SoCProbe: Compositional Post-Silicon Validation of Heterogeneous NoC-Based SoCs

Gabriele Tombesi[1], Joseph Zuckerman[1], Paolo Mantovani[1,*], Davide Giri[1], Maico Cassel dos Santos[1], Tianyu Jia[2,†], David Brooks[2], Gu-Yeon Wei[2], and Luca P. Carloni[1]
Columbia University[1], Harvard University[2]

ABSTRACT

We present SoCProbe, a novel debug unit that enables compositional post-silicon validation of heterogeneous SoCs, particularly those featuring tiles connected by a NoC. SoCProbe, which is instantiated between each NoC router and its corresponding tile, is connected directly to the test environment with a lightweight 4-pin interface. It supports two test modes, which provide complementary capabilities for different types of faults, and has a companion flow for automatically collecting test vectors for any component through full-system RTL simulation. We demonstrate SoCProbe's effectiveness in post-silicon validation and low area overhead by integrating it in a complex SoC prototype fabricated in 12nm.

1 INTRODUCTION

Heterogeneous system-on-chip (SoC) architectures combine general-purpose CPU cores with GPUs, hardware accelerators, DSPs, sensors and I/O peripherals to provide a specialized solution for a target application domain [5, 16]. These SoC architectures increasingly rely on the scalability of a network-on-chip (NoC) [3] to connect a growing variety of intellectual property (IP) components.

However, heterogeneity raises design complexity [12], which inherently increases the likelihood of faults in a chip design. Increasing heterogeneity brings more lines of RTL code, more interactions among components of different natures, and more physical design runs. The required verification effort to ensure a bug-free chip also increases. In turn, this increases the likelihood of a fault, particularly when verification hours are limited due to small team sizes. Moreover, faults in certain portions of the design, such as host processors, primary I/O interfaces, or interconnect, can make it impossible to validate other components. Yet, even in the presence of faults, it is desirable to understand which portions of the chip are functional.

Industry-standard design-for-testing methods such as Boundary Scan, combined with complex Automatic Test Pattern Generator (ATPG) algorithms, provide fine-grained structural testability of manufactured chips by allowing access to each scan register. In this work, we address the complementary need for *coarse-grained functional validation* of particular SoC components, when there are faults present in other portions of the chip that would otherwise render this type of validation impossible. In doing so, we target a solution that is broadly capable of validating any kind of component, addressing the needs dictated by heterogeneity

We develop *SoCProbe*, a low-area debug unit for heterogeneous SoCs with tiled, NoC-based architectures. In fact, the SoCProbe approach relies on the regularity of a NoC architecture as well as on the decoupling between communication and computation that it provides. SoCProbe is instantiated between each NoC router and its corresponding tile and is connected directly to the test environment with a lightweight 4-pin interface that helps meet the tight I/O constraints imposed by large designs.

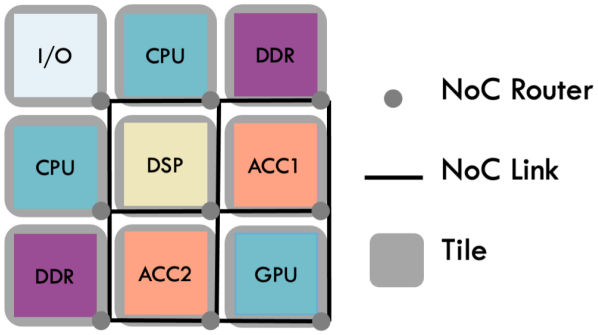

Figure 1: A 3x3 tiled SoC with heterogeneous processing elements connected by a network-on-chip.

SoCProbe features two complementary test modes. *Tile-test mode* provides direct access to the tile from the test environment and enables validation of its contents independently from the rest of the system. *NoC-test mode* instead enables injections to the router directly from the test environment, which can be used to replace a faulty tile's responses or evaluate the impact of a faulty tile on other SoC components.

To support SoCProbe, we made two additional contributions: (1) an automated flow for collecting test vectors for any test application, which can then be used by SoCProbe to stimulate the tile or its associated router, and (2) an FPGA-based test infrastructure that can stimulate each SoCProbe unit in a fabricated chip with these vectors. We demonstrated the effectiveness in post-silicon validation and low-area overhead of SoCProbe by integrating it in a complex SoC prototype that we designed and fabricated in a 12nm technology process.

2 MOTIVATION

Figure 1 shows an example of a heterogeneous, tiled SoC with a 2D-mesh NoC connecting 9 tiles: two CPU tiles for running software, including possibly an operating system, one DSP tile, two accelerator tiles, one GPU tile, two DDR tiles for accessing external DRAM,

∗ Paolo Mantovani is now with Google.
† Tianyu Jia is now with Peking University.

and one I/O tile. This SoC might rely on a *multi-plane NoC*, which instantiates multiple physical networks that operate independently. This approach is alternative to a NoC architecture based on virtual channels, as the multiple planes can be used to both prevent protocol deadlock and improve on-chip communication bandwidth [18]. Even this relatively simple system highlights the added complexity heterogeneity brings. Seven different tiles must be designed or integrated, verified, and carried through physical implementation. To further motivate our work, we outline several possible design faults and how they can impact post-silicon validation. We classify faults into three main categories.

Tile Faults are bugs that affect individual tiles. It is possible that this type of fault only affects the ability to validate the tile that contains the fault. However, if the fault affects a critical tile, it could render the other tiles or even the whole system unusable. For instance, accelerators that cannot execute software typically must be *invoked* from a host processor; if a fault affected the `CPU` tiles in the SoC in Figure 1, `ACC1` and `ACC2` could be unusable.

I/O Faults can prevent communication with the test environment. Interfaces like PCIe and DDR are highly complex and potentially come as third-party IPs from external vendors. A non-functional DDR controller would prevent processing elements from being able to access off-chip data; host processors and accelerators would be unable to fetch their programs and data, respectively, from memory. Test-chips typically utilize a debug interface to run tests, diagnose issues, and gather performance data; faults in this type of interface could lead to a completely untestable chip.

On-Chip Communication Faults can potentially be the most severe of the three, and NoC-based architectures are more prone to these than bus-based architectures due to their added complexity. A logical bug in the NoC can entirely disrupt on-chip communication, while physical design bugs may only affect communication between a subset of tiles. In these cases, it is desirable to have the ability to test tiles entirely *in isolation*. This was the primary motivation for developing SoCProbe, although its utility extends to the other two fault categories.

3 SOCPROBE

SoCProbe serves as a Debug Unit which sits between an individual *tile* and its NoC routers, enabling direct access to both the tile and the NoC in case of the faults described above. Here we define a tile to be the combination of all logic that is serviced by a set of corresponding routers across multiple physical planes[1]; a tile can potentially consist of one or more IPs or processing elements. SoCProbe can be configured to work in NoC-test mode or tile-test mode, which isolate the NoC router or the tile, respectively, from the surrounding logic to directly exchange flits with the test environment. For convenience, we shall henceforth refer to the target component that is connected to the test environment as the *element under test* (EUT).

Three key requirements inform our design choices:
(1) *Bypassable*: when disabled, SoCProbe should not impact the SoC's operation; communication between the NoC and the tile must pass through unaffected and with no added latency.
(2) *Latency-Insensitivity*: SoCProbe must cope with any sequence of communication from the EUT with arbitrary delays. By leveraging the paradigm of latency-insensitive design [6], SoCProbe is designed to be independent from both the NoC and tile implementations, thereby increasing its reusability across heterogeneous tiles.
(3) *Low I/O Requirements*: SoCProbe must not require more than a few pins to the external test environment, and this value should be constant with respect to the number of physical planes or channels in the NoC architecture.

[1]Each physical plane contributes one router per tile.

3.1 Interface

A first design decision is to leverage a lightweight 4-pin interface for SoCProbe's I/O in order to better meet the scalability requirements of large, tiled SoCs. The SPI pin serially shifts in requests to the EUT. The SPM pin toggles SoCProbe between *normal mode*, in which the standard communication is established between the NoC and the tile, and one of the supported test modes, in which SoCProbe is used to stimulate the EUT. This satisfies Requirement 1. The SPO pin is used to serially shift out requests from the EUT. Finally, since we do not require high performance in test mode, we utilize the SCLK pin to provide SoCProbe with its own slower clock, thus relaxing timing constraints.

3.2 Microarchitecture

Maintaining latency-insensitivity (Requirement 2) and reducing the communication bandwidth from the potentially hundreds of pins of a multi-plane NoC design to the 4-pin interface (Requirement 3) are the key properties of our design of SoCProbe. Figure 2 shows SoCProbe's microarchitecture. It consists of two layers of logic that serve distinct purposes: the *Serialization/Deserialization Layer* and the *Multi-plane Orchestration Layer*. The figure abstracts these two layers for a three-plane NoC architecture and highlights the logic involved in the management of a single NoC plane. The design can be extended to accommodate an arbitrary number of planes.

Serialization/Deserialization Layer. This layer converts between the serial I/O interface of SoCProbe and the bitwidth used at the tile-NoC interface. The basic unit of exchange over a NoC link is a *flit*. Multiple flits compose a *packet*, which contains a complete message between two components (e.g. a write request or read response). SoCProbe operates at the granularity of flits; each NoC flit from the test vector is shifted in serially through the SPI pin from the test environment and stored in *a serial-in-parallel-out (SIPO) register*.

Figure 2 illustrates a detailed view of a test flit injected to SoCProbe from the testing environment; the NoC flit is highlighted in green and matches the bitwidth of the NoC. The test flit also comprises a test-type bit (*tt*), which determines if SoCProbe is operating in NoC-test mode or tile-test mode. The flit also contains some metadata, shown in blue, required by SoCProbe to appropriately handle it: one bit identifies the flit message type (*ft*), and another set of bits identifies the index of the physical plane to which the flit corresponds. The flit type can be:

- A *request for injection*: a message to send the EUT as part of the input test vector recorded in simulation.
- A *response for extraction*: the *golden response* expected from the EUT as a result of the sequence of requests previously injected, to be compared with the actual response obtained.

This article has been accepted for publication in IEEE Design & Test. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2023.3310355

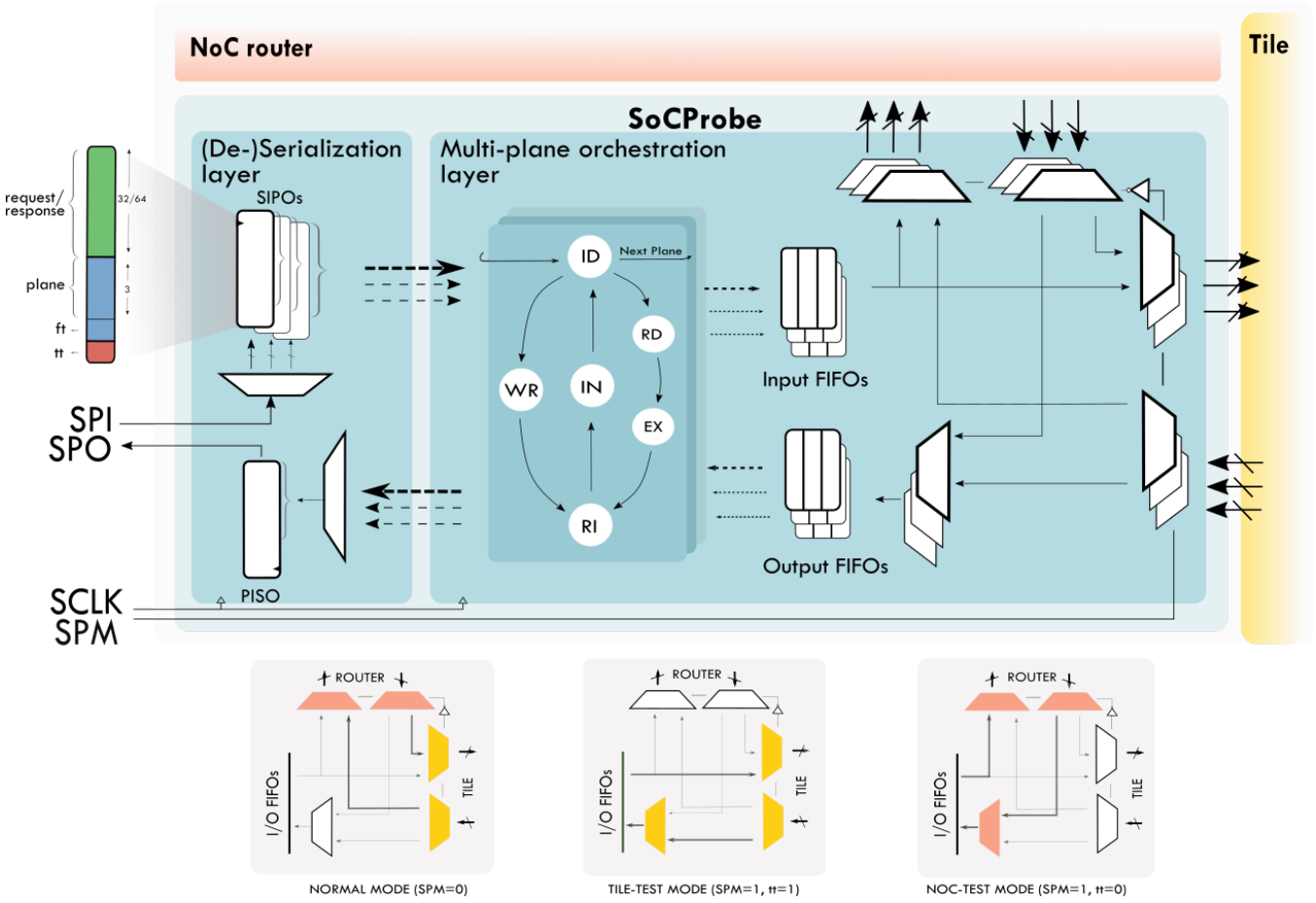

Figure 2: SoCProbe is inserted between the NoC and tile. Its design consists of a serialization layer, to convert between the width of the test interface and the width of the NoC, and a multi-plane orchestration layer, which manages control across the independent NoC planes. In normal mode, the tile and the routers are connected for normal operation; in test mode, it provides test access point to the EUT.

Demultiplexing logic steers the incoming flit to the SIPO register for the corresponding plane, where it stays until it is read by the orchestration layer. A dedicated register for each NoC plane allows for its independent control. In the opposite direction, multiplexing logic selects the response to extract from SoCProbe, including bits identifying the NoC plane and the test result, through a *parallel-in-serial-out (PISO) register*, whose contents are then sent out serially through the SPO pin to the test environment. Since the control logic of the orchestration layer delivers one response at a time, a single register is sufficient to shift out the response while keeping independent control over the NoC planes. The SIPO/PISO register operations and the multiplexing/demultiplexing logic are managed by control logic in the orchestration layer.

Multi-Plane Orchestration Layer. This layer manages the communication of SoCProbe with the EUT. There are three main components:

- Two asynchronous FIFOs for each plane of the NoC: one for request injections and one for response extractions. These convert the signals from the clock domain of SoCProbe to that of the EUT.
- Multiplexing and Demultiplexing logic for each plane of the NoC to establish connection between SoCProbe and the EUT in test mode and to restore the router-tile connection in normal mode.
- Control logic that implements SoCProbe's finite-state machine (FSM).

The lower section of Figure 2 shows the multiplexing logic of the orchestration layer for the three supported operating modes: When normal-mode is enabled (SPM=0), the multiplexers and demultiplexers steer the messages from the NoC directly to the tile and vice versa, without involving the rest of SoCProbe logic. On the other hand, in test-mode (SPM=1), the test-type bit of the current flit is used to program the mux/demux logic to either disconnect the router and connect the tile to SoCProbe (tile-test mode) or disconnect the tile and connect the router to SoCProbe (NoC-test mode).

This article has been accepted for publication in IEEE Design & Test. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2023.3310355

The control FSM implements round-robin arbitration across the NoC planes, servicing one plane at a time, to execute the correct routing of input requests and output responses. In the IDLE (**ID**) state, the type of instruction stored in the SIPO register of the current plane is checked. The register could have either a valid flit or an invalid flit. If the flit is invalid, the FSM moves to states for managing the next plane. If the flit is valid and the register contains a request for injection, then it is routed to the EUT through the corresponding asynchronous FIFO in the WRITE (**WR**) state. Instead, if it is a response for extraction, the controller attempts to read a response from the corresponding asynchronous FIFO in the READ (**RD**) state, compares it with the golden response value, and stores it in the PISO register, together with a bit encoding the comparison result and bits encoding the physical plane of the response result. The test response is then shifted out through the SPO pin in the EXTRACT (**EX**) state. In both cases, once the operation is executed, SoCProbe sends out a new flit request in the REQ_INSTR (**RI**) state (specifying the plane for which a new flit is required), shifts in the new flit in the INJECT (**IN**) state, and goes back to the IDLE state once the transfer is completed.

Implementing SoCProbe with a latency-insensitive approach automatically tolerates cases where the status of the FIFO queues prevents the current action from completing. When the input FIFOs are full, the EUT cannot currently receive additional messages on this plane and exerts *backpressure* on SoCProbe, which reacts by blocking subsequent write operations until there is space in the FIFO. Instead, if the output FIFOs are empty, the EUT has not yet produced a response on this plane, which blocks subsequent reads until there is a valid flit in the FIFO. If an operation is blocked in either case, the FSM moves to the next NoC plane to avoid idle cycles.

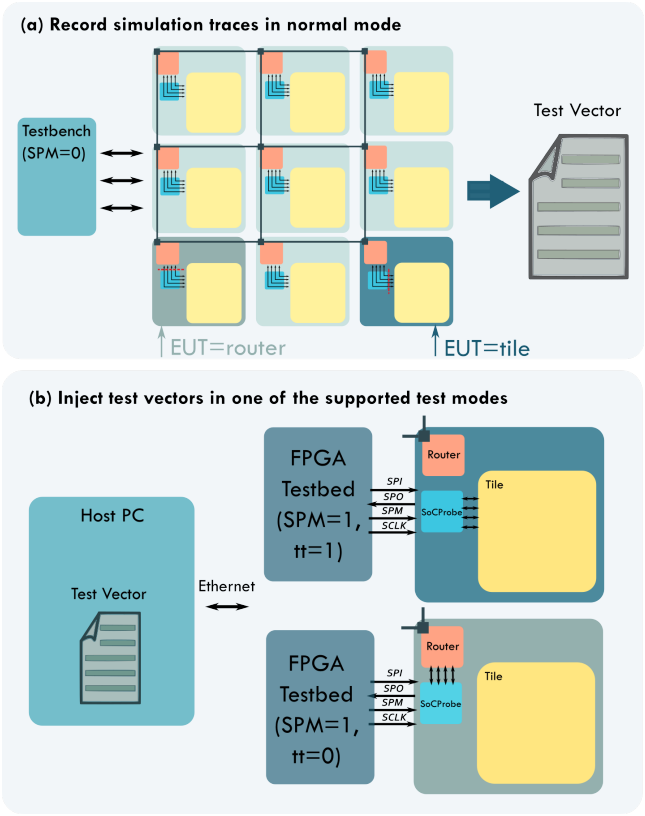

Figure 3: Our validation flow, consisting of (a) recording simulation traces in RTL simulation in normal mode and (b) stimulating the chip with the recorded vectors through an FPGA testbed in test mode.

3.3 Validation Flow and Infrastructure

Together with the design of SoCProbe, we develop an automated flow for collecting test vectors from RTL simulation, as well as an FPGA-based test infrastructure that can be used to stimulate a chip with these vectors. Figure 3 illustrates our validation flow.

SoCProbe expects a list of NoC flits, including both the input requests and the expected golden responses, with which to test the EUT. Because SoCProbe connects directly to the EUT, it is possible to merely collect flits at the NoC-tile boundary for any application, while ignoring what is happening in the rest of the system. We leverage full-system RTL simulation, with SoCProbe in normal mode (SPM=0), and record traces of the signals at the NoC-tile boundary. This approach simplifies the development of tests, which can be written as programs running on the host core. We also developed scripts for automated post-processing of the simulation traces into a time-ordered list of NoC flits, including information about the target NoC plane, the instruction type (injection or extraction) and the EUT type, which are needed by SoCProbe to appropriately handle each flit.

Additionally, we developed an FPGA test-bed that can be used to stimulate the fabricated chip by using extracted NoC flits as a test vector; industrial automated test equipment could also be used for this purpose. The FPGA communicates with a host PC via an Ethernet connection. The Ethernet connection is used to load the test vector into a set of FIFO queues in the FPGA design, which then sends the flits serially over SoCProbe's interface to the chip. The chip's serial response to the FPGA is reconstructed into flits, which are buffered into another set of FIFO queues, where they can be read back by the host through Ethernet for validation. We provide scripts to automate this whole procedure for a given test vector.

In order to make SoCProbe robust to communication delays from the host, it is designed to initiate transfers by requesting flits on a certain plane in the REQ_INSTR state illustrated above. If the test-bed does not yet have any flits for that plane, it sends an invalid flit to the chip that informs SoCProbe to move to the next plane.

The functionality of the SoCProbe debug unit itself can be extensively verified at different design stages before a tape-out. For any given application, a test vector can be obtained by simply running an RTL simulation of the whole SoC with all potential EUTs in normal mode (SPM=0). At this point, the proposed validation flow allows verifying SoCProbe's functionality both in RTL simulation and through FPGA emulation, by simply enabling test-mode for the target EUT through the testbench or the FPGA-testbed, as explained in Section 3.2.

4 SYSTEM-ON-CHIP INTEGRATION

Integrating SoCProbe in a complete SoC implementation is critical to evaluate its functionality, efficacy, and overhead. In this work, we conduct a case study on ESP [13], an open-source SoC design platform with a tiled, NoC-based SoC architecture. We note that ESP is one of several platforms [2, 14] with these characteristics and in which SoCProbe could be integrated. ESP features heterogeneous tiles and a 2D-mesh NoC with multiple physical planes, which make it a particularly suitable evaluation platform for SoCProbe.

Each of the 4 main ESP tile types – CPU, Accelerator, I/O, and Memory – is encapsulated in a modular *socket*, which connects to the NoC through latency-insensitive channels. These latency-insensitive channels greatly simplify the instantiation of SoCProbe inside each *tile wrapper*, which contains the tile itself along with its corresponding set of NoC routers. The connections between the routers and the tile are made through the SoCProbe instance, instead of directly.

The implementation of SoCProbe is tailored to match the configurable number of physical planes of the ESP NoC. The default configuration of ESP uses a six-plane NoC, where five planes are 64-bit wide and one is 32-bit wide. Accounting for input and output data and control signals, the NoC interface to the tile is in total 728 bits; SoCProbe can manage this complexity with only its 4 pins.

4.1 Chip Implementation

SoCProbe was first designed and validated leveraging RTL simulation and FPGA emulation. Then, we included it as an integral feature of an SoC we designed in a 12nm process. The fabricated chip has 16 tiles composed of 6 different types, including multiple open-source processors and accelerators, connected by a 4x4 2D mesh NoC [11]. Figure 4 shows the implementation of a RISC-V CPU tile in the chip design; SoCProbe is highlighted in red. In a 12nm process, SoCProbe occupies $3370\mu m^2$, i.e. 0.9% of the tile's area. We note that the taped-out version of SoCProbe only features the tile-test mode. Based on our experience testing this chip, we provided SoCProbe with the additional NoC-test mode feature and tested its functionality leveraging the FPGA emulation setup described in Section 3.3.

Adding the NoC-test mode results in negligible additional area overhead because it only requires additional (de-)multiplexers, as highlighted in orange in the NoC-Test Mode portion of Figure 2.

There are several options for how to connect the SoCProbe units, potentially instantiated in each tile of the SoC, to the chip I/O; these options present a tradeoff between design complexity and I/O utilization. A first option is to use one 4-pin interface for all SoCProbe units. This would require a centralized SoCProbe controller with some additional logic that selects the correct tile's SoCProbe unit to interact with. The target tile could be selected through a sequence of bits sent on the SPI signal. However, in a hierarchical physical design approach, tiles are implemented separately, and NoC connections between adjacent tiles are made during top-level chip integration. Adding a dedicated controller would break the regularity of the tile-based floorplan and result in long wires at the top-level that potentially span the entire chip.

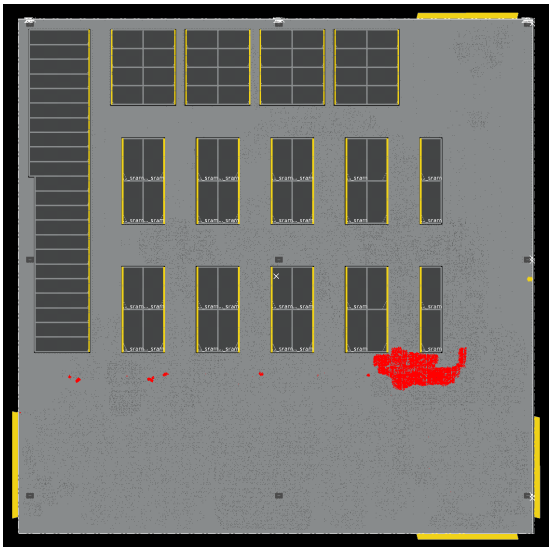

Figure 4: Implementation of a RISC-V CPU tile as part of the SoC that we designed and fabricated in a 12nm process. The SoCProbe unit is highlighted in red.

A less scalable option in terms of I/O utilization is to use a dedicated 4-pin I/O interface for each SoCProbe unit. For small designs, this might result in an acceptable amount of pins.

A third option, which sits in between the first two, consists in sharing common SPM and SCLK pins across all SoCProbe units, but using separate SPI and SPO pins for each unit.

The main trade-off of these strategies is between pin counts and wiring at the top level. Area is only marginally affected by the chosen strategy since it is mostly dominated by the amount of instantiated SocProbe instances, which is invariant across the different options described above. For our chip, we selected the third option. Two pins per tile was an acceptable I/O utilization that removed the need for any additional top-level logic and wires, thus mitigating risk. However, we acknowledge that the scalability of the first approach would be a better choice for larger chips. We discuss scalability further in Section 6.

5 CASE STUDIES

To demonstrate the versatile capabilities of SoCProbe, we report its use to carry out a variety of types of tests. Table 1 collects these test cases and describes their main properties. For the tests that use the tile-test mode, we use our fabricated chip; for those that feature the newer NoC-test mode, we use RTL simulation and FPGA emulation. We demonstrate that SoCProbe can be used for multiple types of validation and debugging. Table 1 reports some recorded metrics for each test, including the number of required NoC flits of each type, the test time in normal mode (*Norm. time*), the test time when leveraging SoCProbe in test mode (*Test time*), and the number of physical planes of the NoC involved in the test[2].

[2]In the tile-test mode, we keep the notation *NoC physical planes* to refer to the number of planes on which flits were recorded entering the tile under test, even if the NoC itself is not involved in the test. This value corresponds to the number of tile-NoC interfaces used to inject the flits from SoCProbe.

Test Case	Test Name	Test Mode	Tile Under Test	Total Flits	Reqs.	Resps.	Norm. Time (μs)	Test Time (μs)	Planes
1	CSR write	Tile	any	6	6	0	2.13×10^1	5.3	1
2	PC read	Tile	any	7	5	2	2.14×10^1	9.8×10^1	1
3	NoC intercept	Tile	IO	60	0	60	3.64×10^2	1.9×10^3	1
4	CPU in isolation	Tile	CPU	5842	2938	2904	4.45×10^2	7.97×10^4	3
5	ACC in isolation	Tile	ACC	682	536	146	1.21×10^3	9.88×10^3	3
6	MEM resp.	NoC	MEM	753	484	269	4.45×10^2	3.13×10^4	3
7	SYS orch.	NoC	CPU	22721	12151	10570	1.21×10^3	8.14×10^5	4

Table 1: Qualitative and quantitative summary of the seven test cases conducted with SoCProbe.

Measurements from the chip were obtained with the tiles and NoC running at a nominal frequency of 400MHz and SoCProbe running at 25MHz. We run experiments at the nominal chip frequency for simplicity, although various elements of the chip can run at maximum frequencies ranging from 800 MHz to 1.5 GHz. In normal mode, increasing the frequency of the chip will decrease the test time. In test mode, however, the overall time will be limited by the frequency and bandwidth of SoCProbe, so changing the main clock's frequency will not have an impact on the collected metrics.

For the NoC-mode tests, we report values from RTL simulation with the same frequencies and further validate functionality through FPGA emulation running at 50MHz.

SoCProbe is not meant to be a high-performance peripheral. Instead, it is designed to come with a scalable interface that requires little bandwidth and a low target frequency to relax timing constraints. Hence, it is expected that functional validation is slower in test mode, compared to running the same test in normal mode. NoC flit/plane counts are provided to show the complexity of each test, while latency is reported only to give a sense of the testing overhead in these case studies.

Writing a Configuration Register. Tuning Configuration Status Registers (CSRs), such as those for on-chip clock generators can be an important first step of chip bringup. In case of I/O faults in the primary debug interface or system-wide communication faults, SoCProbe can be used to modify these configuration registers by being set to operate in tile-test mode in the tile containing the target configuration registers. Test Case 1 demonstrates the ease and simplicity of this procedure, requiring the injection of just six flits on a single plane. We note that this test case displays a *Test time* lower than the *Norm. time*. This is mainly because running such a simple task from the CPU involves an overhead that is not needed when directly writing from the NoC interface to the tile.

Reading Performance Counters. Reading performance counters can be a helpful debugging tool during chip bringup. For example, if certain tiles are not responding to messages, it is helpful to know if NoC packets have reached them; accessing the NoC performance counters provides this information. In case of I/O faults in the primary debug interface or system-wide communication faults, SoCProbe can be used to access these performance counters by being set to operate in tile-test mode in the tile containing the target counters. Test Case 2 shows an example of this task.

Figure 5 shows a simplified view of the test vector collected for this task, which includes five injection flits (*ft* bit in dark blue) and two extraction flits (*ft* bit in light blue) exchanged on the same

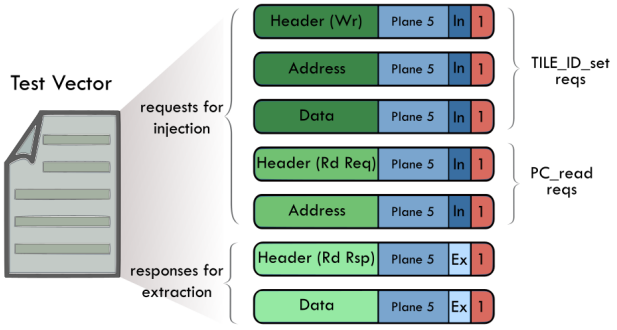

Figure 5: Simplified view of test vector for test case 2.

physical plane (Plane 5) in tile-test mode (*tt*=1). The requests can be divided into two packets. The first packet (dark green) is used to initialize the Tile ID CSR. In ESP, this happens in every tile at reset time, so this packet is common among all tests in Table 1. The three flits in this packet are the packet header, containing the message type and routing information for the packet to be appropriately steered on the NoC, the address of the target memory-mapped register for the write operation, and the data payload. The next packet (medium green) is a read request for the target performance counter. It also consists of a header and address, but it only has 2 flits because it does not carry any data. Similarly, the 2-flit read response packet (light green) has a header followed by the requested data.

Intercepting NoC Messages. An unintended use case of SoCProbe is being able to intercept NoC messages. If it is suspected that messages from certain tiles might be causing problems in the rest of the SoC, SoCProbe can be enabled in tile-test mode on these tiles to intercept these messages and examine their contents. We show this scenario in Test Case 3, where SoCProbe intercepts a 60-flit initialization sequence coming from the tile containing the chip's I/O peripherals.

Functional Validation of Tiles in Isolation. In case of tile-level faults in host processors, I/O faults on primary debug interfaces, or system-wide communication faults, SoCProbe can perform complete validation of individual components using the tile-test mode. We perform two tests: a Hello-World program running on the CPU (Test Case 4) and a small computational kernel executing on a deep-learning accelerator (Test Case 5). Given the increased

complexity of these tasks, the tests involve three NoC planes in both cases and a significantly higher test execution time.

Issuing Memory Responses. In case of I/O faults on the interface to external DRAM, SoCProbe can be used to inject expected values into the NoC as if they were responses from DRAM, thereby enabling the rest of the system to still execute with functional correctness. We provide a demonstration of this type of task in Test Case 6, where we use SoCProbe in NoC-test mode to stimulate the router of a tile that provides external memory access, running the same test as in Test Case 4.

Orchestrating Full-System Tests. In the case of faults in the host processor, SoCProbe can be used in its place to orchestrate a test that involves a subset of the rest of the system, e.g. an accelerator performing a computation by reading and writing data from main memory. In this case, SoCProbe can be used to mimic a host processor by injecting its expected requests directly into the NoC in the NoC-test mode. A concrete example of this use case is run in Test Case 7, where we activate NoC-test mode for a CPU tile while running the same deep-learning kernel as in Test Case 5. While Test Case 5 is limited by the bandwidth of SoCProbe, Test Case 7 allows the accelerator to execute with the full bandwidth of the NoC.

6 RELATED WORK

SoCProbe is primarily a *design-for-validation* solution for SoCs. *Design-for-testing* (DFT) and *design-for-debug* (DFD) are important complementary capabilities for chip design. Next, we discuss related work on these topics and existing IEEE standards that can support the adoption of the SoCProbe approach.

6.1 Design-for-Test/Debug for NoC

DFT focuses on adding mechanisms to test for and locate manufacturing faults in an integrated circuit (IC). There is abundant research on DFT in NoC-based SoCs, mostly focused on testing of the NoC itself or using the NoC as a *test access mechanism* [8]. One such example is utilizing the NoC to perform *concurrent online testing*, in which various components are tested throughout the SoC's deployment, while applications are running using other components [4]. Amory et al. developed a DFT approach for NoC-based architectures that consists of test wrappers for each component, and an interface to *automated test equipment* that converts from the data width of the test pins to the width of the NoC, which is used to deliver the tests [1].

DFD instead seeks to enable fine-grained control of the execution and observability of the internal state of on-chip components. Ciordas et al. developed a *NoC analyzer* that is capable of reconstructing NoC transactions on chip at runtime for debugging purposes; it has multiple modes for reconstructing the data at various levels of abstractions [7]. Tang and Xu developed a complete debug platform for NoC-based SoCs that consists of debug probes inserted between each core-under-debug and the network interface and uses the NoC to pass debug information throughout the system [17]. They also develop a novel two-pass debug strategy that can synchronize debugging across multiple cores distributed throughout the system.

Our DFV approach is orthogonal to this class of research, as we focus on the functional validation of individual components in the case of communication difficulties and validation of the rest of the system in case of faults in particular components. As described in Section 5, SoCProbe adds some debug capabilities during chip bring-up, but at a coarser granularity than the described DFD approaches. Most of the existing research in this field focuses on *homogeneous multicore* designs. Instead, we explicitly target *heterogeneous* SoCs and thus abstract the design to make it reusable for any component and any application invoking that component.

6.2 Relationship to Existing IEEE Standards

The interface of SoCProbe is inspired by that of IEEE standard 1149.1 (JTAG) for Test Access Port (TAP) and Boundary Scan Architecture [10]. The 4 (optionally 5) pin interface and an accompanying finite state machine used by the TAP is a widely adopted standard for testing SoCs. Due to the interface similarities, SoCProbe could be integrated with a JTAG architecture by utilizing a user-defined instruction.

More similar to SoCProbe is the IEEE standard 1500 for Embedded Core Test [9]. This standard addresses testing difficulties in the era of IP reuse by requiring that 1500-compliant components be delivered with a test wrapper or with their test requirements described in the Core Test Language to enable the design of such a wrapper. Similarly to SoCProbe, the Embedded Core Test standard enables testing of both internal and external (interconnect) logic, although at a finer granularity. SoCProbe could be integrated as a part of a 1500-standard wrapper, again controlled through user-defined instructions.

Both of the above standards can suffer from scalability problems, as controlling the test of multiple components from a single TAP requires daisy-chaining a scan chain throughout the system. The IEEE standard 1687 for Integrated JTAG (IJTAG) attempts to solve this problem by defining scalable scan networks interface for accessing and controlling embedded test instruments [15] and standardizing their operation with a plug-and-play interface. IJTAG supports a hierarchical structure, including a system-level TAP controller for managing the entire IC and multiple embedded TAPs for access to a smaller portion of the design. The IJTAG standard could be utilized to help SoCProbe scale to larger SoCs.

In contrast to the prevailing DFT techniques, the SoCProbe approach is an innovative and cost-effective strategy for modular post-silicon validation. Our proposed debug unit does not affect timing closure of the rest of the system and comes with a fixed area overhead for a given NoC architecture, irrespective of the target IP content. While modern EDA tools facilitate boundary scan integration, doing so requires enlarged routing channels along standard cells to accommodate scan chain signals. This can result in an area overhead of up to 5% to 20% and a frequency reduction of up to 5%.

These IEEE standards are complementary to SoCProbe, but integration with one or more of them may be desirable in order to combine design-for-test and coarse-grained design-for-validation capabilities into a single infrastructure.

7 CONCLUSION

We presented a solution for *compositional validation* of tiled SoC architectures connected by a network-on-chip. In the case of communication difficulties, our approach permits the direct access to

individual tiles of the SoC for validation, diagnostic, and debugging purposes. By leveraging the regularity of a NoC architecture, SoCProbe is designed to be reused across the different tiles of a heterogeneous SoC and supports a seamless flow for collecting test vectors for a variety of applications. We integrated SoCProbe in ESP and evaluated its efficacy and overhead in a 12nm chip prototype. *We plan to make the SoCProbe's implementation, test-flow scripts, and FPGA test-bed design available as open-source artifacts before NOCS 2023.*

ACKNOWLEDGMENTS

This research was developed, in part, with funding from the Defense Advanced Research Projects Agency (DARPA), and in part with funding from the Army Research Office under Grant Number W911NF-19-1-0476. This research is also supported by a National Science Foundation Graduate Research Fellowship. The views, opinions and/or other findings expressed are those of the authors and should not be interpreted as representing the official views or policies (either expressed or implied) of the Department of Defense, the Army Research Office, the National Science Foundation, or the U.S. Government. Distribution Statement "A": Approved for Public Release, Distribution Unlimited. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] Alexandre M. Amory et al. 2007. DfT for the Reuse of Networks-on-Chip as Test Access Mechanism. In *25th IEEE VLSI Test Symposium*. 435–440.
[2] Jonathan Balkind et al. 2016. OpenPiton: An Open Source Manycore Research Framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. 217–232.
[3] Luca Benini and Giovanni De Micheli. 2002. Networks on chips: a new SoC paradigm. *Computer* 35, 1 (2002), 70–78.
[4] Praveen Bhojwani and Rabi N. Mahapatra. 2007. An Infrastructure IP for Online Testing of Network-on-Chip Based SoCs. In *8th International Symposium on Quality Electronic Design*. 867–872.
[5] Shekhar Borkar and Andrew A. Chien. 2011. The Future of Microprocessors. *Commun. ACM* 54, 5 (may 2011), 67–77. https://doi.org/10.1145/1941487.1941507
[6] Luca P. Carloni et al. 2001. Theory of Latency-Insensitive Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20, 9 (Sept. 2001), 1059–1076.
[7] Calin Ciordas et al. 2006. Transaction Monitoring in Networks on Chip: The On-Chip Run-Time Perspective. In *2006 International Symposium on Industrial Embedded Systems*. 1–10. https://doi.org/10.1109/IES.2006.357464
[8] Érika Cota et al. 2003. The impact of NoC reuse on the testing of core-based systems. In *Proceedings. 21st VLSI Test Symposium*. 128–133.
[9] F. DaSilva et al. 2003. Overview of the IEEE P1500 standard. In *International Test Conference, 2003. Proceedings. ITC 2003.*, Vol. 1. 988–997. https://doi.org/10.1109/TEST.2003.1271086
[10] IEEE. 2001. IEEE Standard Test Access Port and Boundary Scan Architecture. *IEEE Std 1149.1-2001* (2001), 1–212.
[11] Tianyu Jia et al. 2022. A 12nm Agile-Designed SoC for Swarm-Based Perception with Heterogeneous IP Blocks, a Reconfigurable Memory Hierarchy, and an 800MHz Multi-Plane NoC. In *IEEE 48th European Solid State Circuits Conference*. 269–272.
[12] Brucek Khailany et al. 2018. INVITED: A Modular Digital VLSI Flow for High-Productivity SoC Design. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465897
[13] Paolo Mantovani et al. 2020. Agile SoC Development with Open ESP. In *IEEE/ACM International Conference On Computer Aided Design*. 1–9.
[14] Daniel Petrisko et al. 2020. BlackParrot: An Agile Open-Source RISC-V Multicore for Accelerator SoCs. *IEEE Micro* 40, 4 (2020), 93–102.
[15] J. Rearick et al. 2005. IJTAG (internal JTAG): a step toward a DFT standard. In *IEEE International Conference on Test, 2005.* 8 pp.–815. https://doi.org/10.1109/TEST.2005.1584044
[16] Yakun Sophia Shao et al. 2015. The Aladdin Approach to Accelerator Design and Modeling. *IEEE Micro* 35, 3 (2015), 58–70. https://doi.org/10.1109/MM.2015.50
[17] Shan Tang and Qiang Xu. 2007. A Multi-Core Debug Platform for NoC-Based Systems. In *2007 Design, Automation Test in Europe Conference Exhibition*. 1–6. https://doi.org/10.1109/DATE.2007.364402
[18] Young Jin Yoon et al. 2010. Virtual channels vs. multiple physical networks: A comparative analysis. In *Design Automation Conference*. 162–165.

Gabriele Tombesi (gtombesi@cs.columbia.edu) is a PhD student in computer science at Columbia University. His research interests include novel design flows and dataflow optimization for deep learning accelerators and design-for-testing techniques for heterogeneous SoCs. He received his M.S. degree in electrical engineering and B.S. degree in physical engineering from the Politecnico di Torino.

Joseph Zuckerman (jzuck@cs.columbia.edu) is a PhD candidate in computer science at Columbia University. His research interests include agile design techniques, novel architectures, and runtime optimization for heterogeneous SoCs. He received the S.B. degree in electrical engineering from Harvard University.

Paolo Mantovani (paolo@cs.columbia.edu) is a senior software engineer at Google. His research interests include architecture design and system level methodologies for the integration and programming of heterogeneous computing platforms. He received the Ph.D. degree in computer science from the Columbia University.

Davide Giri received the PhD degree in computer science from Columbia University in 2022, focusing on accelerator integration in heterogeneous SoC architectures. He received the Master's degree in electronic engineering from the Politecnico di Torino and the Master's degree in electrical and computer engineering from the University of Illinois at Chicago.

Maico Cassel dos Santos (mccassel@cs.columbia.edu) is a PhD student in computer science at Columbia University. He received the M.S. degree in Microelectronics (2015) and B.S.E.E. degree (2008) from Federal University of Rio Grande do Sul (UFRGS). Previously he designed security chips and trained hundreds of engineers in Cadence's digital tools in Brazil.

Tianyu Jia (tianyuj@pku.edu.cn.) is currently an Assistant Professor of school of integrated circuits with Peking University, Beijing, China. His research interests include accelerator design for domain-specific applications, heterogeneous SoC design and optimization. He received the PhD degree in computer engineering from Northwestern University, Evanston, IL, USA. He was an Assistant Research Professor with Carnegie Mellon University.

David Brooks (dbrooks@seas.harvard.edu.) is currently the Haley Family Professor of computer science with Harvard University, Cambridge, MA, USA. His research interests include architectural and software approaches to address power, thermal, and reliability issues for embedded and high-performance computing systems. He received the PhD degree in electrical engineering from Princeton University, Princeton, NJ, USA.

Gu-Yeon Wei (guyeon@seas.harvard.edu) is currently a Robert and Suzanne Case Professor of electrical engineering and computer science with Harvard University, Cambridge, MA, USA. His research interests include mixed-signal integrated circuits, computer architecture, and runtime software, looking for cross-layer opportunities to develop energy-efficient systems. He received the PhD degree in electrical engineering from Stanford University, Stanford, CA, USA.

Luca P. Carloni (luca@cs.columbia.edu) is a professor and the department chair of computer science at Columbia University. His research interests include methodologies and tools for system-on-chip platforms, heterogeneous computing, and design of distributed embedded systems. He received the Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley. He is an IEEE Fellow and senior member of the ACM.