AFRL-RY-WP-TR-2023-0134

COHERENT INTERCONNECT AND FIELD PROGRAMMABLE GATE ARRAY (FPGA) ENABLING REUSE (CIFER)

David Wentzlaff
Princeton University

Christopher Batten
Cornell University

OCTOBER 2023
Final Report

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with The Under Secretary of Defense memorandum dated 24 May 2010 and AFRL/DSO policy clarification email dated 13 January 2020. This report is available to the general public, including foreign nationals.

Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2023-0134 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

BOZADA.CHRISTOPHER.A.1131856993 Digitally signed by BOZADA.CHRISTOPHER.A.1131856993 Date: 2023.09.20 12:15:39 -04'00'

CHRISTOPHER A. BOZADA
Program Manager
Aerospace Components and Subsystems Division

WILKINS.GENE.MICHAEL.1116849263 Digitally signed by WILKINS.GENE.MICHAEL.1116849263 Date: 2023.09.22 12:35:35 -04'00'

GENE M. WILKINS, Lt Col, USAF
Deputy Chief, Aerospace Components &
Subsystems Technology Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show "//Signature//" stamped or typed above the signature blocks.

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
October 2023	Final	11 June 2018	13 December 2022

4. TITLE AND SUBTITLE

COHERENT INTERCONNECT AND FIELD PROGRAMMABLE GATE ARRAY (FPGA) ENABLING REUSE (CIFER)

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
FA8650-18-2-7852	N/A	62716E
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
N/A	N/A	Y1TM

6. AUTHOR(S)

David Wentzlaff (Princeton University)
Christopher Batten (Cornell University)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)		8. PERFORMING ORGANIZATION REPORT NUMBER
Princeton University 171 Broadmead St Princeton, NJ 08540	Cornell University 245 Day Hall. Ithaca, NY 14853	

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'SREPORT NUMBER(S)
Air Force Research Laboratory Sensors Directorate Wright-Patterson Air Force Base, OH 45433-7320 Air Force Materiel Command United States Air Forces	Defense Advanced Research Projects Agency (DARPA/MTO) 675 North Randolph Street Arlington, VA 22203	AFRL/RYD	AFRL-RY-WP-TR-2023-0134

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

This material is based on research sponsored by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-2-7852. 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 the Air Force Research Laboratory (AFRL), the Defense Advanced Research Projects Agency (DARPA), or the U.S. Government. Report contains color.

14. ABSTRACT

This effort has worked to create high-quality open-source Intellectual Property blocks as part of the DARPA POSH program, and in particular has focused on creating a Scalable Cache Coherence system and a Field Programmable Gate Array. As a portion of creating the Scalable Cache Coherent Memory System, this work created a parameterizable On-Chip Network. This work silicon tested these blocks in the CIFER chip tape-out.

15. SUBJECT TERMS

Open-source electronics intellectual property, field programmable gate array, scalable cache, on-chip network

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
Unclassified	Unclassified	Unclassified	SAR	27

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Christopher Bozada	N/A

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

PREVIOUS EDITION IS OBSOLETE.

Table of Contents

Section **Page**

List of Figures

Figure **Page**

ACKNOWLEDGEMENTS AND DISCLAIMER

This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-2-7852. 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) or the U.S. Government.

We wish to thank the three DARPA MTO program managers Andreas Olofsson, Serge Leef, James Wilson, and Sung-Kyu Lim who provided support and feedback on this work. We also wish to thank the DARPA team and AFRL team who provided support to this work.

1 INTRODUCTION

The DARPA Posh Open Source Hardware (POSH) program seeks to reduce the cost of designing future (DoD) computer chips and Systems on Chip (SoC) by funding the creation of an ecosystem of open-source hardware Intellectual Property (IP) modules that can be used together. By creating high-quality, silicon-proven, well documented, open-source, IP modules, the design time and cost of creating future computer chips and SoCs can potentially be reduced. By having open-source designs, the designs can also be inspected and validated from a security perspective which can be difficult to do with closed source IP. The program has funded different performers to create different IP blocks that are primarily open-source to attempt to accomplish this goal. The POSH program has used innovative integration exercises to encourage the performers to meet, share designs and code, integrate designs, and work together to fix complex issues.

As part of the POSH program, the CIFER: Coherent Interconnect and FPGA Enabling Reuse team has worked to create high quality open-source IP blocks, and has focused on creating a Scalable Cache Coherence System and a Field Programmable Gate Array (FPGA). The Scalable Cache Coherence System leverages and builds upon the memory system portion of OpenPiton and is a directory-based cache coherence system that can coherently interconnect different types of processor cores. To connect the different cores, the CIFER project has created a parameterizable On-Chip Network which uses Python to customize the network called PyOCN. For the FPGA IP block, the CIFER team has created the Princeton Reconfigurable Gate Array (PRGA) which is highly customizable, FPGA generator which uses open-source tools to synthesize, place, map, and route designs to the generated FPGAs. PRGA is written in Python and highly customizable. To demonstrate and silicon-verify the open-source IP blocks designed in CIFER, the CIFER team has taped out the CIFER chip in the GlobalFoundries 12nm process node. The CIFER chip is a 4mm x 4mm chip that integrates cache coherently multiple RISC-V cores together with a PRGA-generated FPGA. The CIFER team has attended the POSH integration exercises and discussed our designs with other POSH and IDEA performers.

2 SUMMARY

The CIFER team has worked to create high-quality, open-source, IP blocks that are easy to use by others and can be used in future SoCs. The CIFER team has created a Scalable Cache Coherence system that can connect cores of differing types. As part of this Scalable Cache Coherence system, the CIFER team has created a Python-configurable On-chip Network generator called PyOCN. The second major IP block that this work has created is the Princeton Reconfigurable Gate Array (PRGA) which is a FPGA generator which can be configured with Python. These two IP blocks have been integrated together and silicon tested by the CIFER team with the creation of the CIFER chip which is a hybrid Heterogeneous Multicore / embedded FPGA SoC. This chip was taped-out and tested as part of the CIFER program. To ease the testing of open-source IP blocks, the CIFER team created a methodology to perform property-based random testing (PyH2).

In particular, the highlight achievements of the CIFER team include:

- Open-Source Scalable Coherent Memory System
    - Open-Source Pluggable cache coherent interface that supports heterogenous ISA designs (TRI)
- Open-Source Python-based On-Chip Network Generator (PyOCN)
- Property-Based Random Testing for Open-Source IP (PyH2)
- Open-Source FPGA Generator (PRGA)
    - Novel Cycle-free FPGA routing design and algorithm enables eFPGA portability and use of off-the-shelf EDA tools
- Taped-out and successfully tested a Heterogeneous Multicore/eFPGA SoC

In the following sections, this report discusses these main achievements in more detail.

3 RESULTS

3.1 Open-Source Scalable Cache Coherent Memory System

(https://github.com/PrincetonUniversity/openpiton/tree/openpiton-dev)

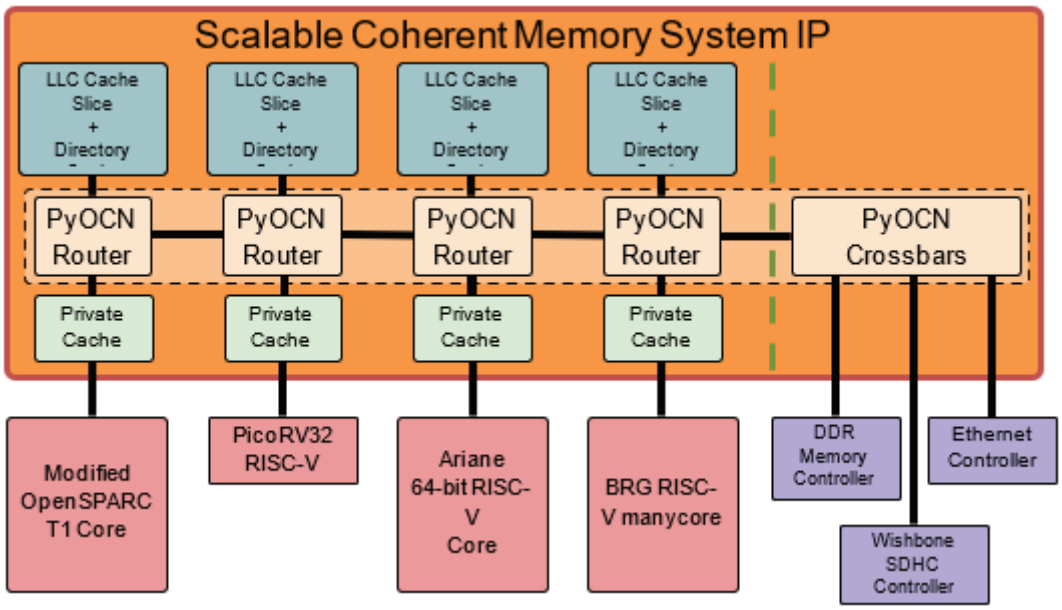

Figure 1: High-Level Block Diagram of the Scalable Coherent Memory System

The Scalable Cache Coherent Memory system IP block is an IP block that enables cache coherent memory communication between processor cores, I/O devices, and even FPGAs. It provides a last-level cache with directory which implements a directory-based cache coherence protocol. It provides a private cache which is designed to be instruction set architecture (ISA) agnostic. Each private cache is integrated with a processor core via the Transaction-Response Interface (TRI) which is an interface designed to support many different core types. The different caches are connected together with a configurable on-chip network (PyOCN) which is described in the next section. The Scalable Cache Coherent Memory system is designed to be an SoC backbone and enable next generation SoCs with potentially hundreds of cores. The Scalable Cache Coherent Memory System leverages the OpenPiton memory system.

The Scalable Cache Coherent Memory system uses three on-chip networks to connect the various components. These three networks are assigned a strict priority and the communication on a higher priority network cannot block waiting on a lower priority network. The coherence protocol is made out of a set of messages that are sent over the three networks. The choice of which network a message type is sent on is statically assigned (at protocol design time) to prevent deadlocks.

The Scalable Cache Coherent Memory System implements a distributed last level cache (LLC) which is made up of slices/shards. In addition, each core has a private cache provided by the Scalable Cache Coherent Memory System. The private cache may be implemented in addition to a cache inside of the processor core. Each cache slice contains a portion of the LLC cache

storage as well as a directory which tracks which private caches have a copy of a particular cache line. It also tracks the state of the cache line in the private cache. When a processor requires a cache line, it may find it in its private cache. If it is not in the private cache, it messages the home node (slice) associated with the physical address of the data. The home node contains a directory and using that information may take different actions. If the data is not in any other private caches, the home node can provide the data and add the requesting cache to the share list (in case of a clean read), or it could add it as the owner of the data. If the data is in another private cache (or multiple), and there is a need to write the data, the home node will invalidate all other copies before returning the data to the original requester. If the data is only shared and only being read, it will be added to the share list. Finally, if the data is dirty in another private cache, then the original owner needs to be invalidated before providing data to the requesting core.

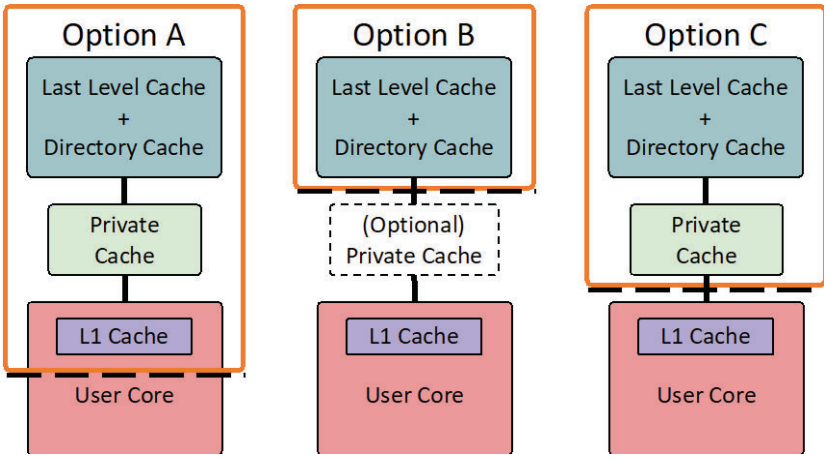

Figure 2: Options of where to Interface a Cache Coherent Memory System with a User Core

An important design decision for the Scalable Cache Coherent Memory system is what is the correct interface between the cores and the memory system. The above Figure shows three likely locations to interface between a memory system and a user core. Option A is to have the Scalable Cache Coherent system include the Level 1 cache. Unfortunately, this is typically tightly integrated with the core. Option B is to have the user core directly speak the cache coherence protocol. Unfortunately, with this solution, each user core would have to understand all of the complexity of the messages flowing inside of the Scalable Cache Coherent memory system. Finally, Option C provides a Private Cache to the user core, but the Level 1 cache is still located inside of the core. More importantly, with Option C, the User core does not need to be aware of the coherence protocol and messaging types, but rather a simpler interface. This type of design allows designers to Bring Your Own Core (BYOC).

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

Figure 3: Message Types Between the Processor Core and BYOC Private Cache

For the Scalable Cache Coherence Memory system, we use Option C and have defined a standard interface between the L1 cache and the private cache which we call the Transaction-Response Interface (TRI). This interface is simple and has been extended to many different core types. The TRI interface requires the cores to write through to the private cache. The Figure above shows the message types between the BYOC Private Cache and the Core being added to the Scalable Cache Coherent Memory system.

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)

Figure 4: Atomic Memory Types Supported by the Scalable Cache Coherent Memory System

One of the more challenging issues with creating such an interface was making atomic operations function correctly as there is a diversity of different atomic operation types in different processors. The figure above shows the types of atomic operations supported.

The Scalable Cache Coherent Memory system provides a total store order (TSO) memory model but can support cores which use weaker memory models. A detailed discussion of the issues involved with creating a memory interface that works for many different core types was presented at the ACM International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS) 2020 [1].

Using the TRI interface and the Scalable Cache Coherent Memory system, we have demonstrated 10 different core types and multiple ISAs connecting into the coherent memory system.

3.2 PyOCN: Open-Source On-Chip Network Generator

(https://github.com/cornell-brg/pymtl3-net)

PyOCN is a unified framework that vertically integrates multiple methodologies to enable productively exploring the design space of on-chip networks required for the open-source scalable cache coherent memory system described in the previous section. PyOCN is the first comprehensive framework for modeling (e.g., functional-level, cycle-level, and register-transfer-level), testing (e.g., unit testing, integration testing, and property-based random testing), and evaluating (e.g., simulating, generating, and characterizing) on-chip interconnection networks. PyOCN is built on top of PyMTL3 [2], a Python-based framework for hardware modeling, generation, simulation, and verification.

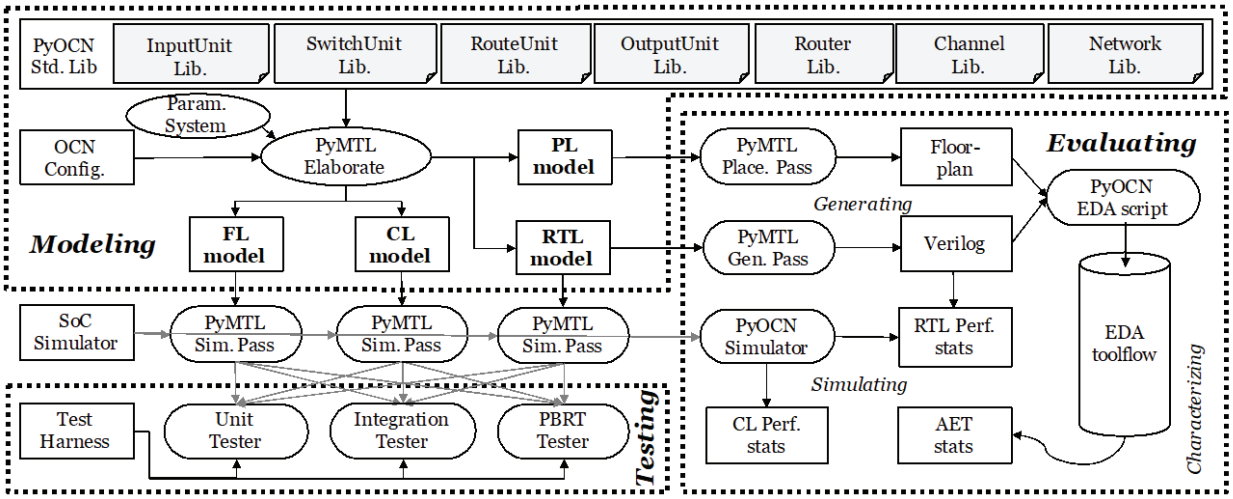

Figure 5: Overview of PyOCN Framework

The above figure illustrates the PyOCN framework. PyOCN provides a library of OCN components to compose networks. PyOCN also provides unit test for each building block and integration test for the target OCNs. Property-based random test (PBRT) is used for stress testing network models. An OCN simulator is implemented for simulating different OCNs and backend scripts are provided to drive the EDA toolflows for area, energy, and timing characterization. PyOCN also features a parameterization system facilitating easy OCN configuration. Green components are included as part of the PyMTL3 framework. Orange components are added as part of the PyOCN framework.

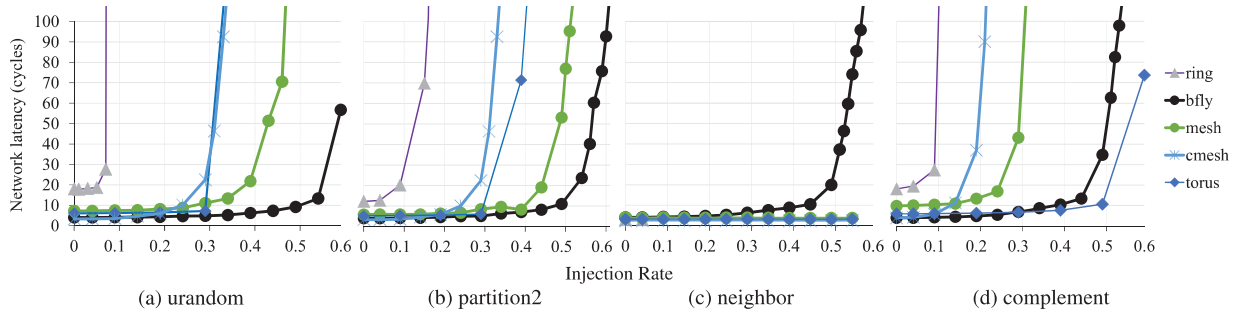

Figure 6: RTL Simulation Results Using PyOCN

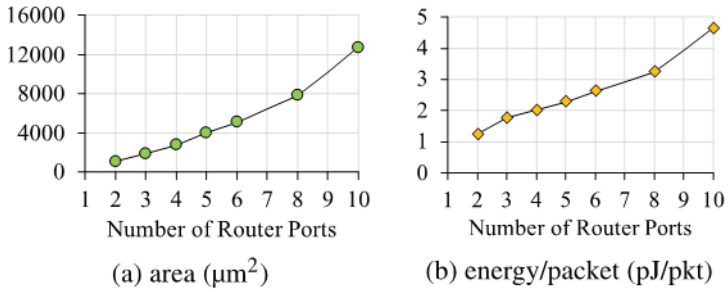

Figure 7: Router Characterization Using PyOCN

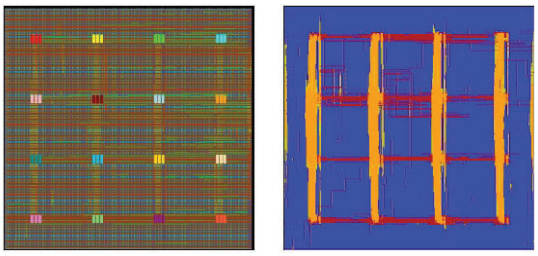

Figure 8: Post PnR Layout of 4-ary 3-fly Butterfly OCN Generated Using PyOCN

The above figures illustrate the kind of analysis enabled by the PyOCN framework spanning simulation, generation, and characterization of on-chip networks. Figure 6 shows average latency at different injection rates across different network topologies with 64 terminals using RTL *simulation*. Figure 7 shows how PyOCN can be used to first *generate* many OCN routers and then *characterize* the area and energy of these OCN routers with different number of input and output ports targeting a 500 MHz clock frequency using the FreePDK45 technology. Figure 8 shows post place-and-route layout for a 4-ary 3-fly butterfly topology which can be used to *characterize* the area, energy, and cycle time of the complete OCN. A paper [3] describing PyOCN was published in the 37th IEEE International Conference on Computer Design (ICCD).

PyOCN also enabled exploring on-chip interconnection networks for large-scale manycore processors with hundreds of cores. These manycore processors largely rely on high-diameter mesh on-chip networks (OCNs) without complex flow-control nor custom circuits due to three reasons: (1) manycores require simple, low-area routers; (2) manycores usually use standard-cell-based design; and (3) manycores use a tiled physical design methodology. PyOCN enabled exploring mesh and torus topologies with internal concentration and/or ruche channels that require low area overhead and can be implemented using a traditional standard-cell-based tiled physical design methodology. We were able to use PyOCN to ensure these networks were suitable for a tiled physical design methodology meaning they: (1) tile a homogeneous hard macro across the chip; (2) implement chip top-level routing between hard macros via short wires to neighboring macros; and (3) use timing closure for the hard macro to quickly close timing at the chip top-level.

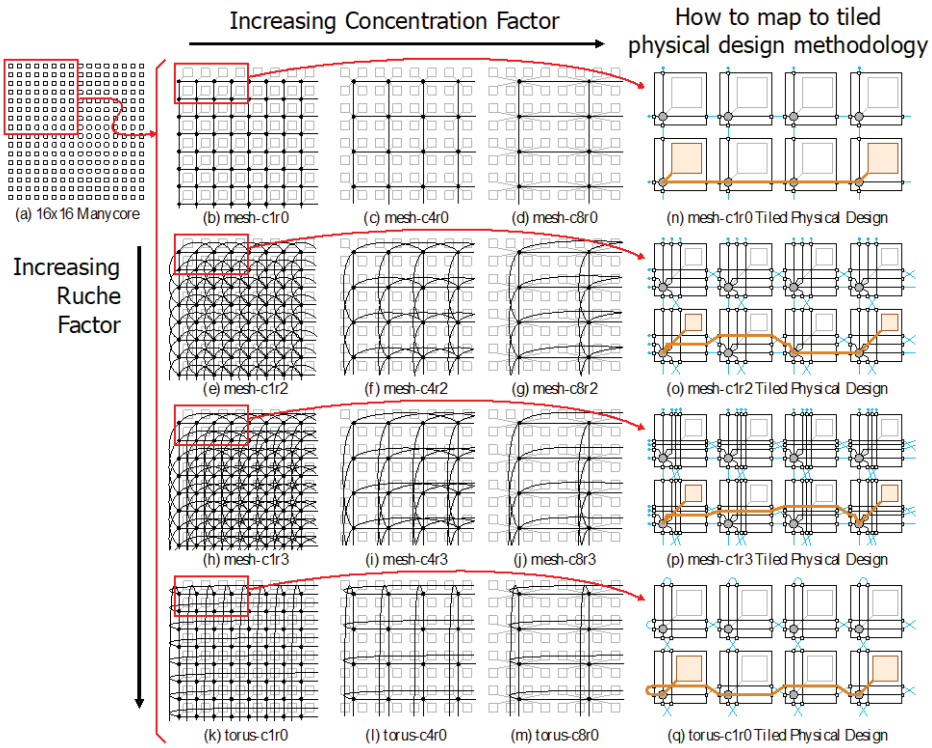

Figure 9: 12 Topologies Implemented Using a Tiled Physical Design Methodology

The above figure shows 12 topologies evaluated using PyOCN using a tiled physical design methodology suitable for a 16x16 manycore processor. These topologies include increasing concentration and increasing ruche factors. The above figure also shows how we were able to use the same homogeneous hard macro to implement each of these topologies.

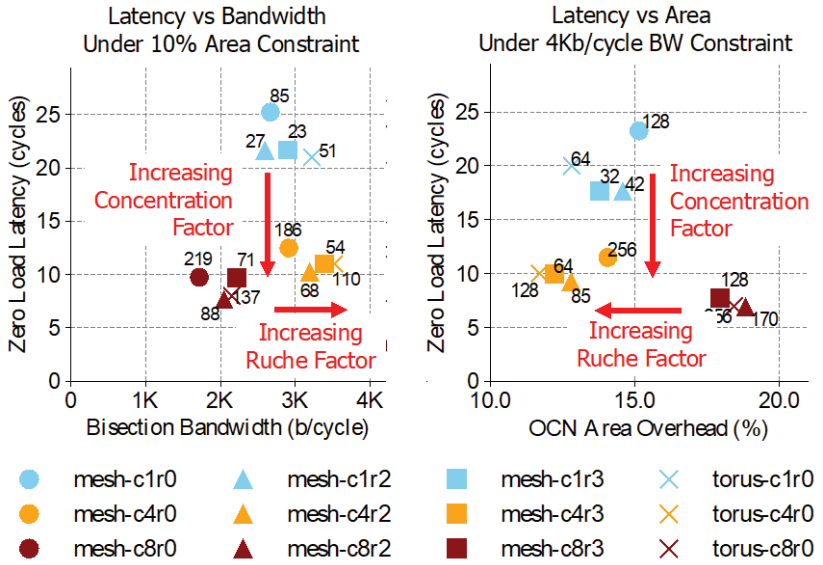

Figure 10: Area, Latency, Bandwidth Trade-offs for Various OCN Topologies

The above figure shows analytical modeling results for these topologies. Key take-aways are that a moderate concentration factor reduces latency at similar bandwidth and area, while moderate ruche factors can improve bandwidth and/or reduce area.

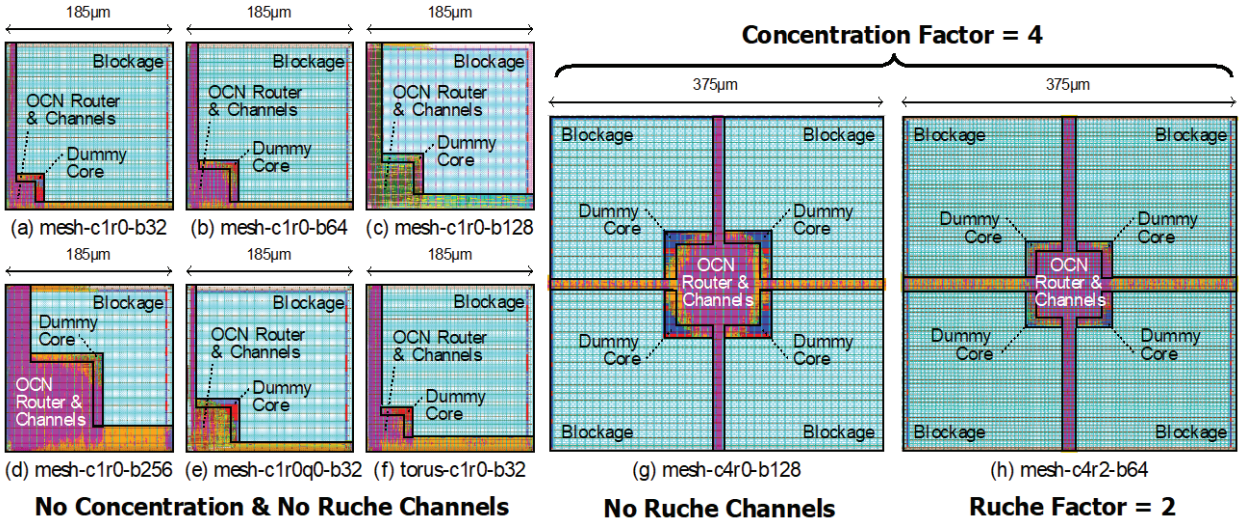

Figure 11: Example Macro-Level Post-Place-and-Route Layouts

The above figure shows the hard macros we generated and pushed through a commercial EDA toolflow using PyOCN. The insights from this design-space exploration using PyOCN helped inform the CIFER chip taped out later in the project. A paper [4] describing how to implement low-diameter on-chip networks for manycore processors using a tiled physical design methodology was published in the 14th ACM/IEEE International Symposium on Networks-on-Chip (NOCS).

3.3 PyH2: Property-Based Random Testing for Open-Source IP

PyH2 combines PyMTL3, a Python hardware modeling, generation, simulation, and verification framework, with Hypothesis, a Python property-based random testing framework, to enable productive testing of open-source hardware. Traditional open-source testing methodologies usually use a mix of completely random testing (CRT) and iterative deepening testing (IDT). CRT can detect errors quickly because it randomly samples the input space, but can produce very complicated failing test cases which are difficult to debug. IDT finds bugs more slowly because it gradually samples the input space, but can produce simple counter examples. Property-based testing (PBT), first popularized by QuickCheck, is a high-level, black-box testing technique where one only defines *properties* of the program under test and uses *search strategies* to create randomized inputs. Properties are essentially partial specifications of the program under test and are more compact and easier to write and understand than full system specifications. Users can make full use of the host language when writing properties and thus can accurately describe the intended behavior. Most PBT tools support *shrinking*, a mechanism to simplify failing test cases into a minimal reproducible counterexample. With these features, PBT can achieve the benefits of both CRT and IDT, and PyH2 applies these benefits to open-source hardware.

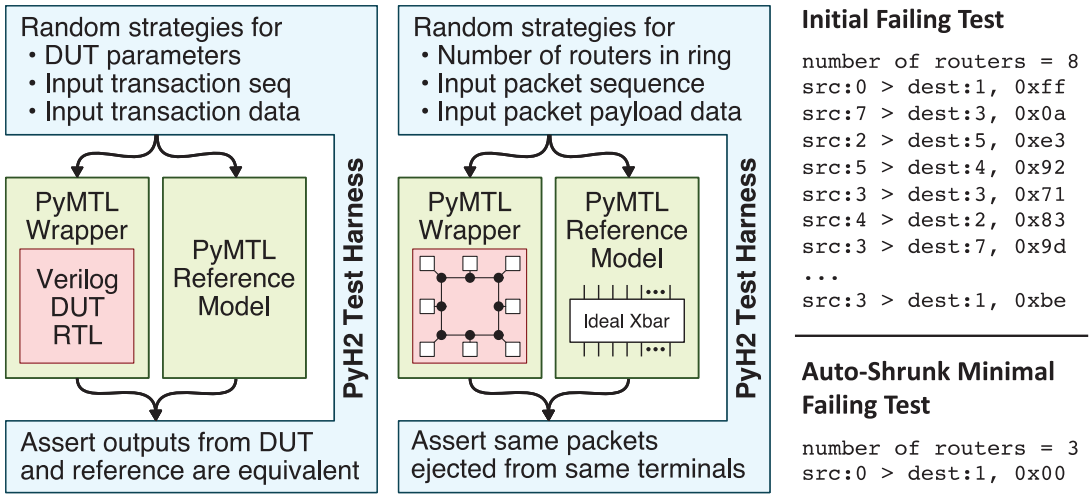

Figure 12: PyH2 Example for Testing On-Chip Network

The above figure illustrates the overall approach for using PyH2 to test open-source hardware. In this specific example, PyH2 is being used to test a ring on-chip. The PyH2 test harness uses Hypothesis strategies to create a sequence of network input packets that are then sent to both the design under test (DUT) as well as a PyMTL3 functional-level (FL) reference model. Network output packets from both the DUT and FL reference model are compared. If the response packets do not match, then Hypothesis can automatically shrink the failing sequence of packets to find a more minimal failing sequence. This automatic shrinking is key to improving testing productivity. A unique feature of PyH2 is it shrinks the input transaction sequence and the design itself; the framework attempts to reproduce a failing test case with the minimal failing sequence *and* the simplest possible design instance. In this specific example, PyH2 can shrink the number of nodes in the ring network.

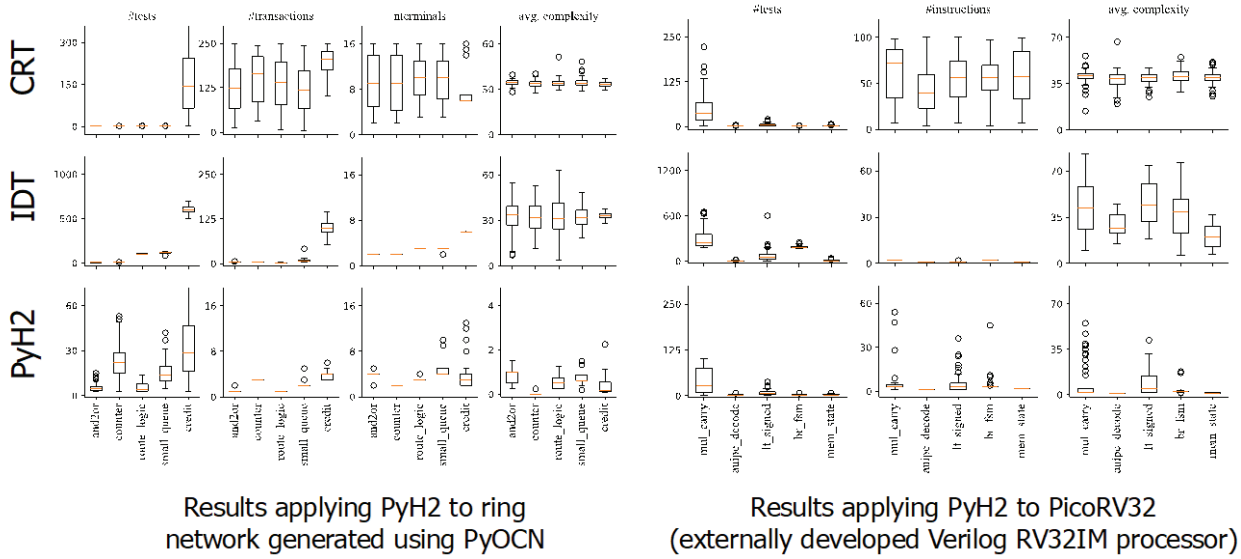

Figure 13: Experimental Results from Applying PyH2 to Ring Network and PicoRV32 Processor

The above figure shows results when applying PyH2 to both a ring network generated using PyOCN as well as an externally developed Verilog RV32IM processor. We manually injected a variety of representative bugs. After injecting a bug, we applied CRT, IDT, and PyH2. We ran 50 trials for each bug, and the results are shown as box-and-whisker plots. Overall, PyH2 detects errors quickly with a small number of test cases and produces a simple failing test case that has a short sequence of transactions, simpler transactions (shown as average complexity in the above results), and a simpler design instance (i.e., smaller number of terminals/nodes in the ring network). We have applied PyH2 to a variety of open-source hardware including complex data structures (reorder buffers, queues), pipelined cache implementations with software-centric cache coherence which were eventually integrated into the CIFER tapeout, and an externally developed AXI-4 adapter. A paper [5] describing our work on PyH2 was published in *IEEE Design & Test* in 2021.

3.4 PRGA: Open-Source FPGA Generator

(https://github.com/PrincetonUniversity/prga)
(https://prga.readthedocs.io)

The Princeton Reconfigurable Gate Array (PRGA) is an open-source FPGA platform. PRGA generates Verilog RTL for the desired FPGA as well as creates the inputs needed to feed open-source place-and-route tools to enable designs to be mapped onto the PRGA-generated FPGA. PRGA is easy to use and uses a modularized, extensible, Python API to configure the desired FPGA. PRGA is designed to be highly customizable and is scalable enabling the creation of different sized FPGA designs. PRGA has been integrated together with our Scalable Cache Coherent IP block enabling the FPGA to be communicated with in a cache-coherent manner. PRGA enables the integration of other IP blocks into the FPGA fabric. For instance, blocks like Block RAMs (SRAMS) or DSPs are easily integrated.

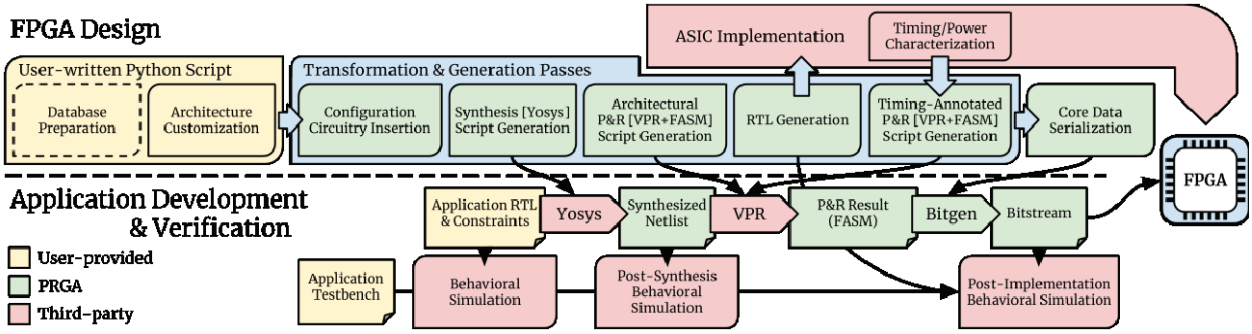

Figure 14: Overview of the PRGA Workflow
Figure from our FPGA 2021 Paper [6].

The figure above shows the flow of how PRGA works. The user writes a Python script which defines the desired parameters for the to-be generated FPGA. Next the core PRGA scripts generate Verilog Register Transfer Language (RTL) which then is passed to ASIC implementation tools and can ultimately be taped-out. We have shown that this output is suitable to be taped-out in the CIFER Chip tape-out. Equally important to the RTL, a set of tools are needed to enable the FPGA user to map designs onto the FPGA. In order to support this, PRGA

generates the needed configuration (XML) files for VPR [7]. To accurately model the highly customizable routing resources that PRGA supports, PRGA generates the Routing Resource Graph XML in addition to the Architecture Description XML for use in VPR. VPR is an open-source FPGA place-and-route tool created by the University of Toronto. In addition to the inputs generated for VPR, PRGA also generates a description of how the different portions of the PRGA-generated FPGA can be configured. PRGA then uses the open-source FASM tool for bitstream generation. The bitstream is the configuration data for the resulting FPGA.

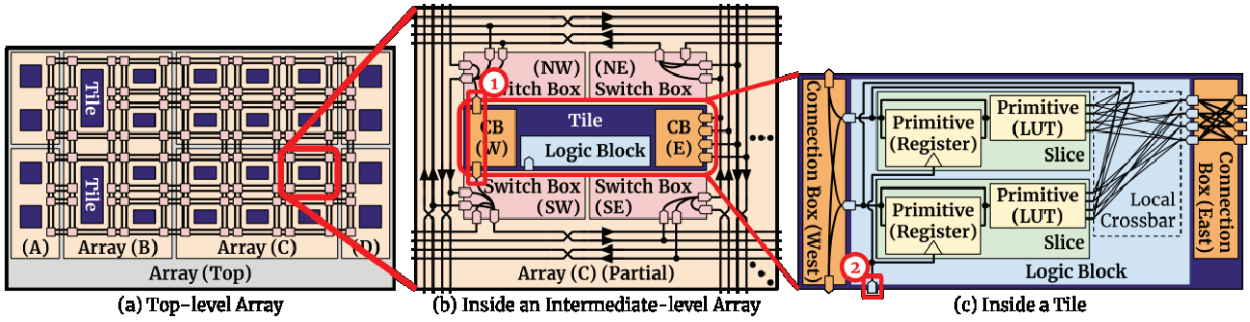

Figure 15: PRGA Hierarchy

Figure from our FPGA 2021 paper [6].

The figure above shows the logical design hierarchy for designing FPGAs supported by PRGA. The top level of the hierarchy is a 2-dimensional Array composed of Tiles, Switch Boxes, and nested Arrays. Each Tile contains one Logic Block or potentially multiple IO Blocks. The Tile also contains a varying number of Connection Boxes. Logic Blocks and IO Blocks are made up of Slices and Logic Primitives. For instance, the right most Tile contains two slices where each Slice contains a register (flip-flop) and a LUT (used to implement combinational logic in FPGAs).

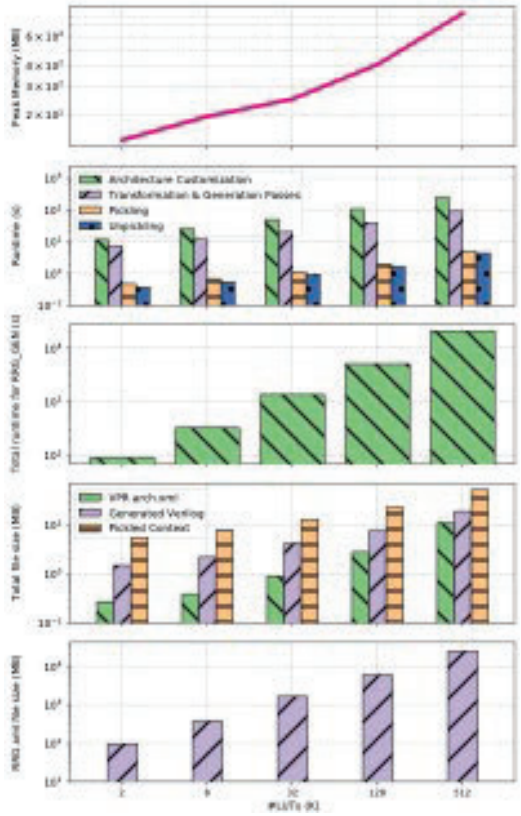

Figure 16: Scalability and Performance of the PRGA Tools
Figure from our FPGA 2021 paper [6].

The figure above shows the scalability and runtime of the PRGA framework for different FPGA designs. As can be seen, the file size, memory footprint, and tool runtime scale roughly linearly with the design size being created. For a relatively large half-million-gate architecture, PRGA uses less than 1GB of memory and runs within five minutes on a standard server computer, therefore PRGA is practically useful and scalable in terms of runtime. The size of the Routing Resource Graph (RRG) grows beyond gigabytes as the FPGA size grows. This is a file format dictated by the VPR tool and our team has provided feedback to the VPR maintainers and they are planning to update the format to be smaller in size. A presentation on PRGA and paper [6] detailing PRGA was presented and published in the proceedings of at the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA) 2021. Significant documentation for PRGA has been created (https://prga.readthedocs.io) and two online tutorials have been released for PRGA (https://prga.readthedocs.io/en/latest/tutorial/index.html)

While developing PRGA, the CIFER team made the observation that to make an easy to adopt FPGA generator, that the design needs to be easy to move between semiconductor process nodes. Traditional FPGAs require extensive physical design which limits their embedded use in future SoCs, makes them difficult to port to new processes, and the use of hardened FPGA macros negate the possibility of customizing the FPGA fabric for specific applications. Typical FPGAs typically have highly configurable routing between different slices/LUTs and those routing graphs typically have cycles (loops) in them. Because they have cycles, out of the box static timing analysis tools and other electronic design automation tools have difficultly timing the

circuits. In order to address this challenge, the CIFER team proposed using a novel cycle-free FPGA routing algorithm.

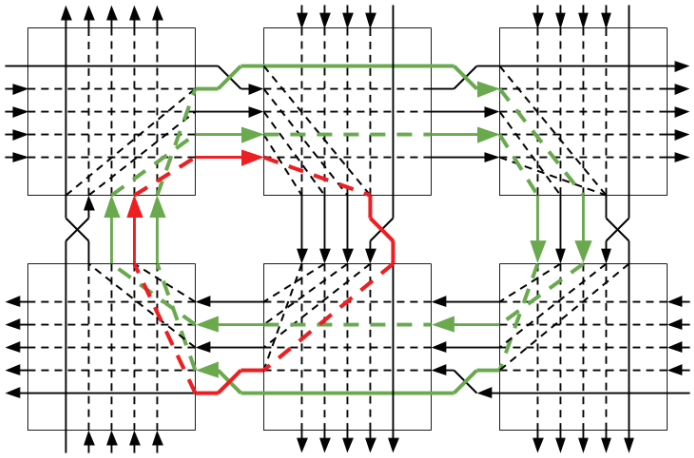

Figure 17: Green and Red Paths show Cycles on typical FPGA Designs

By modifying the default FPGA routing to not have cycles (acyclic) outside of a slice, off the shelf ASIC place and route tools can place the FPGAs. Likewise, off the shelf static timing analysis tools can analyze acyclic FPGA routing. PRGA supports both cyclic and acyclic FPGA routing, but to enable ease of automated physical implementation uses acyclic. The CIFER Chip uses an acyclic routing design.

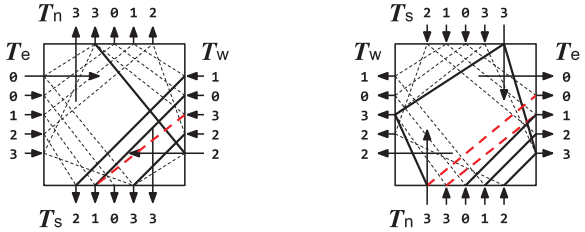

Figure 18: Logical Ranking of an Acyclic Wilson Switch Block

Connections that are removed are shown in dashed red lines. Figure from our FPL 2020 paper [8].

In order to create acyclic designs that do not hurt routability, we had to come up with an algorithm which judiciously removed connections. In order to do this, we used inspiration from the Turn Model [9]. The Turn Model is a model used to detect cycles and hence deadlocks in on-chip networks. The insight is that by limiting certain turns, cycles can be probably prevented. Leveraging this insight and applying it to FPGAs, our algorithm logically ranks routing tracks and then selectively removes switch block connections. This is done by disallowing connections from higher rank to lower ranked routing tracks and disallowing north to east and west to south turns. This has been shown to break cycles in the turn model and in effect turns the routing into spirals instead of loops.

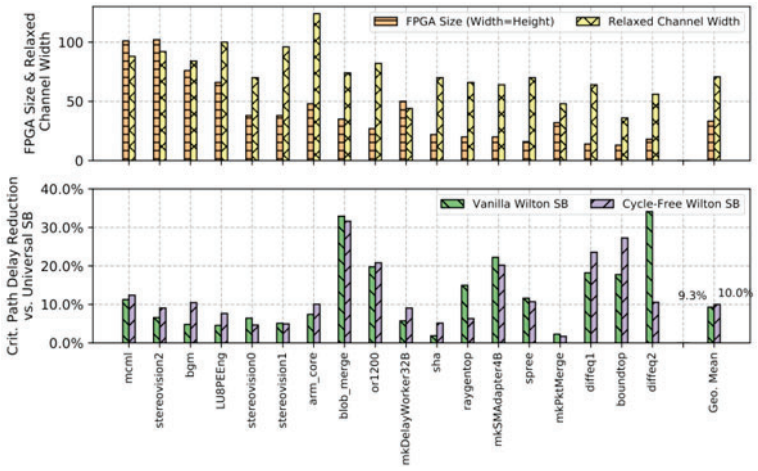

Figure 19: Routing Results

The bottom shows that the delay when comparing a traditional design with cycles versus a cycle-free (acyclic) design have similar delay. Figure from our FPL 2020 paper [8].

We quantitatively compared cyclic versus acyclic designs and found that acyclic switch blocks have no negative impact on the routability of the FPGA routing graphs. In the above figure, we show that a cycle free design has similar delay as a traditional switch block. The acyclic routing algorithm was presented at and published [8] in the proceedings of the International Conference on Field-Programmable Logic and Applications (FPL) 2020.

3.5 CIFER Chip: Heterogeneous Multicore/eFPGA SoC

An important component of having high quality open-source IP blocks is to make sure that those IP blocks are built in a manner that is physical design aware. Simply expressing a hardware design in Verilog RTL does not guarantee that it is implementable or efficiently implementable when built in a silicon implemented SoC. In order to demonstrate that the IP blocks designed in the CIFER program can be efficiently synthesized, that they are physical design aware, and that they can be implemented in actual chips, the CIFER team designed and had fabricated the CIFER Chip. By silicon validating our POSH IP, the CIFER team was able to understand and incorporate physical design issues into the open-source IP blocks. Likewise, others are more likely to use the CIFER team's IP blocks if they have been silicon validated.

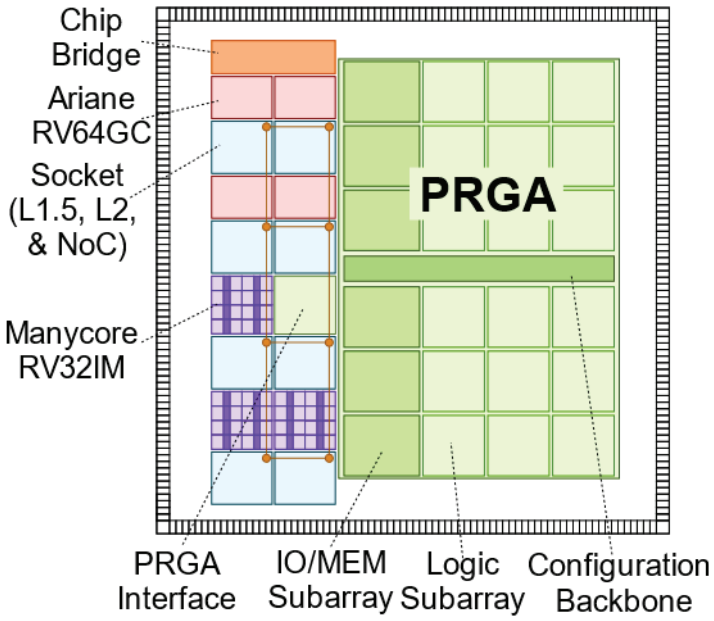

Figure 20: Block Diagram of the CIFER Chip

The CIFER Chip is a Heterogeneous Multicore/eFPGA chip which tightly couples an embedded FPGA fabric together with two different processor types. It contains a sizable PRGA-based FPGA together with four open-source Ariane 64-bit RISC-V processor cores together with 18 simpler 32-bit RISC-V cores. This design uses the Scalable Cache Coherent System and uses PyOCN to connect the different blocks together.

The CIFER chip was taped-out in GlobalFoundries 12nm process on an Oct/Nov 2020 multi-project wafer run. The chip die is 4mm x 4mm and is packaged in a 208-pin ceramic quad flat pack package. As shown in the block diagram above, the CIFER chip contains a 2x4 2D mesh (left side of figure) of tiles where the on-chip network is composed of three physical, bi-directional, on-chip networks. In the top left, are four Ariane processor cores. Ariane is a 64-bit RISC-V processor which contains a 16KB L1 instruction cache and an 8KB L1 data cache. In the bottom left, there are three Manycore clusters where each Manycore (TinyCore) cluster contains six 32-bit RISC-V cores where each core contains a 4KB L1 Data cache and teach pair of cores share a 4KB L1 instruction cache. Each of the eight tiles contains a socket which contains the Network-on-Chip (NoC) together with a private 8KB cache and a 64KB slice of the shared last-level-cache (LLC). All-together the CIFER chip has 512KB of shared last-level-cache. There is a PRGA interface tile which coherently interconnects the PRGA-based eFPGA into the Scalable Cache Coherent fabric as well as containing some auxiliary logic like the chip's interrupt controller.

The PRGA-generated embedded FPGA on the CIFER chip has 6720 multi-mode 6-input LUTs. The eFPGA utilizes the cycle-free routing topology developed as part of this project. Also, the eFPGA has asynchronous FIFOs for communication to and from the FPGA fabric enabling the eFPGA to run at a differing clock frequency than the processor core frequency.

The CIFER team has successfully tested the CIFER chip at our hardware lab in Princeton. We are pleased to report that the CIFER chip has been accepted for presentation and inclusion in the proceedings of the 2023 IEEE Custom Integrated Circuits Conference (CICC).

4 DISCUSSION

4.1 Impact and Workforce Development

Overall, the POSH program has created many exciting open-source IP blocks with the CIFER team creating two main IP blocks. It has also trained many researchers to create high-quality open-source IP blocks and created a community around open-source hardware IP blocks. The CIFER team recommends using Integration Exercises for future DARPA MTO programs. In addition, as a university team, the CIFER team brought graduate student engineers and postdocs to the Integration Exercises. The CIFER students enjoyed the Integration Exercises and had opportunities to increase their professional networks by meeting and working with graduate students from other universities which they would otherwise have been unlikely to experience in their graduate studies. The Integration Exercises have contributed to developing the workforce of future engineers and has helped foster students who are excited about working on hardware and semiconductors. We see the excitement around chip development growing in our own institution and see the IP blocks developed in CIFER as blocks that can be leveraged within our own institutions and at other universities for education. Last, the excitement around open-source hardware continues to grow and we hope that the CIFER work helps grow the eco-system around open-source hardware.

4.2 Challenges

One of the major challenges encountered during the POSH program by the CIFER team is that of the outbreak of the global COVID-19 pandemic during the middle of the program. This had multiple impacts on the CIFER team. First, the physical labs at Princeton University and Cornell University were closed for multiple months due to institutional and state orders and the respective state-level work-from-home orders had a negative impact on the productivity and spending of the team. Second, the COVID-19 pandemic made it more difficult to interact with other POSH and IDEA team performers who had their own delays and the integration exercises during the pandemic were made virtual which was less productive than in-person integration exercises. Also, the respective travel restrictions by the universities and state made in-person interaction and traveling to in-person events not possible during parts of the pandemic which made it more difficult to disseminate the results of our research.

Another major challenge faced by the CIFER team was that of the global semiconductor chip supply chain challenge (chip shortage) of 2021/2022 and associated packaging capacity shortages. In this project, we taped out the CIFER chip in late Fall 2020 with chips coming back in 2021 which was during the chip shortage. The most notable impact our team experienced was when we were trying to get the CIFER chip packaged. Our original chip packaging vendor was full and was not taking prototype packaging requests and we tried another vendor which was also full for a year. We ended up finding a third vendor which ultimately packaged the CIFER chip, though this did add delay to the chip packaging. Likewise, the chip shortage made it difficult to purchase test equipment and test boards to test our CIFER chip.

5 CONCLUSIONS

In conclusion, we believe that the DARPA supported POSH program will have a significant positive impact on the open-source hardware community. We look forward to people using the Scalable Cache Coherent Memory System, PyOCN, and PRGA and potentially being used in future DoD chips. The CIFER Chip has silicon-validated these designs and the team has learned much from the design and testing of the CIFER Chip.

We look forward to seeing the IP blocks and tools created in the CIFER program have a large impact on future chip designs, future large scale DoD SoCs, and a continuing positive impact on the semiconductor industry.

6 REFERENCES

[1] J. Balkind, K. Lim, M. Schaffner, F. Gao, G. Chirkov, A. Li, A. Lavrov, T. M. Nguyen, Y. Fu, F. Zaruba, K. Gulati, L. Benini and D. Wentzlaff, "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*, 2020.

[2] S. Jiang, P. Pan, Y. Ou and C. Batten, "PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification," *IEEE Micro,* vol. 40, no. 4, pp. 58-66, 2020.

[3] C. Tan, Y. Ou, S. Jiang, P. Pan, C. Torng, S. Agwa and C. Batten, "PyOCN: A Unified Framework for Modeling, Testing, and Evaluating On-Chip Networks," in *IEEE International Conference on Computer Design*, 2019.

[4] Y. Ou, S. Agwa and C. Batten, "Implementing Low-Diameter On-Chip Networks for Manycore Processors Using a Tiled Physical Design Methodology," in *ACM/IEEE International Symposim on Networks-on-Chip*, 2020.

[5] S. Jiang, Y. Ou, P. Pan, K. Cheng, Y. Zhang and C. Batten, "PyH2: Using PyMTL3 to Create Productive and Open-Source Hardware Testing Methodologies," *IEEE Design & Test,* vol. 38, no. 2, pp. 53-61, 2021.

[6] A. Li and D. Wentzlaff, "PRGA: An Open-Source FPGA Research and Prototyping Framework," in *29th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021.

[7] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent and V. Betz, "VTR 8: High-performance CAD and Customizable FPGA Architecture Modelling," *ACM Transactions on Reconfigurable Technology and Systems,* vol. 13, no. 2, pp. 1-55, 2020.

[8] A. Li, T.-J. Chang and D. Wentzlaff, "Automated Design of FPGAs Facilitated by Cycle-Free Routing," in *30th International Conference on Field-Programmable Logic and Applications*, 2020.

[9] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *Journal of the ACM,* vol. 41, no. 5, pp. 874-902, 1994.

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

ACRONYM	DESCRIPTION
ACM	Association for Computing Machinery
AFRL	Air Force Research Laboratory
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
BYOC	Bring Your Own Core
CAD	Computer Aided Design
CLB	Combinational Logic Block
CRT	Completely Random Testing
DARPA	Defense Advanced Research Agency
DUT	Design Under Test
EDA	Electronic Design Automation
FL	Function Level
FPGA	Field Programmable Gate Array
IDEA	Intelligent Design of Electronic Assets
IEEE	Institute of Electrical and Electronics Engineers
IO	Input Output
IP	Intellectual Property
ISA	Instruction Set Architecture
LLC	Last Level Cache
LUT	Look Up Table
MTO	Microsystems Technology Office
NoC	Network on Chip
OCN	On-Chip Network
PBRT	Property Based Random Testing
PDK	Physical Design Kit
POSH	Posh Open Source Hardware
PRGA	Princeton Reconfigurable Gate Array
RAM	Randomly Accessible Memory
RTL	Register Transfer Language
SAR	Successive Approximation Register
SIGDA	Special Interest Group for Design Automation
SoC	System-on-Chip
SRAM	Static Randomly Accessible Memory
STA	Static Timing Analysis
TRI	Transaction-Response Interface
XML	eXtensible Markup Language