cache way allocated for the load or store request, which the BPC tracks. TRI invalidations come to the L1 caches including the way to be invalidated. With write-through L1 caches there is no write-back of dirty data which keeps the invalidation interface simple.

To provide efficient synchronisation, the core can also implement the atomic memory operation subset. For cores without L1 caches, this is simply another message type. For cores with L1s, the atomic subset requires invalidations.

4.1.2 Write-through L1 Caches. If a core has L1 caches, it must write-through all memory operations to the BPC so that the BYOC LLC can track the cacheline state in its directory. This requirement was a design decision made to make TRI a simpler to use interface. Write-back caches would need an additional data flow in response to invalidations from the BPC to the L1 data cache for dirty lines.

We have found it to be relatively straightforward to modify a write-back L1 data cache to be write-through, while also reducing the complexity of the L1 cache subsystem and thus the likelihood of bugs. However, to achieve higher performance by supporting multiple outstanding memory transactions, some modifications are needed, like the addition of a merging write buffer mentioned in Section 5.3. Proper care must be taken that the L1 miss handling and write buffer adhere to the memory consistency model of the specific ISA.

While the L1 is write-through, the BPC remains write-back. This minimises NoC traffic between the BPC and LLC.

4.2 BYOC Private Cache

The BPC serves dual purpose as both a cache (configurable in size and associativity) and a transducer to bridge the predefined, potentially non-customisable L1s to BYOC's cache coherence protocol. To increase manycore scalability, the BPC maintains optional support for features such as Coherence Domain Restriction (CDR) for software-configurable coherence domains and cache line placement [17] and a Memory Inter-arrival Time Traffic Shaper (MITTS) for high-performance and fair memory traffic shaping [58], both of which come as part of OpenPiton.

To maintain cache coherence with the optional L1 cache, the BPC includes a "way-map table" (WMT) as a cost-efficient

alternative to a shadow tag table to track the L1 data cache. The WMT does not store address tags, but rather the ways of cache lines allocated in the L1. The BPC can invalidate the L1 data cache without having to model its eviction policy. The WMT also decouples the size and associativity of the two caches, though the BPC must be inclusive of the L1.

The LLC maintains code-data coherence (e.g. for self-modifying code), with the BPC bypassing all instruction cache fills to the LLC. This increases space-storage efficiency of the chip, but, in principle, the BPC could be a unified cache. At the moment, instruction cache lines are 32 bytes, compared with 16 byte data cache lines. This is required to support the OpenSPARC T1 core, which has a fixed line size. Other cores have similar, specific restrictions, which makes supporting configurable line sizes a priority, though modern cores typically make cache line size parameterisable. Making BPC line size configurable is a work-in-progress.

4.2.1 Memory Consistency. The BPC can handle up to four concurrent memory threads. For a single memory thread, transactions conform to the total store order (TSO) memory model, but BYOC supports cores that are TSO or weaker. With the OpenSPARC T1 core, each CPU thread is bound to one BPC memory thread, to satisfy TSO. Across memory threads, no ordering is enforced to maximise memory parallelism and performance. Multiple BPC memory threads can be used by a single core to increase single-threaded performance. With the Ariane core, each memory thread can serve one outstanding MSHR for load and store transactions that do not have ordering requirements. This works for weaker consistency models (like RISC-V) to increase performance.

4.3 Shared Last-Level Cache

The LLC is configurable in size and associativity and is sharded by design, meaning that each slice is allocated a portion of the address space and is unaware of the other shards (LLCs do not communicate). The BPC performs the allocation of homes for lines. Each LLC has a slice of the directory which (in sum) tracks the coherence states of all cache lines in the memory system. The LLC interacts with the BPC over the NoC to implement the MESI cache coherence protocol. For a simple multicore processor, the LLC can be reduced to fewer shards, instead of scaling with core count. In that case, only the home allocation unit in the BPC needs modified to home data in the few available shards.

4.4 Atomic Operations

Atomic memory operations are essential to implement software synchronisation mechanisms in multi- and manycore systems. Different ISAs require different flavours of atomic operations to be supported by the memory system. Two widely used variants are load-reserved/store-conditional (LR/SC), and "compare-and-swaps" (CAS). BYOC supports **both flavours**, in addition to a rich set of fetch-and-op operations such as atomic arithmetic or logic operations. These fetch-and-ops could be emulated with LR/SC or CAS operations, but supporting a small set of integer and logical ALU operations makes for faster execution. The supported atomic operations are shown in Table 2 and described below.

Atomic Operation	Fetch-and-Logical	Fetch-and-Arithmetic
Compare-and-swap	AND	Add
Swap	OR	Minimum (signed) Minimum (unsigned)
Load Reserved / Store Conditional	XOR	Maximum (signed) Maximum (unsigned)

Table 2. The atomic operation types supported in BYOC.

Atomic Performance. Compare-and-swap, swap, and fetch-and-op atomic operations are implemented in the LLC. For highly contended lines (particularly lines not being concurrently used for non-atomic access), implementing the atomic operations in the LLC provides higher throughput than performing them in the modified state of the BPC. This is because consecutive operations do not require invalidations to BPCs and also do not require a round-trip from the BPC for acquisition of the line into the modified state for each operation.

4.4.1 Support for SPARC Atomic Operations. The SPARC V9 instruction set includes three atomic operations: compare-and-swap (CAS), swap, and load-store unsigned byte (LDSTUB). These are implemented in the LLC as two atomic operations, with LDSTUB a subset of swap. These are issued via TRI as non-cacheable operations. The LLC invalidates all sharers before performing the atomic operation.

4.4.2 Support for RISC-V Atomic Operations. The RISC-V ISA relies on LR/SC-type atomics and fetch-and-op atomics. We leveraged the same LLC datapath used for CAS and swap to implement fetch-and-op by adding a small ALU that performs all of the RISC-V fetch-and-op atomics. The logical operations are AND, OR, and XOR. The arithmetic operations are add, minimum, unsigned minimum, maximum, and unsigned maximum. The old value read from the LLC is returned to the core after the operation is performed.

Load-reserved/store-conditional (LR/SC) is handled within the BPC. The BPC has a flag to indicate the status of the most recent LR/SC. After receiving an LR over the TRI, the cache requests an upgrade for the line to the “M” MESI state and sets the LR/SC flag to high. From then, any operation that changes the line’s MESI state will clear the LR/SC flag (e.g. a load from another core which downgrades the MESI state to “S”). The later SC returns 0 (success) only when the LR/SC flag is still high, otherwise the store fails and 1 is returned.

4.4.3 Support for x86 Atomic Operations. The i486 ISA implemented by the ao486 core supports CMPXCHG

and XADD atomics, analogous to CAS and fetch-and-add respectively. Since both of these are already implemented for SPARC and RISC-V, we can reuse them. The ao486 core needs to be modified to send the atomics over TRI and support invalidating the affected L1 cache line.

4.4.4 Support for Unaligned Atomic Operations. Some ISAs (including x86) support unaligned atomic operations. BYOC does not currently support these, but the LLC and BPC can be straightforwardly modified to support unaligned atomics within a cache line. To support the operations across cache lines, the BPC can be modified to include a small state machine and ALU which will bring both lines into the modified state and then perform the atomic operation across the two lines. Ensuring that both lines are the M state simultaneously has similar forward progress challenges to LR/SC, as described in Section 5.3.2.

4.5 Virtual Memory

The BYOC memory system operates on physical memory addresses. This means the system designer, when choosing which cores and accelerators to connect, must choose how to build their cross-ISA virtual memory configuration. In our Oxy prototype, the PicoRV32 core accesses a 4GB region of physical memory addresses directly, but is directed by the OST1 core on where the application is located in memory. For our Dicho prototype, the OpenSPARC T1 and Ariane cores use independent page tables managed by their independent Linux kernels and negotiate their shared region using a fixed memory location which is updated with a pointer to the region. Both cores `mmap` the shared region to allocate their independent virtual memory mappings in their page tables. In a more heterogeneous-ISA-friendly OS environment, the designer could enable common virtual memory mappings without modification to the existing BYOC hardware. Cores can use BYOC's (instruction-triggerable) inter-processor interrupts to request TLB invalidations and make cross-calls to other cores (as is required by many ISAs). Accelerators in the system are expected to use IOMMUs for translation to BYOC physical addresses.

4.6 Networks on Chip

BYOC inherits its networks on chip (NoCs) and NoC ordering requirements from OpenPiton [7]. BYOC defaults to a 2D mesh topology for its three networks on chip, though a crossbar is also provided as an alternative. The network can be replaced with other networks provided they maintain per network point-to-point ordering.

4.7 Chipset

NoC traffic sent off-chip is routed to the chipset where "chipset crossbars" connect all of the peripherals. The chipset crossbars are generated from an XML description of the system using the PyHP preprocessor provided as part of OpenPiton. To add a new peripheral or accelerator, the user simply adds a device in the system XML description and the necessary Verilog code is automatically generated. The XML device description includes the memory range assigned to the device and whether the device will generate memory requests. A packet filter is attached to any device which generates its own memory requests. The packet filter reads the requested address and rewrites the packet's destination ID according to the XML description. This way, the devices do not need to be aware of the system memory map.

4.7.1 Peripherals. A number of platform-agnostic peripherals are supported in the system. These are usually connected via standard interfaces like AXI4-Lite or in the case of the SDHC controller, Wishbone. The supported peripherals include an SDHC controller, UART, Ethernet MAC, VGA framebuffer, and PS/2 for keyboard input. The OpenPiton platform provides an interface from the NoC to the Xilinx memory controller. We have added an AXI4 bridge to provide a more generic interface to other memory controllers.

4.7.2 Accelerators. TRI makes it possible to coherently connect accelerators to some of the system's tiles, while the chipset crossbar enables incoherent connections. These can be useful to build heterogeneous systems with a mix of general purpose cores and dedicated accelerators for specialised tasks. Below we describe our experience with adding the non-coherent accelerators MIAOW [6] and NVDLA [39].

MIAOW GPGPU. MIAOW [6] is an open-source GPGPU implementing the AMD Southern Islands ISA for architectural research and experimentation. MIAOW on FPGA was tethered to a Microblaze microcontroller (on the FPGA) and a host (connected to the FPGA). For a student project to integrate non-coherent accelerators into BYOC, MIAOW was connected to the chipset via AXI-Lite and programmed from the OpenSPARC T1 core. MIAOW was then modified to natively send memory requests on the NoC to increase performance and better reflect everyday CPU+GPU systems.

NVDLA. We have also connected the NVIDIA Deep Learning Accelerator (NVDLA) [39] to the system similarly to MIAOW. The configuration interface for NVDLA is transduced to AXI-Lite, and its interrupt output is wired to the platform's interrupt controller. NVDLA's AXI memory requests are translated to corresponding P-Mesh packets.

5 Case Studies

5.1 OpenSPARC T1

We started with the OpenSPARC T1 (OST1) core [35, 41], which is the original core provided with OpenPiton. This core implements the (big endian) 64 bit SPARC V9 instruction set and supports the swap and CAS SPARC atomics.

Core	OpenSPARC T1	PicoRV32	Ariane	ao486
ISA	64 bit SPARC V9	32 bit RISC-V	64 bit RISC-V	32 bit x86
Physical address	40 bit	32 bit	40 bit	32 bit
L1 caches	✓	✗	✓	✓
Virtual memory	✓	✗	✓	✓
Endianness	Big endian	Little endian	Little endian	Little endian
Atomic operations	Compare-and-swap, swap	Fetch-and-op	Fetch-and-op, LR/SC	Compare-and-swap, Fetch-and-add
OS Capable	✓	✗	✓	*
Unique feature	1/2/4 threads per core	Instructions cached as data	Multiple outstanding memory transactions	Unaligned accesses (realigned in transducer)

Table 3. Cores Integrated with BYOC (*OS capable core. OS support in BYOC in progress)

The core has virtual memory with 40 bit physical addresses and features write-through L1 instruction and data caches. Uniquely among the cores we have integrated with TRI, the OpenSPARC T1 supports multiple hardware threads per core, which requires a thread ID to be included in each TRI transaction.

The OpenSPARC T1 chip environment used the core-cache crossbar (CCX) interface [35] to connect cores, L2 cache, and FPU. The interface is somewhat coupled to core implementation details, such as L1 cache size, associativity, and line size. CCX was also used by OpenPiton [7], which resulted in CCX details becoming embedded in OpenPiton L1.5 cache. This made the path to building from OpenPiton and adding new cores unclear in the beginning. However, through adding new cores and developing the TRI, we were able to remove some CCX couplings. Importantly with TRI the core is decoupled from the cache coherence protocol.

Platform Details. The SPARC V9 ISA separates memory into two regions: I/O and memory. Memory addresses have a most-significant bit of 0 and are cached as normal. I/O addresses have a most-significant bit of 1 and are non-cacheable. Specific memory accesses or pages can be marked as (non-) cacheable which is indicated to the cache system. Specific memory accesses or pages can also be marked as little endian, though this is handled in the core.

SPARC uses the Open Firmware Device Tree to describe the system hardware to the software. The SPARC hypervisor reads the device tree description from the PROM and uses it to allocate memory, interrupts, etc. It filters the device tree and passes it to the guests' bootloaders and operating systems which will use it to discover hardware components and initialise the appropriate drivers.

Device interrupts connect to an interrupt controller which sends NoC interrupt packets to signal the core. In SPARC, interrupts (both device and inter-processor) are first handled by the hypervisor which passes them to the corresponding guest via an ISA-specified queue in memory.

5.2 PicoRV32

As the first step to add new cores of different ISAs and thus build TRI, we integrated PicoRV32 [55] to develop JuxtaPiton (or Oxy), the world's first open-source, general-purpose heterogeneous-ISA processor, which consists of OpenSPARC T1 and PicoRV32 cores [27, 28].

PicoRV32 is a multicycle RISC-V RV32IMC core and is little-endian. It does not have an L1 cache or an MMU for virtual memory. It also does not implement any of the RISC-V privilege modes and has a bespoke approach to interrupt handling. In its first iteration for PicoRV32, TRI only needed to support loads and stores. Once we developed TRI and BYOC further, we added (fetch-and-op) atomics to PicoRV32.

As PicoRV32 does not have an L1 cache, the BPC acts as PicoRV32's L1 cache. This is enabled by the fact that the TRI supports cacheable accesses of different sizes. The PicoRV32 core can directly write out just the modified data rather than a whole cache line, simplifying integration. Furthermore, the BPC acts as a combined instruction and data cache for PicoRV32, which issues all requests as data requests. Now integrated with TRI, PicoRV32 can access peripherals and send IPIs just like the OpenSPARC T1 cores in the topology.

Oxy is used as a platform to explore system software impacts of heterogeneous-ISA systems. We developed a proxy kernel that offloads user-mode RISC-V binaries from Linux run on the OpenSPARC T1 core to the PicoRV32 core. The PicoRV32 core can then make syscalls which are proxied by the OpenSPARC T1 core. We describe this system in more detail and evaluate it in Section 6.1.

Platform Details. As PicoRV32 is hosted in Oxy, platform support was limited. Oxy has both big-endian OpenSPARC T1 cores and little-endian PicoRV32 cores. PicoRV32's TRI transducer flips transactions bytewise to preserve a little endian data layout in memory.

We sign extend the core's 32 bit addresses and use the most significant bit to indicate cached or uncached operations. This enables access to peripherals from the PicoRV32 core without the need for hosting by the OpenSPARC T1 core. As the core is hosted, it is awoken to start executing by an IPI NoC packet sent from the OpenSPARC T1 core. Once it completes execution it returns to its reset/idle state.

5.3 Ariane

Ariane [57] is a 64 bit, single-issue, six-stage, in-order RISC-V core (RV64GC). It has support for hardware multiply/divide, atomic memory operations, and an IEEE compliant FPU. Moreover, it has support for the compressed and privileged instruction set extensions. It implements the 39 bit virtual memory scheme SV39 and boots Linux single-core on FPGA.

Ariane is little-endian and its TRI transducer flips endianness similarly to Oxy to ensure interoperability with existing platform peripherals. Sub-word flips are not necessary as Ariane uses 64 bit words like the OpenSPARC T1.

Connecting Ariane to BYOC took less than six months, despite the complexity of the core and the final goal of booting SMP Linux. It is worth noting that once the write-through L1 cache and TRI support were implemented in Ariane, **we successfully booted SMP Linux on two Ariane cores the day after first booting single core Linux**, and booted four cores one week later with no RTL changes needed in between. We attribute this success to the thorough validation of the original infrastructure provided with OpenPiton and the well defined TRI interface that we established for BYOC.

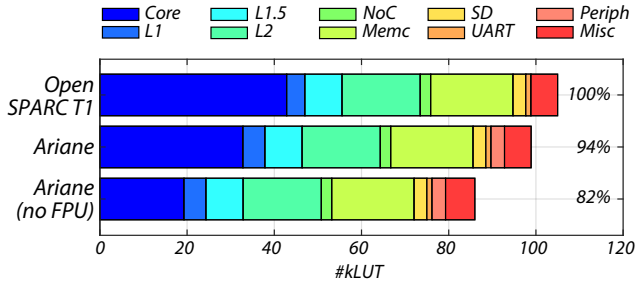

Figure 3. LUT distribution of single-core builds for a Xilinx Kintex 7k325tffg900-2 FPGA. The percentages are normalised with respect to the OpenSPARC T1's total size.

The LUT distribution for single-core Genesys2 FPGA builds is shown in Figure 3. The core uses around 22%-41% of the total resources, depending on the configuration (Ariane with or without FPU, OpenSPARC T1 with FPU). The T1 is around 23% and 93% larger than Ariane with and without FPU, respectively. This can be attributed in part to the T1's large register windows and reliability features. On FPGA, we synthesise the T1 only with one hardware thread, thus there is no hardware multithreading overhead.

5.3.1 L1 Cache Subsystem. Ariane's L1 caches are parameterisable in size, line size, and associativity. Ariane originally used a write-back L1 cache which was prone to subtle bugs. Moving to write-through and supporting TRI made for a simpler L1 cache subsystem. The L1 data cache features a merging write-buffer with forwarding capability to improve write performance. This write-buffer by default is eight, 64 bit words deep and supports two outstanding writes to the BPC by using multiple TRI memory thread IDs. The memory thread ID field's current size of two matches the T1's two hardware threads, limiting Ariane to two outstanding sets of load, store, and instruction fill transactions (six total).

5.3.2 Eventual Completion Guarantee of LR/SC. Without mechanisms that ensure eventual completion of LR/SC sequences, the probability of running into a livelock is high. The current RISC-V specification does not outline such a mechanism, and hence we propose to use an exponential back-off counter to ensure effective LR/SC conflict resolution. The mechanism is implemented in the miss-unit of the L1 data cache (which talks to the TRI) and works as follows: If an LR fails, any subsequently issued LR will not be sent to the TRI until the back-off counter reaches zero. The back-off counter initialises to a masked random value drawn from a 16 bit LFSR. The mask is created using a 16 bit shift register with all flops initially set to zero. A failing LR shifts a one into the shift register, exponentially increasing the range for random back-off times (up to $2^{16} - 1$ CPU cycles).

Platform Details. RISC-V uses Physical Memory Attributes (PMAs) to specify features of the memory map. Most relevant to BYOC, it details which memory ranges will be cached or uncached, which receive atomic operations, and which are idempotent. For Ariane, like OpenSPARC T1, addresses with a most significant bit of 1 are non-cacheable, non-idempotent, and do not receive atomics. Unlike SPARC, the main cacheable, idempotent, atomic-receiving memory range is automatically generated from the system XML description. RISC-V does not address non-cached access to cacheable memory. It is left as a platform detail and the UNIX platform specification is a work in progress.

We implemented the following platform peripherals that are required to successfully boot SMP Linux on a RISC-V manycore system. These include:

- **Debug:** The RISC-V draft spec v0.13 compliant debug module [47] governs external, multi-hart, run-control debug. The cores use their existing pipeline to facilitate debug functionality. An external debug request signal redirects the core to a "programmable" debug ROM which injects debug instructions into the core's pipeline.
- **CLINT:** The CLINT provides inter-processor interrupts (IPI) and a common time-base. Each core uses its own timer register which triggers an external timer interrupt when it matches the global time-base.
- **PLIC:** This interrupt controller manages external peripheral interrupts. It provides a context for each privilege level and core. Software can configure different priority thresholds for each context. Though not yet standardised, there is a reference implementation including a Linux driver which acts as a de facto standard.

We also added an automatic device tree generation script using PyHP. This script reads the XML description of the system (used to generate the chipset crossbars), the core count, and the clock frequency to generate a device tree that is compiled into a bootrom peripheral. That bootrom's "zero-stage" bootloader initialises the cores and loads a pointer to the device tree blob into register `a1` as per RISC-V convention. With this automatic device tree generation, the same Linux image can be booted on differently parameterised instances of BYOC, automatically adapting to the platform at runtime.

5.4 ao486

The ao486 [44] core is a full reimplementation of the Intel 486 SX processor (which lacks floating-point). The core has a 16KB L1 instruction cache and a 16KB data cache as well as a 32 entry TLB. Released alongside the core are the platform peripherals for an FPGA-based SoC capable of booting Linux and both Microsoft Windows 95 and 98. We were interested in ao486 because x86 brings support for legacy code and because CISC ISAs are rare among open source cores.

Our initial implementation aims to integrate ao486 while minimising changes to core RTL to further test the suitability of the TRI. ao486 is unique among our cores as it uses an Avalon memory interface which makes unaligned accesses to memory. Connecting to Avalon without modifications requires realignment in the transducer.

Platform Details. ao486 has been integrated into BYOC and can execute the Bochs [37] BIOS, enter x86 Protected Mode, and run programs on top of the BIOS. This functionality is similar to PicoRV32 and our other non-OS capable cores. However, cache invalidations are still in progress and atomics are done in the core's datapath, so the design requires workarounds to be used as a multicore.

The ao486 SoC provides Verilog implementations of peripherals required for full platform functionality. These include the 8259 Programmable Interrupt Controller (PIC), 8253/4 Programmable Interval Timer (PIT), and a Real Time Clock (RTC). As the platform is single-core, it does not provide the newer Intel Advanced PIC (APIC) used for multicore.

5.5 Other Core Prototypes

Beyond the cores detailed above, we have prototypes for testing other microarchitectures. Four of these, like PicoRV32, are 32 bit RISC-V cores which lack L1 caches and can run RISC-V assembly tests in BYOC. The first is an in-order, bit-serial core. Two more (one of which is written in PyMTL [21, 30]) are five-stage, in-order cores used for computer architecture education. The last is a minimal configuration of the highly parameterisable VexRiscv core [2].

AnyCore [10, 13, 14] is a high performance, highly configurable, out-of-order superscalar core that implements 64 bit RISC-V (RV64IMAFD). AnyCore's configurability made it easy to match BYOC's default parameters and the core runs C and assembly tests in BYOC. We modified the BPC and TRI to return the requested address back to AnyCore's L1 interface, to use for its index and way calculation. Further integration of AnyCore is quite feasible as it previously used the OpenSPARC T2 [36] variant of CCX. It could also reuse Ariane's 64 bit RISC-V platform support unmodified.

We also worked with the developers of the BlackParrot [1] 64 bit RISC-V multicore to connect the core to BYOC. BlackParrot has its own cache coherence system, so we tested multiple methods of connection, including a novel hierarchical coherence method. We settled on connecting the core with its L1 caches but without its coherence engines. This prototype ran single-core RISC-V assembly tests with just a few days of work.

Independently, the Lagarto RISC-V core was integrated into BYOC [26]. This shows BYOC's promise in enabling external users to quickly build their own cores into manycores with the option of heterogeneous-ISA capabilities.

5.6 Hardware Transactional Memory

A group of graduate students were able to modify BYOC and the OpenSPARC T1 core to build a hardware transactional memory (HTM) prototype as a six week course project. To minimise hardware modifications to the core and TRI, they used specially addressed memory accesses to indicate transaction start and commit. Thus, this HTM can be easily transplanted to other cores (though it may not be compatible with the core's ISA's HTM, which is usually coupled with microarchitectural features). In the BPC, four states are added to the coherence protocol to indicate that the line is currently read/written in an ongoing transaction or is in the commit process. In the LLC, the coherence protocol in unchanged, but four new NoC message types are added to implement the two-phase commit mechanism. This project shows that with a few changes in the BPC and the LLC, BYOC can be easily leveraged to implement different memory system research ideas, which can then be reused across multiple cores.

5.7 QEMU Co-simulation with BYOC

Xilinx's open source LibSystemCTLM-SoC library [56] provides a co-simulation environment between QEMU and RTL simulation. In two days, we and the library's developers built a new BYOC enabled proof-of-concept that connected QEMU to the TRI. We bypassed QEMU memory traffic from an emulated RISC-V core through a write-through L1 cache and a new TRI transducer, which enabled the use of the BPC and LLC for the reserved memory range. QEMU is unique in supporting a variety of ISAs, including those for which no or few open core implementations exist. This enables connecting "cores" of more ISAs to BYOC for further ISA heterogeneity and to develop multi-ISA platform support.

6 Results

To enable future heterogeneous-ISA research using BYOC, we conducted several experiments to measure latency, throughput, and comparative performance of two prototypes. The first is Oxy, where we characterise the PicoRV32 core and then compare the OpenSPARC T1 and PicoRV32 cores. For the second prototype, Dicho, with OpenSPARC T1 and Ariane cores, we measure the latency and throughput for cooperative applications running across the two cores which are independently running Linux.

Both of our experimental platforms were implemented at 66.67 MHz on a Digilent Genesys2 board, which has a Xilinx

Operation	Measured Latency (cycles)		True Latency (cycles)	
	Cached	Uncached	Cached	Uncached
Load	17	113 ± 1	4	100 ± 1
Store	17	113 ± 1	4	100 ± 1

Table 4. Memory latency measurements for PicoRV32. The measured latency is the raw cycle count from the test. True latency is adjusted for cycles spent in the cache hierarchy.

Kintex 7k325tffg900-2 FPGA. The BPC on each tile is 4-way associative and 8KB in size while the LLC shard on each tile is 4-way associative and 64KB in size.

6.1 Oxy

6.1.1 Experimental Setup. This Oxy system consists of one OpenSPARC T1 (OST1) and one PicoRV32 core connected via BYOC. The OST1 has its default 16KB, 4-way L1 instruction cache and 8KB, 4-way L1 data cache. We boot Debian Linux on the OST1 core and RISC-V programs are run on PicoRV32 using our proxy kernel.

The proxy kernel consists of a loader and a system to proxy syscalls from the PicoRV32 core. The loader allocates a block of memory for PicoRV32 and sets up the ELF image in that memory. The system to proxy syscalls consists of two parts. The first is a modified version of Newlib, run on the PicoRV32 core, that writes syscall arguments to a (cache-coherent) shared memory region. The second is a loop, run on the OST1 core, that polls on the shared memory. When it receives a syscall request, it executes the syscall locally, taking advantage of the OST1's Linux capability, and writes the return value to the same region of shared memory for the PicoRV32 core to read.

The `binfmt_misc` kernel subsystem enables invocation of RISC-V binaries from the OST1 core like native binaries. `binfmt_misc` allows a binary (typically an interpreter) to be associated with a binary type. The binary type can be determined from the ELF magic number. We registered our proxy kernel binary to be invoked for RISC-V executables using the ELF magic number for RISC-V binaries.

6.1.2 Memory Hierarchy Latency. We investigated the latency to different parts of the BYOC memory hierarchy to better understand performance of the PicoRV32 core. Cycle counts for instructions that are only dependent on the core itself are given on GitHub [55].

We measured latency (shown in Table 4) of a memory operation between two `rdcycle` instructions. The raw measurements given are the difference between the two `rdcycle` values. The PicoRV32 core is unpipelined and takes multiple cycles to execute an instruction. Thus we adjust the raw measurements to calculate the latency within the memory hierarchy versus the other portions of instruction execution.

We first determined the true BPC hit time from the raw BPC hit time of 17 cycles. Every instruction fetch must access the BPC, so a memory instruction accesses the BPC twice.

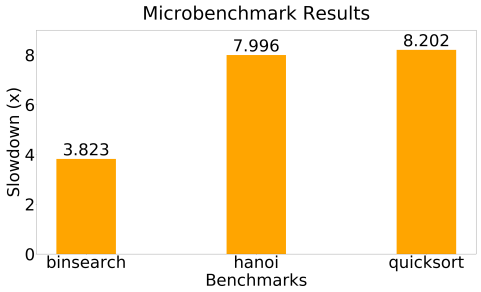

Figure 4. Multiplicative slowdown of running microbenchmarks on the PicoRV32 core versus the OST1 core. Slowdown is also given over each bar.

Some of the latency is also from fetching the second `rdcycle` instruction. Thus, the instructions account for 3 BPC accesses. In addition, load and store instructions take 5 cycles in the PicoRV32 core (given by its documentation [55] and verified in simulation). After subtracting the 5 cycles for the memory instruction and dividing by the 3 memory accesses, we find that a BPC hit for PicoRV32 takes **4 cycles**.

DRAM access latency from our Kintex-7 FPGA to the DRAM on the Genesys2 board is around 113 cycles, with some variance due to clock domain crossings. For this test, both instruction reads go to the BPC and only the actual memory access goes to DRAM. With this in mind, one operation to memory for the PicoRV32 core is about **100 cycles**. Latency from BPC to LLC is the same for PicoRV32 and OST1 cores and varies with core count and the LLC cache homing policy as described in [7].

6.1.3 Microbenchmarks. We ran three microbenchmarks to compare the performance of the PicoRV32 core and OpenSPARC T1 core integrated into BYOC. `hanoi` recursively solves the Towers of Hanoi puzzle with a height of 7. `binsearch` does a binary search for 10 keys randomly chosen in an array of 10,000 32-bit integers. `quicksort` sorts an array of 100 32-bit integers shuffled randomly.

The slowdown of each benchmark is shown in Figure 4. All microbenchmarks experienced a slowdown when running on the PicoRV32 core since it has a less complex microarchitecture. `hanoi` and `quicksort` both saw about an 8x slowdown. `binsearch` experienced a smaller slowdown at 4x.

`binsearch`'s working set does not fit in the 8KB BPC (though it does fit in the LLC). As a result of this, both cores must access the LLC or memory often. Since accesses to the BPC or beyond take approximately the same amount of time for both PicoRV32 and OST1, `binsearch` is less impacted by running on the PicoRV32 core.

Although microbenchmarks running on PicoRV32 suffer reduced performance, PicoRV32 is designed to minimise FPGA area and maximise frequency. The OST1 core was designed for throughput and not single-threaded performance. The core originally had 4 threads to overlap useful work. These trade-offs between performance and other metrics are an intended consequence of having a heterogeneous system

architecture. An intelligent scheduler would optimise for these and use the most appropriate core for its performance and energy-consumption goals.

6.2 Dicho

6.2.1 Experimental Setup. Dicho consists of one OST1 and one Ariane core connected via BYOC. Both cores have their default 8KB, 4-way L1 data caches and 16KB, 4-way instruction caches. Ariane is allocated the first 512MB of memory, OST1 is allocated the next 496MB, and the remaining 16MB is reserved as a fixed communication region. The OST1 core uses a similar mechanism to Oxy. A syscall allocates a contiguous region of physical memory to share. The address of this buffer is written to the fixed region to communicate to Ariane. Both cores then use `/dev/mem` to map the shared buffer with user-mode accessible virtual addresses.

6.2.2 Latency Results. Our latency microbenchmark experiment uses two shared memory FIFO queues between the two cores, one for each direction. Each core removes items from its inbound queue and places items into its outbound queue, updating the head and tail pointers as necessary.

We first measured round-trip latency by inserting a single item into the queues and passing it through ten round trips. These ten round trips define a single iteration. We repeat this for ten iterations, discounting the first iteration for warmup as it takes ~10x the cycles of the subsequent iterations.

We measure 78.97 cycles per round-trip using atomics and 73.50 cycles without atomics. This corresponds to two LLC downgrades (one in each direction) needed after the head pointer for each queue has been updated. This downgrade latency is in line with the L2 cache hit latency observed in a previous characterisation of the Piton processor [34], which was an ASIC implementation of the OpenPiton platform we started our work from. A downgrade round trip will take additional cycles over an LLC hit as the LLC must then request the downgrade itself from the BPC.

Throughput Results. We measured system throughput using a single queue producer-consumer microbenchmark. The OST1 core waits for an empty queue slot to insert an item. Every iteration Ariane waits for the item and takes it out. 10,000 insertions/extractions define a single iteration. We use ten iterations: one for warmup and nine for measurements, and those last nine times are averaged.

We measure 7.52 cycles for one insertion/extraction, which corresponds to 566 Mbps throughput at the 66.7 MHz FPGA clock frequency. The maximum throughput may be higher as our setup restricted the queue size to a maximum of 4KB.

7 Related Work

Heterogeneous Cache Coherence. There has been recent work on building cache coherence systems for SoCs which integrate coherent accelerators. Spandex [3] defines a combination of cache coherence protocols targeting heterogeneous systems. Embedded Scalable Platforms (ESP) include an FPGA-based framework for prototyping and analysing heterogeneous SoCs [32, 33] with their focus on integration of accelerators. ESP's cache coherence system has been extended with support for cache coherence schemes specifically optimised for accelerators [18]. Both Spandex and ESP's accelerator coherence models could be integrated into BYOC as extensions to the existing cache coherence protocol.

Similarly, PULP clusters could be integrated into BYOC using the TRI to make a system similar to HERO [24] but featuring more platform support and enabling coherence capabilities for the clusters.

Comparing TRI to Other Interfaces. TileLink can provide a similar cache coherent memory system to BYOC. We believe that it is more convenient to connect cores using TRI than TileLink. TileLink uses between two and five channels and has 19 different message types in total. TRI only requires request and response channels with 11 total message types. According to its specification, TileLink [49] "was designed for RISC-V but supports other ISAs". However, we are not aware of its use for building heterogeneous-ISA systems.

Buses like Wishbone [42, 43] and Arm AXI [4] are designed primarily for peripheral connections rather than providing cache-coherent interconnects. Arm ACE is an extension of AXI which "provides support for hardware-coherent caches" [4]. ACE relies primarily on snoopy cache coherence, which limits its scalability compared to BYOC.

Crossing Guard (XG) provides a platform-agnostic interface for providing varying levels of coherence to accelerators [40]. XG is conceptually similar to TRI in providing some decoupling of accelerators from the coherence protocol. XG also provides safety guarantees to the coherence system in the presence of misbehaving accelerators.

8 Conclusion

In this paper, we introduced BYOC: an open-source hardware framework for heterogeneous-ISA and heterogeneous systems research. Notable for enabling this type of research, BYOC provides a "Bring Your Own Core" interface to its local, private cache. This "Transaction-Response Interface" (TRI) is straightforward and extensible to connect to, as demonstrated by our connection of ten cores of four different ISAs. With these cores, we have developed several prototypes. The first, Oxy, features OpenSPARC T1 and PicoRV32 cores, with the OpenSPARC T1 core capable of offloading RISC-V binaries from SPARC Linux to the PicoRV32 core. We characterised Oxy to measure BYOC memory hierarchy latency and provide a classical performance comparison of the two cores of different ISAs, as performed in previous simulation-based heterogeneous-ISA research. The second prototype, Dicho, reflects a heterogeneous-ISA system of the future,

with OpenSPARC T1 and Ariane cores which can both boot Linux and share memory. We characterised Dicho's latency and throughput in executing a cooperative shared memory application run across the two Linux instances.

BYOC and the prototypes built with it will form the basis of future heterogeneous-ISA research. For systems research, Dicho provides a solid foundation for the development of operating systems, schedulers, and more systems software designed to run on heterogeneous-ISA systems with hardware-coherent shared memory.

9 Acknowledgements

We would like to thank Ting-Jung Chang, Paul Jackson, Michael McKeown, Kathleen Feng, Zhenyu Song, Aninda Manocha, Teague Tomesh, Christopher Grimm, Tigar Cyr, Kaifeng Xu, Yushan Su, Daniel Petrisko, Mark Wyse, Francisco Iglesias, Edgar Iglesias, and Rangeen Basu Roy Chowdhury for their assistance in making BYOC a reality. This material is based on research sponsored by SNF IZHRZ0 180625 Heterogeneous Computing Systems with Customized Accelerators, the NSF under Grants No. CNS-1823222 and CCF-1453112, Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreements No. FA8650-18-2-7846, FA8650-18-2-7852, and FA8650-18-2-7862. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA), the NSF, DARPA, or the U.S. Government.

A Artifact Appendix

A.1 Abstract

BYOC is a significant piece of infrastructure that has many moving parts. This appendix deals specifically with reproducing the Dicho latency experiment from Section 6.2.2. We provide a DOI for the FPGA bitfiles and OS images to reproduce this experiment. The code for this and the other parts of BYOC are available via GitHub in several ways:

- **Oxy and PicoRV32** The Oxy prototype is an instance of JuxtaPiton, which has been open source as an upstream part of OpenPiton since release 9.
- **Ariane** OpenPiton+Ariane has been open source as an upstream part of OpenPiton since release 10.
- **Dicho** The working Dicho code is available and reproducible on the BYOC GitHub repository [2].
- **ao486** The working ao486 code is available and reproducible on the BYOC GitHub repository [3].

[2] https://github.com/bring-your-own-core/byoc/tree/dicho_repro
[3] https://github.com/bring-your-own-core/byoc/tree/ao486

The Dicho experiment has relatively specific hardware requirements (mainly the two Genesys 2 FPGA boards). To build everything from source (FPGA bitfiles, microbenchmarks for OST1 and Ariane, bootloaders, etc.) requires multiple cross-compilers and other time-intensive infrastructure, so we instead provide pre-compiled bitfiles, binaries, etc. With the appropriate hardware and our instructions below, it should be straightforward to reproduce the results for this experiment.

A.2 Artifact check-list (meta-information)

- **Hardware:** 2 Digilent Genesys 2 Boards with Xilinx Kintex-7 FPGA, 1 4GB SDHC card, 2 FAT-formatted USB drives
- **Execution:** Boot Linux on each core, manually run binaries
- **Metrics:** Latency (nanoseconds)
- **Output:** 10 rounds of round-trip latency measurement
- **Experiments:** Latency with/without atomics
- **How much disk space required (approximately)?:** 5GB
- **How much time is needed to prepare workflow (approximately)?:** 1 hour
- **How much time is needed to complete experiments (approximately)?:** 5 minutes
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** RTL: BSD, Apache, ISC, GPL. Microbenchmark: BSD
- **Archived (provide DOI)?:** 10.5281/zenodo.3563256 (URL: https://doi.org/10.5281/zenodo.3563256)

A.3 Description

A.3.1 How delivered. We provide the OS images including the pre-built microbenchmark binaries. We also provide bitfiles to program the two FPGAs.

A.3.2 Hardware dependencies. Our Dicho latency measurement experiment used two Digilent Genesys 2 FPGA boards, but **both cores are running on one board and the second board is only used for its UART connection**. To provide a UART for each of the two cores, we wired secondary UART tx/rx pins from the PMOD connector on the first board to a second identical board. Alternatively one could connect to a microcontroller or similar that can connect to the 3.3V PMOD pins and provide access to the second serial terminal.

We also need:

- 4GB MicroSDHC card onto which to write the OS images.
- Two FAT-formatted USB drives (exFAT and others will not work) to program the two FPGAs.
- 3 breadboard wires to connect PMOD connectors.

A.3.3 Software dependencies. We assume that the host machine the MicroSD card is written from runs a Unix-friendly OS that can use `dd` to write MicroSD cards, including writing with different offsets. Primary dependencies are:

- `dd` to write images onto MicroSD card.
- PUTTY/screen/minicom or similar to provide serial terminal connection to both FPGAs.
- `tar` to unarchive optional microbenchmark tarball.

A.4 Installation

To install the OS images (which contain the pre-built binaries) onto your MicroSDHC card, follow these steps in order (note that dd on macOS uses a lower-case m for the bs argument):

1. Identify your SD card's device location. We use `/dev/MYSDCARD` as our example.
2. `sudo dd if=dicho_ariane_linux.img of=/dev/MYSDCARD bs=1M seek=2048`
3. `sudo dd if=dicho_ost1_linux.img of=/dev/MYSDCARD bs=1M`
4. `sudo dd if=dicho_ariane_bbl.bin of=/dev/MYSDCARD bs=1M seek=2049`
5. `sudo dd if=dicho_ost1_prom.bin of=/dev/MYSDCARD bs=1M`

Next you must copy each of the two FPGA bitfiles onto its own USB drive. Plug the USB drive into the top USB port of the FPGA board. Plug the MicroSDHC card into the board which has the bitfile for `dicho_g2_repro.bit` (you will know which board it is as when it is programmed, the OLED screen will light up).

A.5 Experiment workflow

1. Plug a micro USB cable into each powered-off board's UART port and connect both to the host computer.
2. Establish two serial terminals on the host computer (one for each FPGA) with 115200-8-N-1 configuration.
3. Connect three breadboard cables to PMOD connector JC, pins 9, 10, and 11, matching the same pins on both boards [4].
4. Ensure both boards have jumpers set to program from USB.
5. Power on the FPGA board which has the `uart_passthru.bit` bitfile in its USB drive and wait for the programming DONE light to illuminate.
6. Power on the FPGA board with the other USB and the MicroSDHC card.
7. Once the OLED screen lights up, both serial terminals start to print output from the SPARC and RISC-V boot processes.
8. The SPARC boot process will reach an "ok" prompt. On that serial terminal, enter: `boot Linux init=/bin/bash`
9. The RISC-V boot process will reach a login prompt. Log in as the `root` user.
10. Once both serial terminals show a bash shell prompt, it is time to run the experiment.

A.6 Evaluation and expected result

The first test uses atomics. In the serial terminals, run:

1. On SPARC, run: `/home/guest/main_ost1_O2_x10`
2. On RISC-V, run: `/bin/main_ariane_O2_x10`

The second test does not use atomics or fences. In the serial terminals, run:

1. On SPARC, run: `/home/guest/main_ost1_nofence_O2_x10`
2. On RISC-V, run: `/bin/main_ariane_nofence_O2_x10`

The serial terminal output from the SPARC core will give the nanosecond count for ten iterations of the test, where each iteration is one hundred round-trips. The first iteration, as noted in Section 6.2.2 is warmup, so average the other nine iterations to get an average nanosecond count for 100 roundtrips. The core runs at 66.67MHz, meaning a cycle takes 15 nanoseconds. If you divide your measured average by 1500, then you will get the average cycle count for one round-trip. As noted, we expect latency of 78.97 cycles per round-trip using atomics and 73.50 cycles without atomics.

[4] https://reference.digilentinc.com/reference/programmable-logic/genesys-2/reference-manual#pmod_connectors

A.7 Methodology

Submission, reviewing and badging methodology:

- http://cTuning.org/ae/submission-20190109.html
- http://cTuning.org/ae/reviewing-20190109.html
- https://www.acm.org/publications/policies/artifact-review-badging

References

[1] BlackParrot. https://github.com/black-parrot, 2020.

[2] VexRiscv. https://github.com/SpinalHDL/VexRiscv, 2020.

[3] Johnathan Alsop, Matthew D. Sinclair, and Sarita V. Adve. Spandex: A flexible interface for efficient heterogeneous coherence. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA '18, pages 261–274, Piscataway, NJ, USA, 2018. IEEE Press.

[4] Arm. AMBA AXI and ACE protocol specification. https://static.docs.arm.com/ihi0022/g/IHI0022G_amba_axi_protocol_spec.pdf. Accessed: 2019-08-15.

[5] ARM. ARM Technologies: big.LITTLE.

[6] Raghuraman Balasubramanian, Vinay Gangadhar, Ziliang Guo, Chen-Han Ho, Cherin Joseph, Jaikrishnan Menon, Mario Paulo Drumond, Robin Paul, Sharath Prasad, Pradip Valathol, and Karthikeyan Sankaralingam. Enabling GPGPU low-level hardware explorations with MIAOW: An open-source RTL implementation of a GPGPU. *ACM Trans. Archit. Code Optim.*, 12(2), June 2015.

[7] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, and David Wentzlaff. OpenPiton: An open source manycore research framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 217–232, New York, NY, USA, 2016. ACM.

[8] Jonathan Balkind, Michael Schaffner, Katie Lim, Florian Zaruba, Fei Gao, Jinzheng Tu, David Wentzlaff, and Luca Benini. OpenPiton+Ariane: The first open-source, SMP Linux-booting RISC-V system scaling from one to many cores. In *Third Workshop on Computer Architecture Research with RISC-V*, CARRV '19, 2019.

[9] Elena Gabriela Barrantes, David H. Ackley, Stephanie Forrest, Trek S. Palmer, Darko Stefanovic, and Dino Dai Zovi. Randomized instruction set emulation to disrupt binary code injection attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 281–289, New York, NY, USA, 2003. ACM.

[10] Rangeen Basu Roy Chowdhury. *AnyCore: Design, Fabrication, and Evaluation of Comprehensively Adaptive Superscalar Processors.* PhD thesis, North Carolina State University, 2016.

[11] Sharath K. Bhat, Ajithchandra Saya, Hemedra K. Rawat, Antonio Barbalace, and Binoy Ravindran. Harnessing energy efficiency of heterogeneous-ISA platforms. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, HotPower '15, pages 6–10, New York, NY, USA, 2015. ACM.

[12] Alex Bradbury, Gavin Ferris, and Robert Mullins. Tagged memory and minion cores in the lowRISC SoC. Technical report, Computer Laboratory, University of Cambridge, Dec 2014.

[13] Rangeen Basu Roy Chowdhury, Anil K. Kannepalli, Sungkwan Ku, and Eric Rotenberg. Anycore: A synthesizable RTL model for exploring and fabricating adaptive superscalar cores. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 214–224, April 2016.

[14] Rangeen Basu Roy Chowdhury, Anil K. Kannepalli, and Eric Rotenberg. Anycore-1: A comprehensively adaptive 4-way superscalar processor. In *2016 IEEE Hot Chips 28 Symposium (HCS)*, pages 1–1, Aug 2016.

[15] Scott Davidson, Shaolin Xie, Chris Torng, Khalid Al-Hawaj, Austin Rovinski, Tutu Ajayi, Luis Vega, Chun Zhao, Ritchie Zhao, Steve Dai, Aporva Amarnath, Bandhav Veluri, Paul Gao, Anuj Rao, Gai Liu, Rajesh K. Gupta, Zhiru Zhang, Ronald Dreslinski, Christopher Batten, and Michael Bedford Taylor. The Celerity Open-Source 511-core RISC-V Tiered Accelerator Fabric. *Micro, IEEE*, Mar/Apr. 2018.

[16] Matthew DeVuyst, Ashish Venkat, and Dean M. Tullsen. Execution migration in a heterogeneous-ISA chip multiprocessor. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 261–272, New York, NY, USA, 2012. ACM.

[17] Yaosheng Fu, Tri M. Nguyen, and David Wentzlaff. Coherence domain restriction on large scale systems. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, pages 686–698, New York, NY, USA, 2015. ACM.

[18] Davide Giri, Paolo Mantovani, and Luca P. Carloni. NoC-based support of heterogeneous cache-coherence models for accelerators. In *Proceedings of the Twelfth IEEE/ACM International Symposium on Networks-on-Chip*, NOCS '18, pages 1:1–1:8, Piscataway, NJ, USA, 2018. IEEE Press.

[19] Mark D. Hill and Vijay Janapa Reddi. Accelerator-level parallelism. *CoRR*, abs/1907.02064, 2019.

[20] Texas Instruments. OMAP4470 multimedia device: Technical reference manual. http://www.ti.com/lit/ug/swpu270t/swpu270t.pdf, 2014. Accessed: 2019-08-15.

[21] Shunning Jiang, Torng Christopher, and Christopher Batten. PyMTL: A unified framework for vertically integrated computer architecture research. In *Workshop on Open-Source EDA Technology*, WOSET, 2018.

[22] Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 81–, Washington, DC, USA, 2003. IEEE Computer Society.

[23] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, and Keith I. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, ISCA '04, pages 64–, Washington, DC, USA, 2004. IEEE Computer Society.

[24] Andreas Kurth, Alessandro Capotondi, Pirmin Vogel, Luca Benini, and Andrea Marongiu. HERO: An open-source research platform for HW/SW exploration of heterogeneous manycore systems. In *Proceedings of the 2Nd Workshop on AutotuniNg and aDaptivity AppRoaches for Energy Efficient HPC Systems*, ANDARE '18, pages 5:1–5:6, New York, NY, USA, 2018. ACM.

[25] George Kyriazis. Heterogeneous system architecture: A technical review. *AMD Fusion Developer Summit*, page 21, 2012.

[26] Neiel I. Leyva-Santes, Ivan Pérez, César A. Hernández-Calderón, Enrique Vallejo, Miquel Moretó, Ramón Beivide, Marco A. Ramírez-Salinas, and Luis A. Villa-Vargas. Lagarto I RISC-V multi-core: Research challenges to build and integrate a network-on-chip. In Moisés Torres and Jaime Klapp, editors, *Supercomputing*, pages 237–248, Cham, 2019. Springer International Publishing.

[27] Katie Lim, Jonathan Balkind, and David Wentzlaff. JuxtaPiton: Enabling heterogeneous-ISA research with RISC-V and SPARC FPGA soft-cores. *CoRR*, abs/1811.08091, 2018.

[28] Katie Lim, Jonathan Balkind, and David Wentzlaff. JuxtaPiton: Enabling heterogeneous-ISA research with RISC-V and SPARC FPGA soft-cores. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, pages 184–184, New York, NY, USA, 2019. ACM.

[29] Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong. K2: A mobile operating system for heterogeneous coherence domains. *ACM Trans. Comput. Syst.*, 33(2):4:1–4:27, June 2015.

[30] Derek Lockhart, Gary Zibrat, and Christopher Batten. PyMTL: A unified framework for vertically integrated computer architecture research. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, page 280–292, USA, 2014. IEEE Computer Society.

[31] Robert Lyerly, Antonio Barbalace, Christopher Jelesnianski, Vincent Legout, Anthony Carno, and Binoy Ravindran. Operating system process and thread migration in heterogeneous platforms. In *2016 Workshop on Multicore and Rack-Scale systems*, Apr 2016.

[32] P. Mantovani, E. G. Cota, C. Pilato, G. Di Guglielmo, and L. P. Carloni. Handling large data sets for high-performance embedded applications in heterogeneous systems-on-chip. In *2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES)*, pages 1–10, Oct 2016.

[33] Paolo Mantovani, Emilio G. Cota, Kevin Tien, Christian Pilato, Giuseppe Di Guglielmo, Ken Shepard, and Luca P. Carloni. An FPGA-based infrastructure for fine-grained DVFS analysis in high-performance embedded systems. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pages 157:1–157:6, New York, NY, USA, 2016. ACM.

[34] Michael McKeown, Alexey Lavrov, Mohammad Shahrad, Paul J Jackson, Yaosheng Fu, Jonathan Balkind, Tri M Nguyen, Katie Lim, Yanqi Zhou, and David Wentzlaff. Power and energy characterization of an open source 25-core manycore processor. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 762–775, Feb 2018.

[35] Sun Microsystems. *OpenSPARC T1 Microarchitecture Specification*. Santa Clara, CA, 2006.

[36] Sun Microsystems. *OpenSPARC T2 Core Microarchitecture Specification*. Santa Clara, CA, 2007.

[37] Darek Mihocka and Stanislav Shwartsman. Virtualization without direct execution - designing a portable VM. In *1st Workshop on Architectural and Microarchitectural Support for Binary Translation*, June 2008.

[38] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[39] NVIDIA. NVIDIA Deep Learning Accelerator. http://nvdla.org, 2018. Accessed: 2019-07-30.

[40] Lena E. Olson, Mark D. Hill, and David A. Wood. Crossing Guard: Mediating host-accelerator coherence interactions. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, page 163–176, New York, NY, USA, 2017. ACM.

[41] Oracle. OpenSPARC T1. http://www.oracle.com/technetwork/systems/opensparc/opensparc-t1-page-1444609.html.

[42] OpenCores Organization. Wishbone B4 - WISHBONE system-on-chip (SoC) interconnection architecture for portable IP cores. https://cdn.opencores.org/downloads/wbspec_b4.pdf. Accessed: 2019-08-15.

[43] OpenCores Organization. WISHBONE system-on-chip (SoC) interconnection architecture for portable IP cores. https://cdn.opencores.org/downloads/wbspec_b3.pdf. Accessed: 2019-08-15.

[44] Aleksander Osman. ao486. https://github.com/alfikpl/ao486, 2014. Accessed: 2019-07-30.

[45] Princeton University. OpenPiton Research Platform. https://github.com/PrincetonUniversity/openpiton, 2019.

[46] Andreas Prodromou, Ashish Venkat, and Dean M. Tullsen. Deciphering predictive schedulers for heterogeneous-ISA multicore architectures. In *Proceedings of the 10th International Workshop on Programming Models and Applications for Multicores and Manycores*, PMAM'19, pages 51–60, New York, NY, USA, 2019. ACM.

[47] RISC-V Foundation. RISC-V External Debug Support Version 0.13 - DRAFT. https://github.com/riscv/riscv-debug-spec/releases/download/task_group_vote/riscv-debug-draft.pdf, 2018.

[48] Sophia Shao and Emma Wang. Die photo analysis. http://vlsiarch.eecs.harvard.edu/research/accelerators/die-photo-analysis/. Accessed: 2019-08-15.

[49] SiFive Inc. SiFive TileLink specification. https://www.sifive.com/documentation/tilelink/tilelink-spec/. Accessed: 2019-08-15.

[50] Kanad Sinha, Vasileios P. Kemerlis, and Simha Sethumadhavan. Reviving instruction set randomization. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 21–28, May 2017.

[51] Michael Bedford Taylor. BaseJump STL: SystemVerilog needs a standard template library for hardware design. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, pages 73:1–73:6, New York, NY, USA, 2018. ACM.

[52] Ashish Venkat, Harsha Basavaraj, and Dean M. Tullsen. Composite-ISA cores: Enabling multi-ISA heterogeneity using a single ISA. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 42–55, Feb 2019.

[53] Ashish Venkat, Sriskanda Shamasunder, Hovav Shacham, and Dean M. Tullsen. HIPStR: Heterogeneous-ISA program state relocation. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 727–741, New York, NY, USA, 2016. ACM.

[54] Ashish Venkat and Dean M. Tullsen. Harnessing ISA diversity: Design of a heterogeneous-ISA chip multiprocessor. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 121–132, Piscataway, NJ, USA, 2014. IEEE Press.

[55] Clifford Wolf. PicoRV32. https://github.com/cliffordwolf/picorv32. Accessed: 2019-08-04.

[56] Xilinx. LibSystemCTLM-SoC. https://github.com/Xilinx/libsystemctlm-soc, 2019. Accessed: 2019-07-30.

[57] Florian Zaruba and Luca Benini. The cost of application-class processing: Energy and performance analysis of a Linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 1–12, 2019.

[58] Yanqi Zhou and David Wentzlaff. MITTS: Memory inter-arrival time traffic shaping. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 532–544, June 2016.