Constellation: An Open-Source SoC-Capable NoC Generator

Jerry Zhao
U.C. Berkeley
Berkeley, CA 94720
jzh@berkeley.edu

Animesh Agrawal
U.C. Berkeley
Berkeley, CA 94720
animesha@berkeley.edu

Borivoje Nikolic
U.C. Berkeley
Berkeley, CA 94720
bora@berkeley.edu

Krste Asanović
U.C. Berkeley
Berkeley, CA 94720
krste@berkeley.edu

2022 15th IEEE/ACM International Workshop on Network on Chip Architectures (NocArc) | 978-1-6654-5589-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/NOCARC57472.2022.9911299

***Abstract*—In response to growing application diversity, System-on-Chip (SoC) architectures have become increasingly heterogeneous, with diverse cores and accelerators, as well as non-uniform memory systems. However, existing open-source design frameworks for SoCs and NoCs (Network-on-Chips) have been unable to facilitate design exploration of heterogeneous SoC architectures with irregular NoCs. We present Constellation, a new NoC RTL generator framework designed from the ground up to support integration in a heterogeneous SoC and evaluation of highly irregular NoC architectures. Constellation implements a highly decoupled specification system that allows an architect to specify an exponentially large design space of irregular virtual-channel wormhole-routed NoC architectures. Additionally, Constellation provides a diverse set of systems, regression tests, and evaluation tools to provide confidence in the correctness and performance of the generated hardware. Constellation is open-sourced and integrated into the Chipyard SoC design framework, allowing full-system exploration of heterogeneous SoC architectures with irregular memory fabrics.**

***Index Terms*—network-on-chip, system-on-chip, open-source**

I. Introduction

As workloads diversify and general-purpose performance scaling wanes, architects have turned to specialization and heterogeneity to meet targets for power, performance, and area. A modern SoC might integrate a heterogeneous core architecture with dozens of specialized compute units, including GPU, NPU, DSPs, ISPs, and IO processors [1]–[7]. This extreme heterogeneity has become prevalent across a wide set of deployment scenarios, from mobile to cloud, and we expect this trend to continue to dominate SoC architectures well into the future.

The dominance of heterogeneous compute architectures suggests the importance of heterogeneous or irregular memory architectures as well, to address both application-specific memory access patterns and physical design constraints. Recent research has re-examined longstanding assumptions about memory architectures and proposed novel approaches for memory protocols, cache coherence, memory organization, and data movement.

Research was partially funded by NSF CCRI Award #2016662 and partially by SLICE Lab industrial sponsors and affiliates. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

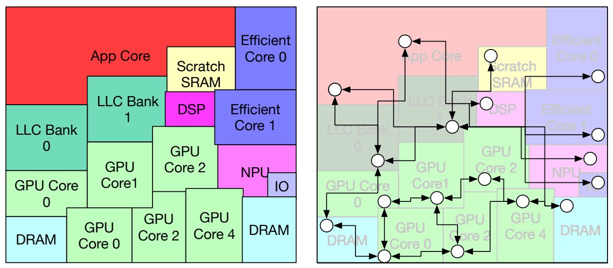

Fig. 1. Physical design and floorplanning concerns in heterogeneous SoCs preclude the implementation of simple, regular NoC architectures.

Open-source SoC design frameworks are popular tools for researchers studying heterogeneous architectures. These frameworks package libraries of SoC components, including cores and memory systems, into highly parameterized SoC generators. By generating physically realizable RTL, these SoC generators enable full system evaluation of power, performance, and area, as architects can evaluate novel SoCs using FPGA prototyping, physical design flows, taped-out test chips, and real software workloads.

While these SoC design frameworks have been successful at modeling compute heterogeneity, we observe that they have generally not allowed significant customization of the NoC component in an SoC. These frameworks generally assume a static, regular NoC topology or crossbar, with perhaps limited support for scaling the dimensions of the network. While suitable for studying homogeneous multi-core architectures, we believe that existing frameworks cannot accurately describe NoC architectures for extremely heterogeneous systems.

Highly flexible NoC generators have been developed for isolated evaluation of diverse NoC designs [8]–[11]. However, the challenges of SoC integration have prevented the inclusion of these systems into existing SoC design frameworks. These NoC generators often implement only a simple idealized network protocol.

To address these limitations, we developed Constellation, a highly parameterized RTL generator for virtual-channel wormhole-routed NoCs [12]. Constellation is protocol-agnostic, yet capable of supporting diverse protocols, including standard cache-coherence protocols. To support describing a large design space of irregular NoCs, Constellation provides

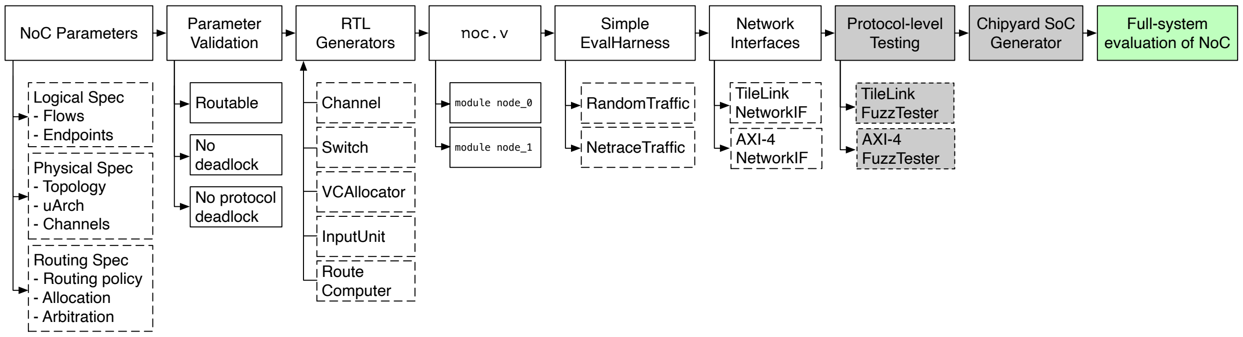

Fig. 2. Flow diagram for deploying the Constellation framework. White boxes indicate Constellation-specific components. Dotted lines indicate standard interfaces for configuring or customizing Constellation.

Listing 1. Parameterizing the NoC depicted in Figure 3. Note that the decoupled functional high-level specification is concise, yet highly expressive.

```
val myNoCParams = NoCParams(
# Logical specification
  flows = Seq.fill(3, 3) { case (i,j) =>
    Flow(ingress=i*2, egress=j*2+1, vnet=0) },
  vnetBlocking = ...,
# Physical topology specification
  ingresses = (0 until 6 by 2).map { i =>
    Ingress(payloadWidth=64, node=i) },
  egresses = (0 until 6 by 2).map { j =>
    Egress(payloadWidth=64, node=j+1) },
  topology = (a: Int, b: Int) => {
    (a + 1) % 12 == b || (a + 2) % 12 == b },
  routers = (r: Int) => RouterParams(
    combineRCVA = r % 2 == 0 ),
  channels = (a: Int, b: Int) => {
    Channel(nVirtualChannels = 4) },
# Routing
  routing = (f: Flow, p: VCID, n: VCID) = {
    val dateline = p.node == 5 && n.vc >= p.vc
    val one_away = f.dst == (p.node + 1) % 6
    !dateline && !(one_away && n.node != f.dst) })
```

a modular specification system, separating specification of physical NoC resources from specification of logical endpoint behaviors and flow control. The generator additionally includes verification systems that can guarantee that important properties of the specified NoC are met.

Constellation[1] is open-sourced and integrated into the Chipyard SoC design framework, allowing for full-system evaluation of heterogeneous SoC architectures with irregular NoCs. Constellation also includes an extensible C++ framework for standalone evaluation of NoC performance, as well as a large suite of regression tests to ease microarchitechural exploration.

II. Generator Design

The software architecture for the Constellation generator is divided into three phases, depicted in the first three columns of Figure 2.

1) The **parameterization** phase decouples specification of physical NoC resources from specification of endpoint behavior and routing policy.
2) The **validation** phase constructs an abstract internal model of the NoC and validates that all requested properties of the NoC are achieved.
3) The **elaboration** phase passes the validated parameters to RTL generators for the NoC router components, generating complete RTL for the requested design.

[1]github.com/ucb-bar/constellation

A. Parameterization Phase

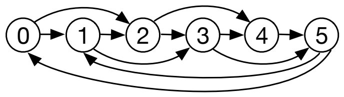

Fig. 3. The sample ring topology with skip-connections described by Listing 1

In the parameterization phase, the user specifies the desired configuration for the NoC using Constellation's specification system, expressed as Scala configuration objects. An example is shown in Listing 1. The aspects of NoC parameterization are roughly orthogonalized into three categories.

1) Logical specification: The logical behavior of a NoC describes the possible flows in the network, as well as potential dependencies between flows. For example, to avoid protocol deadlock, flows corresponding to one protocol channel may need to be nonblocking with respect to flows corresponding to another protocol channel. Constellation supports this by letting the user assign "virtual subnetwork" identifiers to all specified flows, as well as blocking/non-blocking properties across all requested virtual subnets.

TABLE I
Configuration options supported per-router or per-channel.

Field	Range	Default
merge RC and VA stages	T/F	F
merge SA and ST stages	T/F	F
add bypass from SA to VA	T/F	F
router payload width	integer	64
number of VCs	integer	4
buffer slots per VC	integer	4
channel-speedup	integer	1
input-speedup	integer	1
output-speedup	integer	1

2) Physical specification: The physical specification describes the requested network topology, represented as an arbitrary directed graph. Each node in the graph represents an individually configurable router. Each edge in the graph represents an individually configurable physical channel. A partial listing of frequently-used configuration options for routers, physical channels, and virtual channels is shown in Table I.

Notably, all of these parameters can be *individually specified per instance in the design*, allowing for a highly heterogeneous

NoC topology containing subgraphs with different bandwidth, latency, power, or area specifications. For instance, in the NoC shown in Listing 1, the even-numbered routers are configured differently from the odd-numbered routers.

3) Routing specification: Constellation defines the routing relation as:

$$R : F \times V \times V \rightarrow B \tag{1}$$

For a given flow $f \in F$ that currently occupies virtual channel $v \in V$, $R(f, v, v')$ returns a boolean $b \in B$ whether the policy will route f to virtual channel $v' \in V$. In Constellation, the NoC architect can leverage high-level features of the Scala programming language to specify this routing relation instead of directly describing the policy's hardware implementation. Additionally, a policy definition can reference pre-defined routing "sub-policies". For example, the policy definition for fully-adaptive minimal routing on a 2D mesh references separate sub-policies for escape routing, dimension-ordered routing on a 2D mesh, and minimal routing a 2D mesh.

This high-level routing specification system is especially useful for describing routing on an irregular topology, as the routing relation can be specified separately for subsets of the network.

B. Validation Phase

The parameter validation phase verifies that the requested logical behavior can be achieved on the specified topology, given the specified routing relation. Currently, this phase checks that the constructed network is deadlock-free, and fully routed (all flows end at the correct egress). Deadlock-freedom between virtual subnetworks, which is necessary for protocol-level deadlock-freedom, is also verified.

This phase constructs a virtual-channel dependency graph considering all flows and checks that there is no cyclic dependency [13]. For escape-channel based deadlock-freedom, this phase checks only fir the lack of cyclic dependencies for the subset of the graph encompassing the escape channels [14]. This phase also performs a limited degree of optimization, primarily by marking unused virtual channels or router nodes.

C. Elaboration Phase

The validated parameters are passed to the elaboration phase, which consists of Chisel generators for the various components of the network. The generator for the router nodes is fully generalized for arbitrary radix, topology, and routing algorithm. A unique router instance is generated for each node in the network.

The generator implements a conventional microarchitecture for virtual-channel routing, shown in Figure 4, with the default four routing stages being RC (route-compute), VA (virtual-channel-allocation), SA (switch-allocation), and ST (switch-traversal) [15]. The generator can be also configured to fuse the RC/VA stages, as well as the SA/ST stages, if desired, as depicted in Figure 5.

The route-computer in each router node computes the set of all possible virtual channels that a packet in the router node is allowed to allocate next in its wormhole-routed path

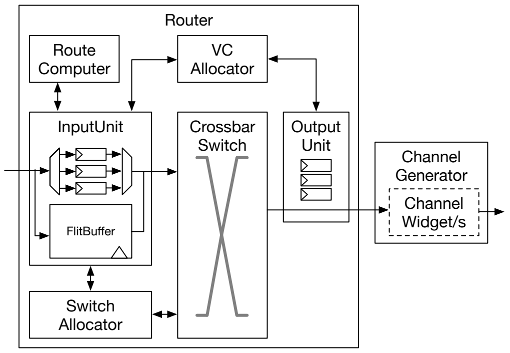

Fig. 4. Standard microarchitecture of a Constellation router. Note channel-widgets allows configurable insertion of clock-crossings/width-converters/buffers/monitors on each channel.

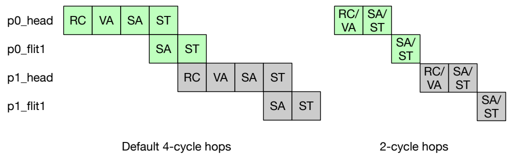

Fig. 5. Pipeline diagrams for min-2-cycle and max-4-cycle hop cases. Merging RC/VA is acceptable for systems with trivial routing policies. Merging SA/ST is acceptable for low-radix routers.

through the network. Formally, the route computer generates a hardware implementation of:

$$R_N : F_N, V_N, \hat{V}_N \rightarrow B \tag{2}$$

Where $F_N \subset F$ represents only the subset of flows that might arrive at router N, $V_N \subset V$ contains the virtual channels that inject to router N and $\hat{V}_N \subset V$ contains the virtual channels that eject from router N. Since F_N, V_N, and $\hat{V}_N$ are each known after the validation phase, the generator can generate the minimal hardware for correct point-to-point routing on the network by using the Quine-McCluskey method to generate the decode table.

For virtual-channel allocation, several different allocator generators are provided, including generators for iSLIP and PIM [16], [17]. The default router generator implements a crossbar switch. All the generators for the router components are fully generalized and can be used for any routing policy, for any NoC topology. The router generators can also take advantage of the optimization opportunities discovered by the parameter validation.

The resulting RTL is fully synthesizable, and can be passed to physical design tools, or coupled with synthetic traffic generators for standalone evaluation of the network.

III. SoC Integration

While the intent for Constellation was to support standard SoC memory transport protocol, it was designed to be protocol-independent. As a result, Constellation exposes only a very minimal packet interface at the endpoints, shown in Table II.

Higher-level protocols, such as the AMBA family or TileLink, typically are decomposed into channels, with each channel carrying some subset of message types [18], [19]. Furthermore, deadlock-freedom guarantees in the protocol are

TABLE II
ENDPOINT PACKET/FLIT INTERFACE.

Field	Width
head	1
tail	1
payload	configurable
ingress_id	log2(# ingresses)
egress_id	log2(# egresses)

only provided if non-blocking properties between channels are met. The complexity increases when recursive messages are present, as in the case where L1-to-LLC and LLC-to-DRAM messages share the same network resources.

The simple approach would be to route each channel on an independent network, while a more efficient implementation would try to multiplex several channels onto the same routing resources. Constellation supports both use cases, as it can be configured with an arbitrary number of virtual subnetworks, while each virtual subnetwork is mapped to an individual protocol channel.

Network interfaces convert a standard high-level protocol to the simple packet format in Constellation. These blocks convert the protocol-level routing information, typically in the form of an address, to the `egress_id` field in a Constellation packet. The protocol channel identifier is implicitly encoded in this field as well. All other fields in the protocol message can be packed in the `payload` field. The network interface can arbitrarily convert protocol messages to Constellation packets, as there is no limit on the length of a Constellation packet.

Notably, Constellation does not maintain ordering between packets corresponding to the same flow, breaking some of the ordering assumptions in protocols like AXI-4, where messages with the same source ID and destination remain ordered. Constellation assumes that endpoint buffering can recover ordering in such cases, and we note that recent revisions of the AMBA protocols suggest that endpoint reordering is expected in modern NoC implementations [20].

IV. USING CONSTELLATION

Listing 2. Configuring a heterogeneous SoC with a heterogeneous Constellation NoC by mixing provided specification fragments.

```
class HeteroSoCArchConfig(
  WithNGPUCores(6),
  WithNSmallCores(2),
  WithNBigCores(1),
  WithNMiniCores(0),
  WithNCacheBanks(2),
  WithNDRAMChannels(2),
  WithScratchpad)
class HeteroNoCSoCConfig(
  constellation.WithMapping(Map(
     "big_core" -> 4, "small_core0" -> 1, ...)),
  constellation.WithSBusNoC,
  constellation.WithNode(node=4, combineRCVA=true),
  constellation.WithChannel(src=0, dst=4, nVC=5),
  constellation.WithUniformChannels(nVC=4),
  constellation.WithRouting(Mesh2DEscapeRouting(4,4)),
  constellation.WithTopology(Mesh2D(4,4)),
  HeteroSoCArchConfig())
```

While Constellation can model highly irregular NoC architectures, it is designed to remain accessible to SoC architects without a deep background in NoC implementation. Although the detailed specification system allows for fine-grained control over the NoC generator, a library of existing definitions encodes the specification of common design points, as shown in Listing 2. Example topology and routing definitions for configurable torus, butterfly, tree, and mesh topologies are provided.

Specification fragments for router microarchitecture, channel parameterization, and legal flows are also provided. These fragments are generally orthogonal to each other, allowing them to be composed together to describe a very large design space of potential NoCs.

For SoC integration, protocol converters for AMBA AXI-4 and TileLink are provided, in addition to specification fragments that properly set up the virtual subnetwork mapping for protocol channels. While the AXI-4 and TileLink converters can be used to generate a standalone, protocol-compliant NoC, they can also integrate Constellation with Rocketchip's Diplomatic memory system graph framework. This enables plug-and-play integration into Chipyard or Rocketchip-based SoCs, replacing the existing crossbar-based interconnects in these systems with Constellation-generated NoCs. To support SoC-level evaluation of Constellation, we integrate Constellation into Chipyard, making it the default NoC implementation for all Chipyard-based SoCs.

A. *NoC Evaluation*

Although Constellation was designed to support full-system SoC-level evaluation, it still provides functionality for evaluating the NoC in isolation. An included C++ traffic injection and measurement framework integrates with an RTL simulation of a Constellation NoC through DPI calls. The tool models warmup, measurement, and drain phases of evaluation, and collects statistics on per-flow bandwidth and latency, similar to the Booksim SW modeling framework [21].

This tool can be extended to support a variety of different traffic pattern generators. Currently, the "Synthetic" model lets the user specify per-flow injection rates, while the "Netrace" model uses trace-driven dependency-graph models of NoC traffic from the Netrace framework [22].

For SoC-level evaluation, Constellation can generate per-physical-channel performance counters. These counters can be memory mapped and read by system software to monitor NoC behavior.

B. *NoC Testing*

To provide confidence in Constellation-generated designs, and to support the health of the project, Constellation includes a large battery of functional regression tests. A pull request to the Constellation repository triggers over 100 different regression tests across a wide set of configurations. Simple tests use a no-protocol traffic generator to load the network, confirm all packets are routable, and the network does not deadlock. AXI-4 and TileLink protocol-level tests confirm that the protocol interfaces correctly encode and decode protocol messages, and that the network is free from protocol deadlock.

Finally, performance tests verify that any requested desired throughput, median latency, and max latency are achieved.

Collectively, these tests run millions of cycles of NoC simulation on every pull request. In the development of Constellation, they were extremely useful at catching obscure RTL bugs or performance regressions, and we hope they will encourage future community contributions.

V. Use Cases

We discuss several representative use cases for Constellation, demonstrating independent NoC exploration and evaluation using Constellation's features, as well as full-system SoC-integrated evaluation.

A. *NoC Exploration*

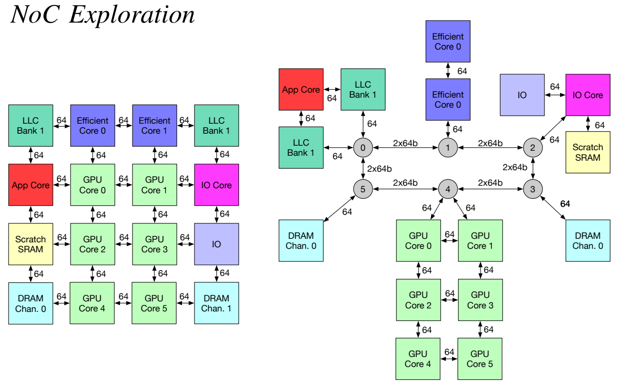

Fig. 6. Example NoC architectures for a hypothetical heterogeneous SoC. Both the regular 2D mesh NoC and the irregular NoC can be described.

We consider a hypothetical heterogeneous SoC architecture with a single high-performance application core, 2 low-power "efficiency" cores, a 6-core GPU complex, and an I/O processor core. The memory system of this SoC contains 2 banks of LLC cache, 2 DRAM channels, 1 incoherent scratchpad, and 1 I/O channel.

We first choose to map the SoC architecture to a conventional 4x4 2D mesh NoC, with adaptive-minimal routing, shown in Figure 6. Deadlock-freedom is provided by dimension-ordered escape channels. We configure Constellation to generate a NoC for this design using existing library specification fragments.

Figure 6 also shows an alternative hypothetical topology for this SoC. The memory system is distributed along a high-bandwidth ring, with subgraphs isolating some local traffic. Despite the changes, this topology is still logically equivalent to the regular mesh, with all managers accessible to all clients. We leverage the flexibility of the Constellation specification framework to rapidly describe the physical topology and routing algorithm of this new system. Defining the topology and routing together takes fewer than 100 lines of code, with the NoC framework automating much of the validation and RTL elaboration.

We approximate a traffic matrix for this system, and use Constellation's synthetic traffic evaluation tool to evaluate latency and under synthesized load. The traffic matrix used in this experiment approximates traffic in a heterogeneous architecture, where the cores access L2 at different rates, the I/O core predominantly acceses I/O, and DRAM is primarily

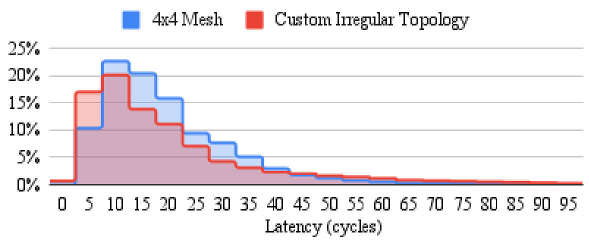

Fig. 7. Packet latency histogram comparing the two configurations in Section V-A.

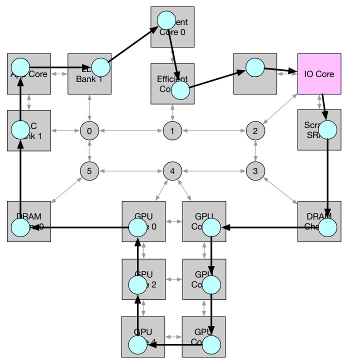

Fig. 8. A low-cost, low-bandwidth control NoC sits alongside the main data NoC in a hypothetical SoC.

accessed by the GPU cores and L2 banks. Figure 7 shows the results of simulation from the tool. Under our defined traffic matrix, the irregular topology reduces median latency using lower-radix routers, at the expense of tail latency.

B. *SoC Design*

Approximate RTL for the hypothetical SoC in Section V-A can be generated when running Constellation within the Chipyard SoC generator framework. We use a RISC-V BOOM core to represent the application core and RISC-V Rocket cores for the efficiency and IO cores [23]. We also use Rocket as a rough approximation for the GPU cores.

While the NoC in Section V-A provides low-latency, high-bandwidth data movement through the SoC, we observe that a low-power, latency-insensitive "control" NoC is useful for out-of-band system monitoring and configuration. Since Constellation is integrated as a generic NoC generator, we can use it to implement a narrow 16-bit wide ring "control" NoC alongside the main data NoC.

To demonstrate this, we run a simple test where the IO-core monitors NoC traffic, and throttles the efficiency cores if the traffic from the L2s exceeds some threshold. When software-managed throttling is enabled, the application-core memcpy throughput is 1.3x the case when throttling is disabled.

C. *Many-core Architectures*

Constellation can scale-out to implement many-core SoC NoCs. We configured a hypothetical SoC with 72 RISC-V cores, 64 banks of LLC, and 8 DRAM channels with an 8x8 mesh NoC. Constellation was able to elaborate functioning RTL for the NoC in this system.

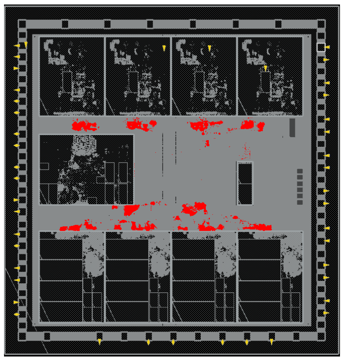

Fig. 9. A heterogeneous multi-core SoC designed and taped-out by 20 undergraduates in an advanced undergraduate course, featuring a 10-node ring network. Red indicates NoC router components.

D. *Test-chip Evaluation*

In a recent VLSI course at our institution, 20 undergraduate students were coached to develop and tape-out a multi-core heterogeneous SoC in an advanced process node. The Constellation generator was provided to them, along with the rest of the Chipyard SoC generator framework. The students developed a multi-core SoC for low-precision sparse machine learning and pixel processing, with 5 heterogeneous cores attached to a banked memory subsystem through a ring configuration of the Constellation NoC.

Constellation acted as the system backbone for this project, simplifying integration in a hierarchical design flow. The memory system closed timing at 500 MHz in this technology, for a 64b-wide 12-node ring interconnect. The successful tapeout of this configuration demonstrates how Constellation is useful for scaling out heterogeneous test-chips, or for simply getting physical design feedback for NoC configurations.

VI. Related Work

A. *NoC Generators*

OpenSoC and OpenSMART are Chisel-based NoC generators supporting several configurable topologies [8], [9]. OpenSoC supports mesh and butterfly topologies, as well as AXI-4 transport. While the BSV variant OpenSMART supports arbitrary topologies, we found that the open-source Chisel variant of OpenSMART only provides support for fixed 2D Mesh topologies. OpenSMART also does not implement any real interconnect protocol.

Unlike OpenSoC and OpenSMART, Constellation implements a multi-phase decoupled approach to NoC generation, enabling more detailed fine-grained customization of NoC implementation, and out-of-the-box support for a much wider set of design points.

CONNECT and Nocgen are other NoC generator frameworks that support varied topologies and configurations [10], [11]. However, these frameworks do not provide support for transporting a real interconnect protocol, and are thus not suitable for integration into an SoC.

Paulo et. al. proposes a framework for homogeneous NoCs implemented on a heterogeneous floorplan [24]. Unlike this work, Constellation is capable of generating synthesizable RTL for arbitrary heterogeneous NoC configurations.

Unlike all of these systems, Constellation was designed with SoC integration as a first-order goal, while still providing convenient interfaces for low-level NoC design exploration. Constellation's specification system also provides a far more expressive mechanism for concisely describing a wide set of topologies and routing policies. Additionally, we believe the fine-grained per-channel and per-router configuration space in Constellation expands the potential design space far beyond what existing frameworks support.

B. *SoC Design Frameworks*

Open-source SoC design frameworks provide a unified, accessible flow for generating and evaluating SoCs. ESP, OpenPiton, Chipyard, and Blackparrot are prominent research frameworks in this space, and all have implemented various features to enable rapid design space exploration of custom SoCs [25]–[28].

ESP, OpenPiton, and BlackParrot all implement configurable 2D mesh NoCs with dimension-ordered routing. While these NoCs are scalable to large meshes, they have fixed topology, and implement a fixed custom coherence protocol. Unlike Constellation, these systems cannot express a highly irregular NoC with fine-grained configurability. Constellation is also protocol agnostic, and supports exploration of protocol mapping strategies or development of new communication protocols.

Chipyard uses Rocketchip's library of memory system components to generate its SoC subsystem [29]. These projects generate the subsystem using Diplomacy, an abstract graph framework for representing memory systems [30]. The standard implementation of a memory system in Chipyard and Rocketchip is a crossbar-based interconnect, not a NoC. Constellation plugs into Chipyard and Rocketchip as a diplomatic widget, replacing any existing crossbar interconnect with a configurable NoC.

VII. Conclusion

Constellation is a new Chisel NoC generator for enabling design exploration and evaluation of heterogeneous NoCs. Unlike existing frameworks, Constellation was developed with heterogeneity and SoC integration as first-order concerns. Constellation provides an expressive, decoupled specification system, a modular multi-phase generator architecture, and an extensible evaluation framework to make it useful for both NoC researchers and SoC architects. Constellation is open-sourced [2] and upstreamed as the standard NoC implementation for the Chipyard SoC framework.

[2]github.com/ucb-bar/constellation

REFERENCES

[1] C. Jacobi and C. A. Z. P. Design, "Real-time ai for enterprise workloads: the ibm telum processor.," in *HCS*, pp. 1–22, 2021.

[2] J. Redgrave, A. Meixner, N. Goulding-Hotta, A. Vasilyev, and O. Shacham, "Pixel visual core: Google's fully programmableimage, vision, and ai processor for mobile devices," in *Proc. IEEE Hot Chips Symp.(HCS)*, pp. 1–18, 2018.

[3] D. Suggs, M. Subramony, and D. Bouvier, "The amd "zen 2" processor," *IEEE Micro*, vol. 40, no. 2, pp. 45–52, 2020.

[4] E. Rotem, Y. Mandelblat, V. Basin, E. Weissmann, A. Gihon, R. Chabukswar, R. Fenger, and M. Gupta, "Alder lake architecture," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, pp. 1–23, IEEE, 2021.

[5] A. Frumusanu, "Apple announces the apple silicon m1: Ditching x86 - what to expect, based on a14," Accessed: 2022-7-25.

[6] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "The aladdin approach to accelerator design and modeling," *IEEE Micro*, vol. 35, no. 3, pp. 58–70, 2015.

[7] M. Hill and V. J. Reddi, "Gables: A roofline model for mobile socs," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 317–330, IEEE, 2019.

[8] F. Fatollahi-Fard, D. Donofrio, G. Michelogiannakis, and J. Shalf, "Opensoc fabric: On-chip network generator: Using chisel to generate a parameterizable on-chip interconnect fabric," in *Proceedings of the 2014 International Workshop on Network on Chip Architectures*, pp. 45–50, 2014.

[9] H. Kwon and T. Krishna, "Opensmart: Single-cycle multi-hop noc generator in bsv and chisel," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 195–204, IEEE, 2017.

[10] M. K. Papamichael and J. C. Hoe, "The connect network-on-chip generator," *Computer*, vol. 48, no. 12, pp. 72–79, 2015.

[11] J. Chan and S. Parameswaran, "Nocgen: A template based reuse methodology for networks on chip architecture," in *17th International Conference on VLSI Design. Proceedings.*, pp. 717–720, IEEE, 2004.

[12] W. J. Dally *et al.*, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed systems*, vol. 3, no. 2, pp. 194–205, 1992.

[13] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," 1988.

[14] J. Duato, "A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 10, pp. 1055–1067, 1995.

[15] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. Elsevier, 2004.

[16] N. McKeown, "The islip scheduling algorithm for input-queued switches," *IEEE/ACM transactions on networking*, vol. 7, no. 2, pp. 188–201, 1999.

[17] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High-speed switch scheduling for local-area networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 11, no. 4, pp. 319–352, 1993.

[18] SiFive, *SiFive Tilelink Specification*, 2021. 1.8.1.

[19] ARM Limited, *AMBA® AXI and ACE Protocol Specification*, 2021. H.c.

[20] F. Socal, "Amba moves forward with major revisions to axi and chi specifications," Accessed: 2022-7-25.

[21] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. J. Dally, "Booksim 2.0 user's guide," *Standford University*, p. q1, 2010.

[22] J. Hestness, B. Grot, and S. W. Keckler, "Netrace: Dependency-driven trace-based network-on-chip simulation," in *Proceedings of the Third International Workshop on Network on Chip Architectures*, pp. 31–36, 2010.

[23] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, vol. 5, 2020.

[24] V. De Paulo and C. Ababei, "A framework for 2.5 d noc exploration using homogeneous networks over heterogeneous floorplans," in *2009 International Conference on Reconfigurable Computing and FPGAs*, pp. 267–272, IEEE, 2009.

[25] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni, "Agile soc development with open esp," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2020.

[26] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, *et al.*, "Openpiton: An open source manycore research framework," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 217–232, 2016.

[27] D. Petrisko, F. Gilani, M. Wyse, D. C. Jung, S. Davidson, P. Gao, C. Zhao, Z. Azad, S. Canakci, B. Veluri, *et al.*, "Blackparrot: An agile open-source risc-v multicore for accelerator socs," *IEEE Micro*, vol. 40, no. 4, pp. 93–102, 2020.

[28] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[29] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, 2016.

[30] H. Cook, W. Terpstra, and Y. Lee, "Diplomatic design patterns: A tilelink case study," in *1st Workshop on Computer Architecture Research with RISC-V*, 2017.