SoCCom: Automated Synthesis of System-on-Chip Architectures

Atul Prasad Deb Nath, *Member, IEEE*, Kshitij Raj, *Member, IEEE*, Swarup Bhunia, *Senior Member, IEEE*, and Sandip Ray, *Senior Member, IEEE*

***Abstract*—We present CAD framework and EDA tool, SoCCom, for automated synthesis of optimized SoC architectures. We delineate a disciplined and streamlined methodology to enable automated IP integration and design optimization. SoCCom supports generation of a wide variety of optimized SoCs by: 1) automating the entire process of intellectual property (IP) standardization and integration; 2) allowing configurable assembly of complex, scalable systems with application-specific subsystems; and 3) enabling optimization and evaluation of generated designs based on area and power constraints. Applications of SoCCom include development of heterogeneous, domain-specific SoCs, rapid register-transfer level (RTL) prototyping of wide-varieties of SoC benchmarks, and many others.**

***Index Terms*—Benchmarking, intellectual property (IP) standardization, system-on-chip (SoC) security, SoC synthesis.**

I. Introduction

SYSTEM-ON-CHIP (SoC) architectures enable system-level functionality on a single substrate of integrated circuit (IC) through the integration and coordination of a set of pre-designed hardware intellectual property (IP) cores. Modern SoCs typically incorporate hundreds of IPs that communicate over a variety of shared-bus and network-on-chip (NoC) interconnects. Their design requirements vary significantly due to the diverse nature of applications, e.g., Internet of Things (IoTs), automotive systems, infrastructure components, and military equipment. On the other hand, current practice relies on manual efforts and human expertise for design insights, optimization, and verification. In addition to being error-prone, this methodology provides little flexibility in design space exploration and optimization for specific applications and target platforms. The difficulty in existing methods of SoC integration arises from several factors, including soaring design complexity, lack of configurability at IP and system-level assembly, and development of SoCs within the tight boundary of design constraints while meeting aggressive time-to-market requirements.

In this paper, we take the position that the limitations of existing approaches of SoC integration can be addressed significantly by adopting the very principle of SoC design, e.g., systematic coordination of pre-designed IP blocks that are standardized and optimized for fast and efficient integration. We demonstrate that by enabling IP standardization and representing the design collateral in a structured, disciplined way, it is possible not only to overcome the major complexities of current SoC development process but the steps of IP integration can be automated providing flexibility in design configuration, parametric optimization, and streamlining of register-transfer level (RTL) prototyping.

Several tools have been developed over the years for many-core and multicore system development [1]–[6]. These frameworks help the users perform micro-architectural design space exploration and develop processor and memory-based subsystems. However, the existing development kits fail to address the SoC integration issues in terms of flexibility in IP and interconnect fabric selection, parametric optimization, and generation of application-specific custom SoCs. Consequently, generating tiled, hierarchical many-core, and multicore systems with arbitrary topology and application-specific subsystems is still a challenging process, and it incorporates several *ad hoc* and manual steps. System-level simulators are often employed as an alternative to developing SoCs from the ground up [7]–[10]. The SoC simulators help the design evaluation process by eliminating the need for RTL implementation. However, the software models developed via existing simulators often fail to mimic the accurate footprint of actual SoC designs. SoC simulators can be complementary to the development platforms, but complete reliance on the software models can lead to inaccurate observations and flawed conclusions [11]. While existing SoC development platforms and simulators help study and evaluate various aspects of the design, these are constrained by several limiting factors and do not fully address the existing challenges of SoC integration process.

To address these limitations, we develop a methodology that streamlines the SoC integration process and enables designers to generate a wide variety of optimized SoC designs. The major contributions of our work are as following.

1) *CAD Framework and EDA Tool:* We develop a novel SoC integration framework and EDA tool that supports standardization of IPs and promotes interoperability across open-source and industry-standard interfaces and interconnect protocols. It enables automated assembly and connectivity of complex, configurable systems with flexibility in developing large-scale SoCs comprised of application-specific subsystems. The modular framework allows the designers to create single core SoCs

Manuscript received July 22, 2021; revised November 17, 2021; accepted December 18, 2021. *(Corresponding author: Atul Prasad Deb Nath.)*

The authors are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: atulprasad@ufl.edu; kshitijraj@ufl.edu; swarup@ece.ufl.edu; sandip@ece.ufl.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2022.3141326.

Digital Object Identifier 10.1109/TVLSI.2022.3141326

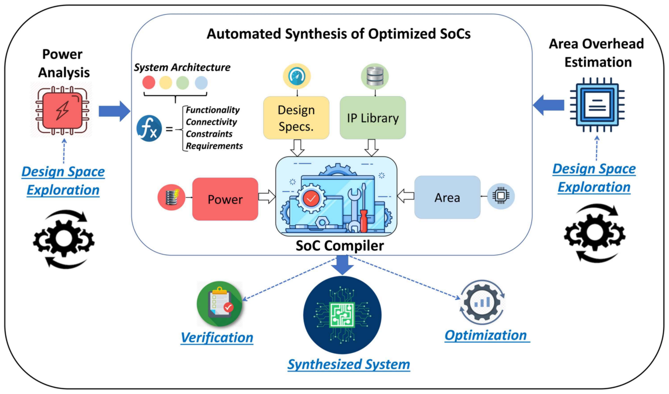


Fig. 1. High-level overview of SoCCom usage model: SoCCom is provided with design specifications and it generates synthesizable Verilog/SystemVerilog RTL of SoCs. The RTL-based SoC designs can be simulated, synthesized, and optimized for critical design parameters, i.e., area and power.

and subsystems from the ground up and gradually build tiled multicore and many-core systems. Our tool works in tandem with the existing commercial design flow and is compatible with off-the-shelf EDA tools.

2) *Optimized Design Composition:* Our work enables design space exploration and parametric optimization in terms of area and power constraints. The tool exhaustively generates SoC variants based on designer specifications and delivers area and power optimized SoC designs. The streamlined methodology of fast RTL prototyping facilitates rapid design evaluation and optimization for targeted applications such as the IoT and automotive systems.

3) *SoC Benchmark Generation:* An upshot of exploiting our framework for SoC synthesis is that the designer can generate a large number of SoC benchmarks with minimal effort. Our tool is capable of generating a wide class of flexible, highly configurable SoC benchmarks, including hierarchical shared-bus based SoCs and subsystems, and tiled many-core and multicore complex SoCs with NoC interconnect fabric. It also incorporates an easily extendable, open-source IP library that contains the major building blocks of present-day commercial SoCs.

The remainder of the article is organized as follows. Section II provides a comparative overview with existing literature. Section III illustrates the usage model from the SoC designer's standpoint. Section IV describes the modular building blocks of SoCCom tool from the developer's perspective. Examples of SoCCom generated illustrative SoCs are presented in Section V. Section VI shows how optimized SoC designs can be generated via SoCCom. The usage of SoCCom in standardized benchmark generation is described in Section VII. Section VIII illustrates the use cases of the tool in SoC security and validation. We discuss the application space of our work in Section IX. We conclude the article in Section X.

II. Related Work

The current approaches of SoC architecture research and development can be broadly classified into two categories. The first category consists of open-source SoC platforms and design kits that help to develop RTL-based designs. The second category relies on system level simulators to develop abstract SoC models via software modeling. In this section, we provide an overview of existing literature in each categories.

A. Frameworks for SoC Development and Architectural Design Space Exploration

In recent years, there has been significant work on various open-source multicore and many-core systems, processor-based subsystems, and design space exploration platforms [1]–[6]. Parulkar *et al.* [1] introduced OpenSPARC, a pioneering open-source hardware development initiative that focused on building open-source microprocessors. 32- and 64-bit SPARC processors are developed in this project with simulation tools and verification suites. These designs are optimized for field-programmable gate array (FPGA) synthesis. While OpenSPARC developed openly available multi-threaded multicore designs, the scope of the project is limited to micro-processor architectures with specific designs such as multilevel caches and crossbars. Fatollahi-Fard *et al.* [2] developed OpenSoC Fabric, a parameterizable NoC tool that enables the generation of connected networks containing processors and memory modules. The tool generates software models of C++ and Verilog hardware models by leveraging Chisel HDL. This work is constrained by generated Verilog models that are only suitable for synthesis. Such Verilog models are not human readable and not amenable to changes by SoC designer. Also, the framework only supports AXI-Lite bus standard for ARM and ARM compatible devices and the NoC generated by the tool supports a limited set of topology; thus, it limits the designer's flexibility in developing heterogeneous SoC designs with diverse architectures and interconnect fabric. Kinsey *et al.* [5] developed Heracles, a functional and parameterizable multicore system tool kit developed for RTL-based design space exploration. It allows flexibility to the designer in terms of processor core parameterization, memory organization, and network topology selection. The application of the Heracles toolkit, however, is confined to MIPs ISA-based subsystem exploration. The work of Kinsey *et al.* [5] is further extended to RISC-V- based ISA by Bandara *et al.* [4] to generate similar configurable processor-based subsystems. Asanovic *et al.* [6] presented Rocket chip generator that generates synthesizable RTL by compiling SoC designs written in Chisel HDL. Rocket chip includes a library of generators for cores, caches, and interconnects. However, it is quite difficult for SoC designers to analyze and make changes to the generated Verilog because of poor readability; thus, Rocket chip offers very limited design flexibility. It is possible to extend the Rocket chip generated SoCs with custom accelerators through instruction set extension or as co-processors. However, it suffers from the limitations of complex IP integration and interface standardization. The scope of the chip generator is also limited by its lack of configurability in interconnect topology. It uses generated ON-chip networks for computing tiles and allows extension of the design with peripheral through shared-bus crossbar. Wallentowitz *et al.* [3]

TABLE I
COMPARATIVE OVERVIEW OF SoCCom WITH EXISTING SoC AND CONFIGURABLE PROCESSOR DESIGN FRAMEWORKS

Framework	Core	Interconnect	Topology	HDL	3PIP Support/IP Library	Area Optim.	Power Optim.	Security Feature Support
OpenSPARC [1]	UltraSPARC	Bus	N/A	Verilog	×	×	×	×
OpenSoC [2]	ARM	NoC	Mesh/Butterfly	Chisel	×	×	×	×
RocketChip [6]	RISC-V	Bus	N/A	Chisel	×	×	×	×
Heracles [5]	RISC-V	NoC	Arbitrary	Verilog	×	×	×	×
BRISC-V [4]	RISC-V	NoC	Arbitrary	Verilog	×	×	×	×
OpTiMSoC [3]	OpenRISC-1000	Bus/NoC	Mesh	Verilog/SystemVerilog	✓	×	×	×
SoCCom	RISC-V	Bus/NoC	Arbitrary	Verilog/SystemVerilog	✓	✓	✓	✓

developed a library-based tool to generate tiled platforms, named, open tiled many-core system-on-chips (OpTiMSoC). The tool allows tile layouts with shared and distributed memory systems. The generated tiles are connected through ON-chip interconnect fabric. The ON-chip network supports only a set of specific topology including mesh and ring. The limitations of many-core, multithreaded designs have been addressed by Xiao *et al.* [12] by introducing a novel methodology that employs complex network theory of social communities. It models the dynamic execution of applications and partitions the applications to an optimal number of clusters for parallel execution. The high scalability of the approach makes it a promising modality for enabling automation in software designs. A novel approach of brain-inspired plasticity is introduced in [13] that exploits the fractal traits of high level programs (HLPs). The work presents plasticity-on-chip (PoC) by enabling plasticity into artificial brains through distributed reinforcement learning. The exploitation of similarity in the data communicated between processing elements and NoCs shows a significant performance improvement of 7×.

A major limitation in existing SoC design and evaluation platforms is the lack of flexibility in terms of IP and interconnect selection, system-level configuration and topology formation, and more importantly, design optimization. We address the limitations of current approaches by developing a comprehensive framework that offers adequate flexibility to the user in the aforementioned design aspects and enables automated synthesis of a wide variety of benchmarks including SoC models capable of scaling to complexity of tasks and realizing system functionalities efficiently. Table I provides a comparative overview of our work with existing design frameworks available in the literature. The comparison is performed based on the salient features of these frameworks such as core support, compatible fabric, i.e., bus and ON-chip network, open-source/3PIP compatibility, employed HDL, and finally the ability to generate area and power-optimized designs with potential to integrate security features. OpenSparc is one of the pioneers in open-source hardware development that supported shared-bus and UltraSPARC processor core written in Verilog. OpenSoC focused on ON-chip fabric-based many-core designs with ARM cores generated from Chisel. Comparatively newer suites of platform development such as RocketChip, Heracles, and BRISC-V deliver RISC-V-based designs with shared-bus and ON-chip network support. RocketChip delivers Verilog-based designs generated from Chisel whereas Heracles and BRISC-V adopts Verilog for RTL designs. OptimSoC generates OpenRISC-1000 processor-based SoCs with bus and NoC fabric and all designs are developed in Verilog. In comparison to existing literature, SoCCom provides enhanced flexibility to the SoC designers by delivering a diverse, extended IP library of Verilog and SystemVerilog-based RISC-V cores, memory modules, I/O peripherals, and a broad spectrum of hardware accelerators for application-specific SoC designs. As it does not employ Chisel-generated HDL, the generated RTL designs are highly customizable in nature. SoCCom further overcomes the limitations of contemporary works by enabling selection of area and power-optimized IPs as per design needs and augmentation of generated SoCs with a diverse set of security features with the ease of plug-n-play-based standardized security IP and wrapper support.

B. System-Level Simulation Platforms

System-level simulators have become a promising modality for architecture research and product development since they eliminate the need of developing RTL designs. The complexity of IP integration and SoC architecture development usually leads researchers to adopt simulators for rapid design evaluation and feasibility analysis of their work.

Binkert *et al.* [7] introduced gem5, an architectural modeling tool that offers the flexibility of simulating different CPU models, memory systems, and system modes. The salient features of gem5 include homogeneous and heterogeneous cores, interchangeable CPU models, multiple ISA support, and event-driven memory systems. It provides workload simulation and performance analysis with full system support for several architectures including ARM, RISC-V, and x86. Agarwal *et al.* [8] augmented gem5 with an NoC simulator, namely GARNET, that aids the designers to develop a detailed NoC model inside gem5's full system simulator. It is a cycle accurate simulator that allows the users of gem5 to leverage and customize the micro-architecture of routers and system-level topology. Hence, the users can conduct the performance evaluation of their designs by generating various traffic scenarios of ON-chip interconnect. Li *et al.* [9] developed McPAT, a simulator for modeling timing, power, and area of multicore and many-core processors. The simulator supports modeling of multiprocessors, NoCs, and integrated memory controllers at micro-architectural abstraction level. It also supports circuit-level modeling including critical-path timing, area, and leakage power modeling. McPAT is further extended as GPGPU-Sim by Bakhoda *et al.* [14] for system-level modeling of GPUs. Leng *et al.* [10] introduced a simulator named GPUWatch to develop GPU power models through cycle-level calculations that are compatible with the GPGPU-Sim developed by [14].

Existing simulators provide a varied range of modeling options to evaluate and validate architectures on different levels of abstractions. From the early days of SimpleScalar to modern state-of-the-art simulators such as gem5, the simulators have become more sophisticated and detailed as they provide cycle accurate performance estimates. However, the problem with ubiquitous application of the simulators is that these are often over-fit for validation, i.e., the parameters of the simulators are only optimized for a small set of benchmarks and configuration parameters. Such over-fitting leads to generalization of different architectures and causes several pitfalls.

The abstraction used by most of the simulators is usually at a very higher level as these act as first-order models. Despite the abstract modeling, the simulators are often labeled as detailed simulation platforms. The decisions of modeling some parts of architecture in detail and abstracting out the rest are made at the sole discretion of designers. Such *ad hoc* nature of the development process often limits the functional efficacy of the platform as these can neither deliver accurate, detailed results, or efficient abstraction of unnecessary details. On the other hand, the simulators are often treated as black boxes by the users and hence, the possibility of bugs being present in the platforms is overlooked. Consequently, the validation results obtained from the simulators provide a false sense of confidence on the designs. Lack of understanding about the underlying mechanism of validation techniques employed by the simulator can lead the users to wrongfully assume that change of parameter at some other design point will help to keep the accuracy similar. However, most simulators work by fitting design parameters to specific validation targets while disregarding the accuracy of the modeling for an individual architectural phenomena or interaction among system components. It is of utmost importance to choose the right platform for architectural evaluation based on the footprints of the designs rather than relying on simulators to perform quantitative observations. Generalized approaches of evaluation based on common practice can lead to erroneous and false validation results. We believe our framework is complementary to the existing simulators and it can be utilized alongside simulators to obtain accurate results through design space exploration and evaluation of diverse architectural schemes.

III. SoCCom Usage Model

An overview of SoCCom usage model is shown in Fig. 1. It depicts the high-level usage of the tool from an SoC designer's perspective while disregarding the intricacies of implementation. SoCCom enables design configurability at varying granularity of abstraction levels, including system-level assembly and connectivity to IP-level microarchitecture. The tool is built with modular components to facilitate configurable IP integration. The primary components of the workflow include user-defined design specifications, SoCCom IP library, and area and power constraints as optional optimization parameters. The automated design process starts with the procurement of design specifications in the form of standardized files for storing simple data structures and objects such as JSON files. These files define the SoC architecture in terms of SoC components and topology. The global (system-level) and local (IP-level) parameters of the SoC platform, including the area and power constraints, are also described in these files.

SoCCom analyzes the design specification files (JSON) to generate a graphical representation of the overall system topology. Each entity, e.g., an IP or a router is considered a node in the system graph, and the connection among the nodes is portrayed as the edges, e.g., the links of the NoC fabric. Note that the very notion of organizing these building blocks of SoC design in a well-defined, structured way, such as the nodal graph, makes it feasible to streamline the SoC integration process and automate the entire flow. The optimized design generation features of SoCCom allow the designers to develop SoCs with optimally chosen IP blocks from the standard library, based on the pre-specified design constraints. The tool exhaustively generates all possible designs, and reports area and power-optimized designs with a standardized metric.

The end products of SoCCom are automatically generated SoC models in the form of synthesizable Verilog and SystemVerilog RTL designs. SoCCom also generates corresponding configuration files based on the user description of respective modules, e.g., memory-mapped hardware accelerators, peripherals, and I/Os, accordingly. Functional validation of the generated RTL-based SoC designs is performed with commercial off-the-shelf and open-source simulation tools. Furthermore, the RTL designs are synthesized via existing tools to develop FPGA and ASIC-oriented SoC prototypes.

IV. SoCCom Platform

In this section, we delineate SoCCom from a developer's perspective and provide a detailed discussion on the major building blocks of the SoCCom platform including the algorithms used for architecture composition. The workflow of the tool is shown in Fig. 2. At first, the compatibility and availability of the IPs are checked based on design specifications. For instance, all IP must be compatible to Wishbone interface for further enhancement to AXI or on-chip network protocol. Then, SoCCom proceeds with the selected interconnect type, i.e., bus or NoC, to standardize the interface of IPs. In shared-bus-based designs, the IPs are connected to target cross-bars (e.g., AXI4 and Wishbone) to form the single-bus SoC or a subsystem that connects to the primary bus on a multibus SoC. In NoC-based designs, the IPs are connected to network adapters to form tiles. Compute tiles are generated with processors whereas peripheral tiles are generated with peripherals and hardware accelerators, and system tiles are generated with multiple IPs connected through a shared bus which act as a separate subsystem. All different tiles are connected to routers in accordance with the design topology to develop the NoC-based design. The generated designs are evaluated for the area and power overhead by invoking the off-the-shelf design tool and an optimized design report is generated that ranks the SoCs in a standardized scoring format.

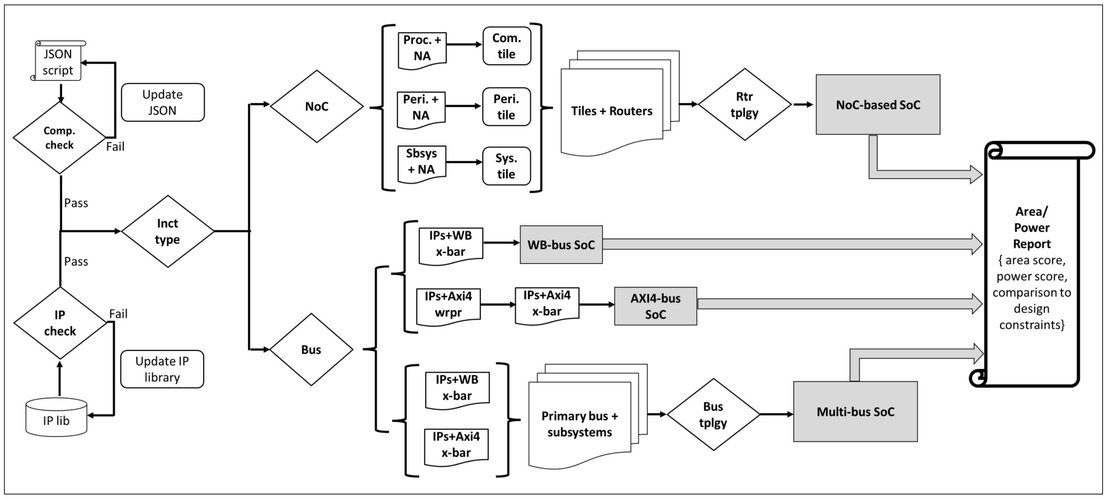


Fig. 2. SoCCom work flow. Based on the design specifications and architecture definitions provided via JSON script, the tool generates SoCs with bus and NoC interconnects including single and hierarchical bus-based SoCs and NoC-based systems with diverse subsystems. The generated designs are analyzed and optimized for area and power overhead with the help of off-the-shelf design tools.

A. HDL Analyzer

IP standardization is a critical step in SoC integration. This step facilitates the design process by ensuring that all IPs and interconnect components are *integration-ready*. The *HDL analyzer* module of SoCCom is developed to standardize and formalize the IPs for streamlined connectivity. In particular, it enables a systematic approach to extract and aggregate IP integration criteria as design metadata. Extraction of metadata from each component of the design is a key step in enabling IP interoperability across various interconnect protocols.

At first, the *HDL analyzer* procures the SoC components, essentially the IPs and interconnect fabric components, from SoCCom IP library, based on the architecture definition provided in the data store/exchange (JSON) file. Then, it employs various pre-defined functions on the retrieved IP blocks to obtain corresponding metadata. The extracted metadata include port definitions, i.e., functional inputs and outputs, port types and width, clocks and resets, interrupt signals, local and global parameters (specific to SystemVerilog designs), and headers such as *define*, *import*, and *include* of each IP, network adapter, router, and shared-bus crossbar. SoCCom supports the integration of memory modules to SoC designs as dedicated instruction and data memory. The configuration of the memory modules can be specified as parameters, i.e., memory size, single port, and dual port through the JSON file used for architecture definition. Cache hierarchy via user specifications is not supported in the current implementation of the tool. The metadata is utilized to generate a compatible interface for each IP with respect to the components of interconnect fabric (e.g., network adapters and routers) and shared-buses (e.g., crossbars). The algorithm employed by *HDL analyzer* is illustrated in Algorithm 1. It adopts regular expressions to analyze the IP source files and extracts the required information (port definitions, IO, port types and widths, clock, resets, etc.) for integration. It uses Python built-in libraries and regular expression patterns to extract the targeted metadata by performing pattern matching to match and store the extracted data from the source files. These matched patterns are then sorted and stored in a centralized local database based on their type identification like type of IO, clock, reset, etc. Several pre-defined classes and methods are used to post-process the matched patterns.

Fig. 3 shows a simplistic version of JSON configuration file with only two routers, a RISC-V core, and an AES slave IP. *R_0* is the configuration of Router 0, with its neighbor being Router 1, and Endpoint 0, i.e., the RISC-V Core. The field *IP_type* shows the class of IP followed by its mode of operation, i.e., slave or master. The neighboring IPs are listed in the corresponding tuple along with the fabric type and additional configurations specific to particular IPs. For instance, the configuration “RV32I” for the processor core represents that it supports the basic integer RISC-V ISA whereas the “128b” configuration of the AES core means that it supports 128 bits of encryption mode. It also shows the base address and offset of memory-mapped slaves as shown in Fig. 3 for the AES core.

B. Hierarchical SoC Generator

The *SoC generator* module helps generate SoC designs with shared-bus architectures. These designs can be shared-bus-based SoCs or modular subsystems for complex, multicore, and many-core systems. It currently supports the generation of single and hierarchical multibus SoCs with Wishbone B3 and AMBA AXI4 bus standards. The SoC generation process starts with processing the metadata obtained by *HDL analyzer*. The generator procures the IP metadata from the *HDL analyzer* and utilizes the information to generate compatible interfaces and connectivity for all the masters and slaves connected to the crossbar. It employs design parameters such as address bus width, data bus width, memory size allocation, memory-mapped I/O, and peripheral address size and ranges, etc. to generate the top-level RTL of the SoC that instantiates the IPs with corresponding wire and register declarations.

Multibus SoCs: The *SoC generator* module supports the generation of SoCs with hierarchical multibus interconnects. It enables augmentation of Wishbone-based subsystems with

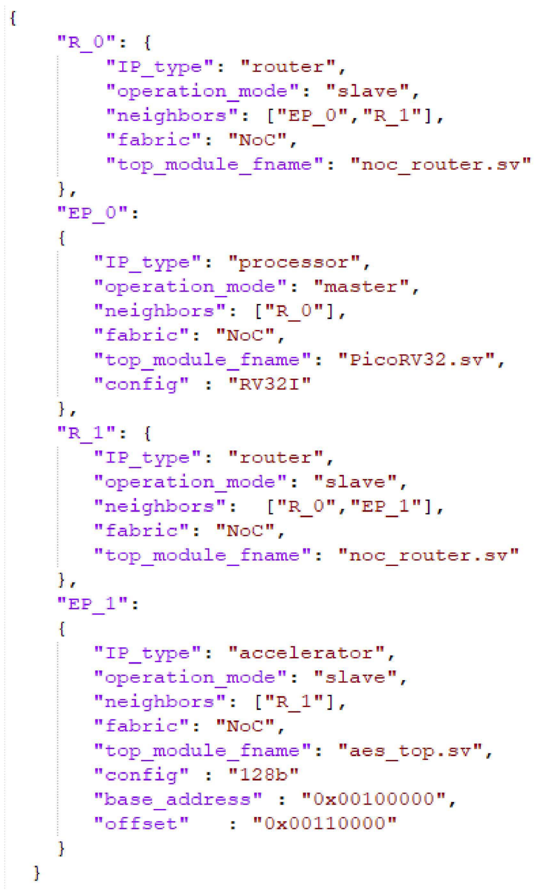
Algorithm 1 HDL Analyzer

```
function EXTRACT_HEADERS(self)
    Extract Packages, Headers, and Include parameters
    for x in Packages do
        $match \leftarrow param.findall(contents)$
        header.append(tuple(match))
    end for
end function
function EXTRACT_PARAM(self)
    $match \leftarrow param.findall(contents)$
    parameters.append(tuple(match))
end function
function EXTRACT_IO(self)
      $io \leftarrow io\_v.findall(contents)$
    for (IP_top_modules in topIPs) do
        $io[net\_index] \leftarrow matches\_io[i].strip()$
    end for
    if (interface.type == 'input'):
    $input[net\_index] \leftarrow matches\_input[i]$
    elseif (interface.type == 'inout'):
    $input[net\_index] \leftarrow matches\_inout[i]$
    else (interface.type == 'output'):
    $input[net\_index] \leftarrow matches\_output[i]$
   endif
end function
```

AXI4 compatible bus bridge. The standard bus-bridge establishes inter-bus, i.e., AXI4 to Wishbone, communication. Hence, the Wishbone-based subsystems can be connected as additional slaves to the main AXI4 bus of the SoC. The multi-bus structure introduces flexibility in integrating open-source Wishbone bus compatible IPs and subsystems to the industry standard AXI4 bus and thus, enhances interoperability in hierarchical designs. Moreover, the hierarchical bus architecture enables the designers to create modular subsystems and reuse them based on application requirements.

C. Many-Core System Generator

The *system generator* module of the framework is designed to generate many-core SoCs with NoC fabric. The scalable architecture of many-core SoCs facilitates the integration of hundreds of IPs to build large-scale, complex SoC designs. The *system generator* enables the generation of NoC-based SoCs with processors, peripherals, and system tiles. The processor and peripheral tiles are augmented versions of the cores and hardware accelerators with bus wrappers, respectively, which are integrated with network adapters for NoC compatibility. The system tiles incorporate multiple IPs connected via an internal shared-bus for local communication. The *system generator* supports the development of SoCs with application-specific subsystems in which several functional IPs (e.g., processors, memory IPs, accelerators, etc.) are clustered together and connected to the ON-chip fabric through dedicated routers. The module analyzes the design specification file provided by the user and utilizes *HDL analyzer* and *SoC generator* modules to generate many-core NoC-based SoCs. The workflow of the *system generator* incorporates several key steps, including network adapter integration, tile generation, router topology formation, configuration file creation, and top-level system RTL generation.



```
{
    "R_0": {
        "IP_type": "router",
        "operation_mode": "slave",
        "neighbors": ["EP_0","R_1"],
        "fabric": "NoC",
        "top_module_fname": "noc_router.sv"
    },
    "EP_0":
    {
      "IP_type": "processor",
      "operation_mode": "master",
      "neighbors": ["R_0"],
      "fabric": "NoC",
      "top_module_fname": "PicoRV32.sv",
      "config" : "RV32I"
    },
    "R_1": {
      "IP_type": "router",
      "operation_mode": "slave",
      "neighbors":  ["R_0","EP_1"],
      "fabric": "NoC",
      "top_module_fname": "noc_router.sv"
    },
    "EP_1":
    {
      "IP_type": "accelerator",
      "operation_mode": "slave",
      "neighbors": ["R_1"],
      "fabric": "NoC",
      "top_module_fname": "aes_top.sv",
      "config" : "128b"
      "base_address" : "0x00100000",
      "offset"    : "0x00110000"
    }
  }
```

Fig. 3. Sample snippet of JSON configuration file with two routers, a RISC-V Core, and an AES crypto core in a NoC-based SoC architecture.

Once the design specifications are provided to the *system generator*, it invokes the *HDL analyzer* to analyze the IPs, and builds processor and peripheral tiles, and the system tiles with the help of *SoC generator* module of the SoCCom. The generated tiles incorporate the network adapters for router connectivity. The *system generator* then assembles the routers (and connected tiles) in the pre-specified topology and creates the top-level RTL module. Since the framework is designed to represent the router connectivity as a nodal graph with endpoints, it is amenable to any arbitrary topology formation. The top-level RTL contains all required parameters, headers, packages, and configurations for system funtionality. Additional system configuration files are generated for the user to configure the peripheral mapping. The algorithm employed by *system generator* is shown in Algorithm 2.

V. SoCCom Generated Representative SoC Models

In this section, we describe various kinds of SoC models generated by SoCCom. By SoC models, we refer to the various SoC designs generated by SoCCom. The models are architected with many substantial SoC components and

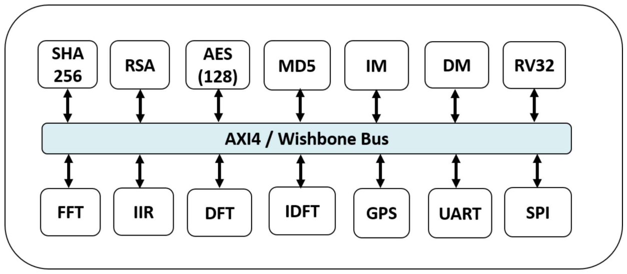


Fig. 4. Illustrative SoC with single shared-bus architecture. The SoC can incorporate an AXI4 or Wishbone bus.

reflects relevant features and complexities of shared-bus and NoC-based industrial SoC designs, including standard hardware accelerators, hierarchical bus designs, tree topology in ON-chip network, a mix of high-speed and low-speed IPs, and large-scale many-core system with application-specific subsystems. SOCCOM generated representative SoC models are available in [15].

A. SoCs With Shared-Bus Interconnect

The users can generate single shared-bus-based SoCs via SOCCOM. The shared-bus SoC can incorporate any IP from the SOCCOM library including processor cores, memory IPs, crypto and DSP accelerators, and IPs for external connectivity. The baseline requirement for IP integration is Wishbone bus compatibility. SOCCOM extends Wishbone compatible IPs with standardized bus wrappers for connectivity to AXI4-based SoCs Fig. 4 shows an example SoC with Wishbone /AXI4 interconnect with a diverse set of functional IPs. Multibus SoCs can be generated in SOCCOM by integrating Wishbone-based subsystems to AXI4 interconnect. Fig. 5 shows a multibus SoC model with hierarchical subsystems connected to the primary bus. Three Wishbone-based subsystems of crypto, DSP, and connectivity IPs are connected to the SoC bus via standardized bridges.

B. SoCs With Network-on-Chip Interconnect

Here, we describe two specific NoC-based SoC models generated via SOCCOM.

1) ClusterSoC: The `ClusterSoC` design consists of the processor and peripheral tiles generated by SOCCOM. The processor tiles incorporate a register-based L1 cache as a representation of distributed memory architecture. The memory tiles work as the system memory modules. `ClusterSoC` includes a 32-bit, size-optimized RISC-V CPU, single and dual port SRAMs integrated as memory modules. Three crypto modules, i.e., AES-128, DES3, and SHA-256 are included for generic cryptographic operations. It also features three high-performance DSP blocks, including FFT, IDFT, and FIR. An SPI controller and a UART module are integrated for external communication. The IPs are segregated into two clusters, inspired by the topology realistic industrial SoC designs for mobile and IoT devices. The North cluster includes the high-speed IPs (e.g., CPU and DSP accelerators) and the South cluster incorporates crypto engines and external communication IPs.

Algorithm 2 Manycore System Generator

```
1:  function EXTRACTIO(IP Filepath, IP Filename)
2:      Extract I/O, Parameters, and Headers from IPs
3:  end function
4:  function IP_INTERFACER(IP Filepath, IP Filename)
5:      IP_NA ← Join IP top module with Network Adapter
6:      IP_Router ← Join IP_NA with Router
7:      return IP_Router
8:  end function
9:  top_IPs = [ ]
10: for (IPs in IP_l) do
11:     topIPs.add(IP_Interfacer(IP_Filepath,IP_Filename)
12: end for
13: for (IP_top_modules in topIPs) do
14:     Create NoC in configured topology
15: end for
```

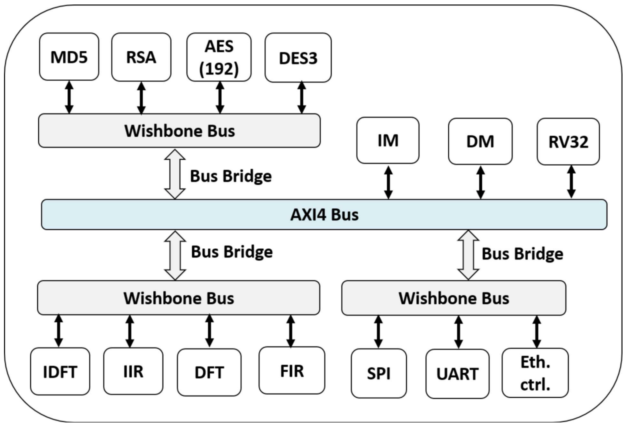


Fig. 5. Multibus SoC: this SoC model has an AXI4 interconnect as the primary bus and three Wishbone buses as subsystem interconnect.

2) AutoSoC: The `AutoSoC` model is inspired by commercial NoC-based automotive SoCs with separate application-specific subsystems. As with realistic implementations, it has a much larger number of IPs. The IPs are organized into a number of subsystems. The SoC incorporates four 32-bit RISC-V cores with variations in instruction set support. These cores are size-optimized implementations of RISC-V processors. The crypto subsystem is augmented with RSA and MD5, DSP subsystem with DFT and IIR, the memory subsystem with DMA controller, and the connectivity subsystem with an Ethernet controller and a toy GPS IP.

Both SoC models incorporate a modular and configurable open-source network-on-chip, named LiSNoC, as the interconnect fabric [3]. It supports wormhole-based flow control. The basic routing algorithm is dimension-ordered, deadlock-free XY-routing. The number of input ports, output ports, and allocation of virtual channels (VCs) to input–output ports are parameterizable. The flit transfer occurs through a handshaking protocol implemented with the ready and valid signals of each VC. The associated network adapters feature DMA engines and message passing functionalities. The IPs are wishbone-bus

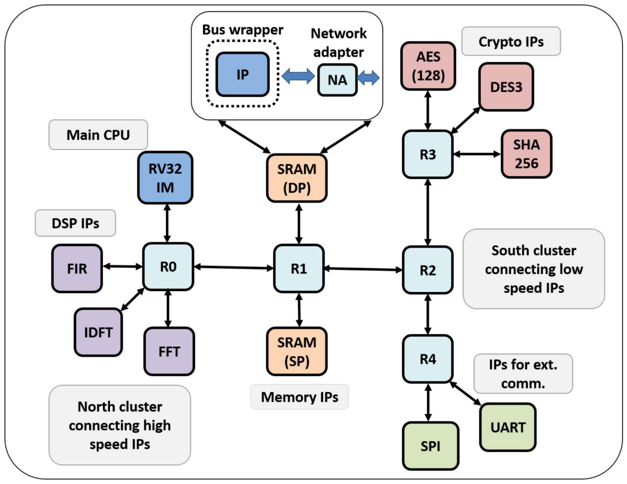


Fig. 6. ClusterSoC: representative IoT SoC model.

compatible. The network adapter facilitates their compatibility with NoC protocol.

C. Design Verification

All IPs in SoCCom library are obtained from various open-source repositories. These designs are written in Verilog and SystemVerilog RTL. The functional testing of individual IP is performed at the unit-level and bus interface. The RTL simulation of the SoC models along with unit and bus level testing of individual IPs are performed in Xilinx Vivado simulator and Verilator. In particular, the bus-level simulation of standalone IPs is performed in Verilator by augmenting the RTL testbenches with C++ wrappers. The routers and network adapter IP are tested with random traffic simulated via RTL testbench. The system-level simulation of SoC models is performed with basic RISC-V firmware and RTL testbenches. The tests include RISC-V ISA compliance test, register access test, and system memory read/write operations. In addition, the control and status register (CSR) read/write tests on the peripherals of the SoC models are performed via RISC-V firmware. The simulation time taken for functional verification and testing varied in the order of microseconds (RTL test-bench) to several minutes (bare-metal test firmware). The users can develop elaborate test cases for the generated SoCs by writing bare-metal RISC-V firmware and RTL testbench and perform use case-specific simulations. Note that the simulation time for elaborate test cases will vary in accordance to targeted test coverage and constraints of RTL simulation or/and FPGA emulation. In addition to functional verification, the use cases will enable the designers to perform SoC level timing and power analysis for targeted use cases. The maximum clock frequency of the SoC can be set through the JSON scripts. However, the critical path delay varies from design to design and SoCCom currently does not support user-specified critical path delays. The SoC models described in this section are synthesized for FPGA mapping via Xilinx Vivado Design Suites. The resource utilization report for the hierarchical multibus SoC, ClusterSoC, and AutoSoC on Xilinx Zynq UltraScale + MPSoCs FPGA is reported in Table II.

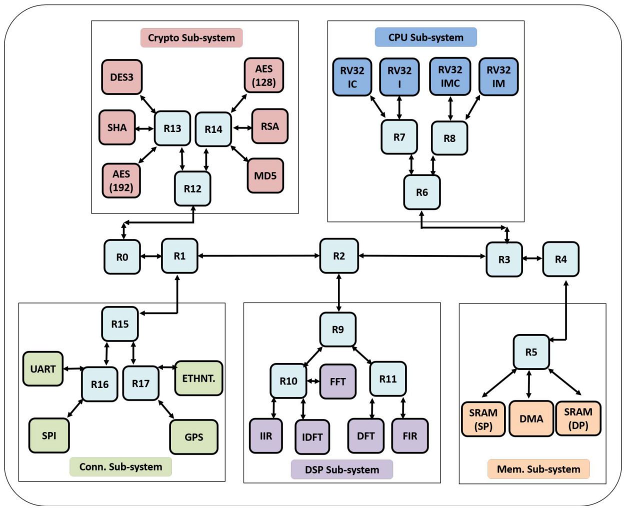


Fig. 7. AutoSoC: representative automotive SoC model with application-specific subsystems.

TABLE II
Area and Power Overhead Estimation of Various SoC Models

SoC Models	Power		Area		
	Static (W)	Dynamic (W)	LUTs	Registers	MUXes
Multi-bus SoC	0.781 W	13.864 W	30,478	27,236	774
ClusterSoC	0.845 W	25.521 W	66,066	64,314	1585
AutoSoC	0.952 W	30.529 W	81,730	79,434	1723

The area and power overheads of multibus SoC are significantly lower compared to that of ClusterSoC and AutoSoC due to the number and scale of IPs integrated into these designs. ClusterSoC and AutoSoC are NoC-based architectures and have a multitude of IPs, whereas multibus SoCs are bus-based designs, with a comparatively conservative number of IPs. In particular, the significant difference in area and power come from the number of additional subsystems in ClusterSoC and AutoSoC. Between Cluster and AutoSoC, the former has a relatively smaller design with cluster-based topology inspired by realistic industrial SoC designs for mobile and IoT devices. On the other hand, the AutoSoC model is inspired by commercial NoC-based automotive SoCs with separate application-specific subsystems. Due to the larger scale of IPs and an overall increase in the number of subsystems, the area and power overhead of AutoSoC is relatively higher. The overhead results of SoCCom SoC models presented here are not optimized for area and power constraints. We discuss optimized design generation feature of SoCCom in Section VI.

We generate a wide variety of SoCs via SoCCom. The SoCs vary from smaller single bus-based designs to complex NoC-based designs with compute tiles, peripheral tiles, and system tiles which in turn contain several IPs with shared buses. We presented experimental results on the area and power overhead as well as optimization scores of representative SoC models in terms of the overhead. However, we found that an apples-to-apples overhead comparison of SoCCom generated SoCs to designs available in the literature is not feasible due to their stark differences in architectural definitions and design components. Though there are CPU-based subsystem [4]–[6] and mesh topology-based scalable

SoCs [3] available in the literature, we could not find comparable designs of multibus SoC, ClusterSoC, and AutoSoC for overhead comparison.

VI. Optimized Design Composition

Multiobjective design optimization is crucial for the efficient deployment of SoC designs in various domains. Design overhead analysis via current frameworks requires manual development of individual RTL model or exhaustive analysis of each model via simulator. This problem can be partly alleviated by enabling the tradeoff analysis between multiple design objectives in terms of critical parameters (e.g., area and power) through a streamlined design flow. Such an approach will significantly eliminate the manual steps involved in the SoC development and evaluation process, and deliver improved efficiency in multiobjective design optimization. We address the limitation of multiobjective design optimization in SoC platforms by enabling SoCCom as an EDA tool for automated, optimized design composition.

Optimization With SoCCom: SoCCom supports automatic generation of designs optimized against multiple objectives. In particular, it generates a comprehensive set of SoC models based on user-provided design specifications and reports the most area and power-optimized designs along with other variants within the boundary of design constraints. SoCCom algorithm exhaustively generates all possible combinations of designer-specified SoCs with varying IP and system-level parameters, IP variants, interconnects, and streamlines the area and power estimation of individual designs with the existing commercial design tool.[1] Consequently, it reports SoC variants within the boundary conditions set by the designer and delivers a wide variety of SoC designs across the range of pre-specified constraints. It uses a standardized area and power metric to help the user quantify implementation overhead in terms of target platform resources. SoCCom uses a metric ranging over 1 – 10 to denote the area and power score from low to high. The higher scores of area and power basically represent the low area and power values of the design compared to other variants with the high area and power overhead.

SoCCom reports SoC variants that are configured with IP specific, i.e., local and system level, i.e., global parameters to meet the area and power preferences of the designer. An example of local parameterization is configuring the RISC-V processor to support-specific instruction set architectures (ISAs). SoCCom facilitates the implementation of different RISC-V ISA including integer (RV32I), compressed (RV32IC), multiply and divide (RV32IM), and extended ISA (RV32E). The modular framework allows the user to configure the RISC-V cores to support optional ISAs in tandem with the mandatory integer ISA implementation. Note that this custom ISA extension is supported by the RISC-V cores available in the IP library. The user specifies the desired ISA extension through the JSON script as a design parameter to configure it in the core. Similarly, based on user-provided metric, SoCCom presents SoC variants optimized in terms of global parameters such as enabling/disabling direct memory access of IPs, NoC VCs, and message passing via network adapters, optimized sizes of distributed local and shared global memory, etc.

[1]Obviously, the set of SoC variants generated would be exponential in the number of configurable parameters. However, in practice, we found that in the context of optimization of power and area on the SoCs we experimented, the set of design constraints keeps the number of variants manageable. Furthermore, as our experiments show, each synthesis run is efficient. Consequently, we can "get away" with an exhaustive enumeration of variants as reported here. However, in future work, we plan to integrate more tunable parameters which can result in the explosion of the number of variants and consequently, heuristics to effectively explore the variant space.

The SoCCom tool works in tandem with Xilinx Vivado design suite to streamline the SoC generation and subsequent design synthesis process. In particular, we extend the framework to invoke the Vivado design suite upon generation of SoC variants and initiate the synthesis process. Once Vivado delivers the synthesis report, the tool parses the results and ranks the designs accordingly (as shown in Table III). Note that pre-synthesized results of all IPs are stored in the IP library to reduce the computational overhead and perform sanity check on the initial overhead constraints provided by the user. For instance, if the user selected IPs and desired area and power overhead numbers are less than the minimum estimated values of the SoC then the tool informs the user to increase area and power overhead values. The synthesis report of Vivado design suite shows static and dynamic power, and FPGA resource utilization in terms of LUTs, registers, and MUXs. The synthesis report generated by the Vivado design suite is highly comparable to actual physical mapping as it provides the closest estimation of power consumption and resource utilization on target FPGA.

We demonstrate the generation of optimized SoC compositions with multiobjective design goals in SoCCom via ClusterSoC, as a proof-of-concept implementation. SoCCom design space consists of a large number of processor cores, memory modules, routers, network interfaces, shared bus crossbars, I/O peripherals, and hardware accelerators. All these types of IPs come with varying specifications. Among all IPs, we chose 5 IPs of ClusterSoC and exhaustively generated SoC variants by employing all variations of these selected IPs. We kept the number of IPs decent (five in this case) to reduce computational complexity of exhaustively synthesizing every possible combination of SoC designs with variants of each IP component. The IP variants include five variants of RISC-V cores including basic integer (I), extended (E), and multiply and divide (M) ISA support; three AES cores of 128-, 192-, and 256-bits; six fully streaming DFT and IDFT DSP cores of 32-, 64-, and 128-bits; and three different sizes ON-chip RAMs of 8, 16, and 32 Mb, that results in a design space of 270 SoCs. Generating these 270 SoC variants takes about 3 s in SoCCom. So, it takes 0.012 s for SoCCom to generate a valid design with this given set of IP variants. This number scales linearly when SoCCom exhaustively searches the design space. However, the synthesis process via Vivado design suite is expensive in terms of latency overhead as it takes about 300 per design. We ran SoCCom and the Xilinx Vivado Design Suite on a system with a Core i7 processor, 16 GB of RAM, and an NVIDIA RTX 2070 graphics processor. While synthesis of large-scale SoCs is an inherently time-consuming process, SoCCom works with the state-of-the-art

TABLE III
OPTIMIZED SoC COMPOSITION RESULTS: CASE STUDY ON CLUSTERSoC

Optimized SoC Composition Results: Case Study on ClusterSoC									
SoC Variants	Selected IPs with Variants	Variation Type	Power		Area			Score	
			Static (W)	Dynamic (W)	LUTs	Registers	MUXes	Area	Power
Variant #1 (Power Optimized)	RISC-V IDFT DFT SRAM AES	RV32IMV1 32-bit 32-bit 8 Mb 128 bits	0.815 W	23.536 W	58,432	63,234	1482	8	10
Variant #2 (Area Optimized)	RISC-V IDFT DFT SRAM AES	RV32IV1 32-bit 32-bit 8 Mb 128 bits	0.802 W	23.918 W	57,248	63,022	1404	10	9
Variant #3	RISC-V IDFT DFT SRAM AES	RV32IV1 32-bit 64-bit 8 Mb 192 bits	0.834 W	24.517 W	60,816	65,982	1532	8	9
Variant #4	RISC-V IDFT DFT SRAM AES	RV32E 64-bit 64-bit 16 Mb 192 bits	0.869 W	24.984 W	62,280	67,038	1522	7	8
Variant #5	RISC-V IDFT DFT SRAM AES	RV32IMV2 32-bit 128-bit 8 Mb 256 bits	0.861 W	25.211 W	64,336	69,984	1682	6	7
Variant #6	RISC-V IDFT DFT SRAM AES	RV32IV1 64-bit 128-bit 16 Mb 192 bits	0.893 W	25.653 W	65,582	68,242	1522	6	6
Variant #7	RISC-V IDFT DFT SRAM AES	RV32IMV1 32-bit 64-bit 16 Mb 128 bits	0.928 W	25.934 W	66,286	69,866	1582	5	6
Variant #8	RISC-V IDFT DFT SRAM AES	RV32IMV2 64-bit 128-bit 16 Mb 192 bits	0.937 W	26.359 W	67,428	69,998	1522	3	5
Variant #9	RISC-V IDFT DFT SRAM AES	RV32IV2 128-bit 32-bit 8 Mb 128 bits	0.952 W	26.214 W	67,584	65,484	1592	2	3
Variant #10	RISC-V IDFT DFT SRAM AES	RV32E 64-bit 32-bit 16 Mb 256 bits	0.977 W	26.618 W	67,986	67,286	1652	1	1

FPGA synthesis tool, i.e., Vivado Design Suite for efficient synthesis of generated designs. Optimization of RTL design synthesis time in off-the-shelf, commercial tools is beyond the scope of the SoCCom framework.

Table III illustrates ten different variants of ClusterSoC with varying IPs, and associated area and power scores that cover the entire spectrum of design constraints. We reported the area and power overhead of the designs based on the resource utilization at the target platform, i.e., a Xilinx Zynq UltraScale + MPSoCs FPGA in the given experimental set-up. We chose Xilinx ZynqUltraScale + MPSoCs FPGA as the target platform since it offers the maximum resources. SoCCom reports the resource utilization of the FPGA model in terms of MUXes, registers, and ALUs. The power analysis on potential FPGA mapping is reported through the ON-chip power broken down into dynamic and static power. As shown in Table III, variants 1 and 2 are the most power and area optimized SoCs, respectively, among the generated designs while variants 8, 9, and 10 are scored low since they consume comparatively higher area and power. In this work, we present a streamlined approach to generate area and power-optimized SoCs via parametric variations. Our tool currently reports the optimized designs in a standardized scoring format. However, we plan to address the area, power, and security aspects of generated design through mathematical optimization in the future as an extension of the current framework. The over-arching goal of the extension is to pose the power, area, and security features of generated SoCs through a well-formulated mathematical problem and achieve design optimization via an efficient solution.

VII. Standardized Benchmark Generation

An upshot of automated SoC synthesis process is the ease and flexibility in benchmark generation. Here we describe the application of SoCCom as an SoC benchmarking platform.

A. Scalable and Flexible SoC Benchmarks

Lack of open-source, standardized benchmarks is a major obstacle for SoC architecture research. Such scarcity makes it challenging to evaluate novel architectural schemes against a diverse set of SoC benchmarks. We fill up this void by employing SoCCom as an automated benchmark generation suite. We utilize the tool to generate a wide class of highly configurable, standardized SoC benchmarks. SoCCom generated benchmarks include hierarchical shared-bus-based SoCs and subsystems, and tiled many-core SoCs with NoCs and shared-buses. The user can configure SoCCom to generate SoC benchmarks with various classes of IPs and interconnects. For instance, the user can specify the IP types and interconnect fabric to generate all possible SoC designs within the given area and power constraints. Such benchmarking would allow the user to quickly generate a diverse set of SoC benchmarks. Moreover, the user can generate different sets of SoC benchmarks with arbitrary network topologies while changing the IPs, the types of tiles, i.e., processor tile, peripheral tile, and system tiles and their connections to the routers. Thus, the user can efficiently generate a large number of variations of system topology and study design performance by evaluating different positions of IPs and subsystems with respect to overall system topology.

B. Domain-Specific SoC Designs

Though past decades have seen the dominance of general-purpose computer architecture, it has become increasingly challenging to achieve domain-specific design and performance goals from generalized hardware. Given the prevalence of SoC designs in numerous applications in diverse sectors, it is crucial to ensure that they have specialized architectures to meet the design requirements of target devices while delivering optimum performance. The unique features of SoCCom enable the user to develop heterogeneous many-core SoC benchmarks comprised of general and special purpose processor cores, a variety of hardware accelerators, memory modules and controllers, and I/O devices to gain significant performance improvement within a specific domain. The systematic automation and streamlined design flow offered by SoCCom reduces the challenges, cost, and time of designing special purpose systems. An illustrative example of such use case would be designing an automotive SoC versus designing an SoC for IoT applications. Typically, for an automotive SoC, the design constraints stipulate higher execution performance for aggressive enforcement of real-time requirements whereas an IoT SoC would have requirements of optimum in-field operation within a tight boundary of area and power. Based on the application requirements of the domain, the designer can optimally select the different variant processor core and hardware accelerators dedicated for specific tasks. Similarly, depending on the design constraints and boundary conditions of area and power set by the designer, SoCCom facilitates optimum selection of IPs, interconnect fabrics, and local and global system design parameters. Thus, SoCCom can be a promising platform for generating heterogeneous SoC designs targeted toward domain-specific applications.

VIII. SoCCom Use Cases in SoC Security and Validation

SoCCom has been employed by our us as well as other research groups for a variety of SoC security and validation research. Following are a few illustrative applications.

A. Systematic Implementation of Security Policies

Case Study on E-IIPS: At present, there is a significant inadequacy of systematic methodologies and tool flow for SoC security feature integration that can be widely adopted by designers. SoCCom enables systematic integration of security features to generated SoCs and helps ensure system trustworthiness through *secure-by-construction* security architectures. In particular, it facilitates the addition of a standardized security architecture, namely, extended infrastructure IP for security (E-IIPS), for flexible implementation of IP and system-level security policies in SoC designs [16], [17]. The security policies implemented by E-IIPS architecture ensure provable system resilience against untrusted IP issues, diverse run-time security violations, and all major inter-IP communication attacks [18], [19].

SoCCom aids the automatic integration of the primary building blocks of E-IIPS architecture, i.e., a centralized security policy engine (SPE) block and distributed security wrapper integrated IPs. The SPE acts as a *central security brain* of the SoC and provides a programmable interface for configuring security specifications while the security wrappers communicate IP-specific security critical events with SPE to collaboratively determine the operating status of the SoC and assert security controls accordingly. SoCCom's IP library includes the SPE block and security wrapper augmented IPs. The user can select these from the library to realize the E-IIPS architecture and generate SoCs augmented with standardized security architectures for policy implementation. The current version of SoCCom incorporates two different implementations of the SPE: 1) a firmware upgradeable micro-controlled implementation and 2) a reconfigurable SPE designed on embedded FPGA. The SPEs work in tandem with IPs with security wrappers to enforce the security requirements. The user can generate E-IIPS integrated SoC design and implement custom security policies by updating microcontroller firmware or patching the FPGA bit stream. The user can flexibly choose specific security architectures based on domain-specific applications or target systems. For instance, the user can select FPGA-based SPE for improved performance and easy in-field patch whereas micro-controller-based SPE can be selected for area constrained designs such as IoTs and wearables.

B. SoC Security Validation Research

SoCCom generated SoCs are adopted as optimal vehicles for research endeavors aiming security validation. Researchers of the Trustworthy and Intelligent Embedded System (TIES) Group at the University of Texas at Dallas employed SoCCom generated SoCs for detecting security violations under asynchronous resets [20]. In particular, they developed a CFG extraction methodology that works in tandem with concolic testing to systematically explore the security violations stemming from multiple asynchronous reset domains in these SoC models. Three different kinds of security bugs, resulting information leakage, privilege mode violation, and data integrity breach, are inserted in SoCCom generated SoCs to evaluate the performance of their framework. Similarly, the SoC models generated by SoCCom are currently being used by researchers at the Embedded Systems Laboratory (ESL) at the University of Florida to develop scalable formal methods for validating SoC security vulnerabilities using security assertions [21].

C. Specification Mining for SoC Design Validation

The execution traces of SoCCom generated SoCs are analyzed for reconstructing system-level behavior via specification mining. The current methods of specification mining are reliant on manual, human insight of the architects. The researchers of the Secure, Efficient, and Evolvable Computing Systems (SEES) Laboratory at the University of South Florida are using SoCCom generated SoCs to conduct in-depth research and develop new approaches of system-level behavior reconstruction amid the presence of synthetic traffic [22]. Since SoCCom supports the generation of many-core large-scale SoCs, the researchers are able to exercise complex, system-level protocols of industrial SoCs and further explore the possibilities of re-constructing such on-chip interactions via trace patterns.

IX. SoCCom Stakeholders and Application Spaces

SoCCom is proven to be useful in several research endeavors that encompass a wide range of applications. It has critical impact on current practices of SoC architecture study and research topics, especially in an academic environment. The application space of SoCCom is described here.

A. Pedagogical Application as Design Education Kit

The lack of RTL-based models and benchmarks makes it difficult for engineering students to learn about SoC designs in an academic environment. This limitation can be addressed by employing SoCCom as an SoC design education kit and help students learn about different architectural aspects of the development process. The students can use pre-generated SoC benchmarks as illustrative models or generate their own SoCs by configuring the design parameters to get a suitable hardware composition. By using SoCCom as a design kit, the students can acquire good knowledge about RISC-V processors, memory modules, crypto modules, DSP blocks, and other peripherals for off-chip communication such as SPI, UART, and Ethernet controllers. As an intellectual learning outcome, the students will get trained on different techniques of SoC design and learn to evaluate the implementation results in terms of design constraints. As for the practical learning outcome, the students will learn about industry standard SoC protocols and functionalities and get exposure to commercial, off-the-shelf SoC design and simulation tools.

As a part of SoCCom's distribution plan as an educational design kit, it has been incorporated into the academic curriculum of graduate students enrolled in the Advanced VLSI design (EEE 6323, Spring 2020), a graduate-level course, at the University of Florida. The students employed the platform, i.e., the SoCCom tool and associated IP library, to study various aspects of SoC architecture and security, gain hands-on experience with RTL design simulation and emulation, and generate custom, optimized SoCs for a variety of class projects. The list of projects included, but were not limited to, the following: 1) extension of SoC designs with test wrappers to demonstrate improved testing features and coverage; 2) augmentation of functional IPs with standardized bus and NoC interfaces for shared-bus and on-chip network compatibility; 3) integration of standardized security wrappers to IPs, crossbars, and routers for critical event monitoring; 4) demonstration of a series of confidentiality, integrity, and availability attacks on SoC designs; and 5) development of subsequent attack mitigation strategies via security policies, etc. The participating students provided very positive feedback about SoCCom.

B. Research Applications

1) Rapid Exploration of SoC Design Space: SoCCom facilitate register-transfer level (RTL) design space exploration of SoC architectures. It enables exploration of SoC micro-architecture at varying levels of granularity, i.e., IP-level configurability and system-level assembly. The modular design components of SoCCom allow fast design exploration of processor cores, memory controllers, hardware accelerators, shared-bus, and NoC components, including bus parameters, router micro-architecture, and system-level topologies. The generated, synthesizable Verilog and SystemVerilog designs can be analyzed via RTL simulation or FPGA emulation. Moreover, the SoCCom generated designs are vendor independent and highly compatible with existing design flows. Hence, SoC architects and researchers across academia and industry can equally exploit the design exploration features of SoCCom.

2) Applications of SoCCom *in Hardware Security:* Here we illustrate the prospective use cases of SoCCom in different sectors of hardware security.

a) SoC Trojan benchmarking: A plethora of studies are available in the literature on IP-level hardware Trojans [23], [24]. However, a major drawback of existing research is the lack of SoC-level Trojan detection and mitigation techniques. SoCCom generated SoCs can be conveniently used for studying the detection and mitigation strategies of system-level Trojans. Since the payload of system-level Trojans propagate via on-chip communication, the security researchers can employ generated SoCs to overcome the limitations of static IP-level Trojan verification techniques, and detect and thwart the payload of system-level Trojans. In addition, the amenable design flow of SoCCom can be modified to work in tandem with existing Trojan insertion tools and develop a database of Trojan inserted SoC benchmarks for further studies [25].

b) SoC design obfuscation benchmarking: Design obfuscation is a proven technique to limit the attacker's access to OEM's designs [26]. While IP obfuscation techniques help the researchers study design security at the unit level, it is crucial to explore the impact and efficacy of obfuscation methods at system level operation. For instance, a secure boot process of an SoC typically incorporates unlocking the functionality of obfuscated IPs at power-up. Consequently, host SoCs are critical to fully exercise and validate the test cases pertaining to obfuscated IP designs. SoCCom generated SoC benchmarks can be utilized as host SoCs to fully simulate the steps of a secure boot flow, validate the locking and unlocking procedure of locked designs, and analyze the security performance tradeoffs of obfuscated IP blocks.

c) Information flow analysis: With the proliferation of 3PIPs in the SoC designs, information flow analysis has become more critical than ever [27], [28]. However, information flow analysis cannot be performed effectively via stand-alone IP designs as it requires a thorough evaluation of system-level communication. For instance, it is not feasible to study the vulnerabilities stemming from SoC designs due to insecure information flow via NoC components, such as routers and network components, through static IP analysis. This limitation is addressed by utilizing SoCCom-generated benchmarks as evaluation platforms. In particular, availability of these SoC benchmarks with on-chip network will facilitate the researchers to explore novel information leakage scenarios via ON-chip network components.

d) Access control analysis: The amalgamation of trusted and untrusted IPs on the shared interconnect fabric can lead to a series of access control vulnerabilities. SoCCom generated SoCs can be utilized for the vulnerability analysis of diverse access control scenarios stemming from ON-chip interactions between the trusted and untrusted components. While it is not feasible to mimic the access control scenarios with static IP analysis, the generated SoCs can be fully utilized for vulnerability analysis of system-level access control policies.

X. Conclusion

In this article, we presented a novel CAD framework and EDA tool for automated synthesis of optimized multicore and many-core SoC architectures. Our platform enables designers to generate synthesizable SoC RTL via rapid design space exploration and optimize designs for domain-specific applications in terms of area and power constraints. The open-source, extendable IP library allows users to build heterogeneous SoCs with diverse IPs. An upshot of the framework is that designers can quickly generate a wide variety of heterogeneous SoC benchmarks with varying IPs cores and interconnect fabric. The proposed SoC generation platform has a large application space and proven to be a crucial in various aspects of architecture and security research.

A key feature of SoCCom is its conceptual simplicity. The primary function of the tool flow described in this article is to connect (open-source) IPs through communication fabrics using a topology specification. However, such a view perhaps belies the non-triviality of the system. Note that SoC integration in practice requires months of human effort. The fact that SoCCom can ***automatically*** generate ***functioning*** SoCs that can be used as is for a variety of application tasks shows the value of the idea in addressing a critical bottleneck in industrial SoC designs. In particular, SoCCom shows how to streamline the integration process in practice through the disciplined specification of appropriate structural specification of interfaces, and identifies the architectural collateral needed to make the process automatic.

Our work paves the pathway to several promising research directions in SoC design automation, architectural exploration, security analysis, and design validation. The disciplined approach of automated SoC integration via standardized architectures enables benchmark generation with minimal effort while leading to the possibility of streamlined integration of diverse security features to SoC designs from the early stages of the design flow. In future work, we aim to explore the following: 1) extend SoCCom framework for fully automated integration of security wrappers and distributed satellite units as *local security brain* to target IPs; 2) enable high-level specification of security policies in a formal language representation and develop a SoCCom compatible framework for automated translation and integration of formally defined policies to actionable RTL design constraints and corresponding firmware; and 3) develop SoCCom as a comprehensive framework for automated generation of optimized and secure SoC platforms with integrated security policies and enable user flexibility in analyzing design power, area, and security tradeoffs.

References

[1] I. Parulkar, A. Wood, J. C. Hoe, B. Falsafi, S. V. Adve, J. Torrellas, and S. Mitra, "OpenSPARC: An open platform for hardware reliability experimentation," in *Proc. 4th Workshop Silicon Errors Log.-Syst. Effects (SELSE)*, 2008, pp. 1–6.

[2] F. Fatollahi-Fard, D. Donofrio, G. Michelogiannakis, and J. Shalf, "OpenSoC fabric: On-chip network generator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, 2016, pp. 194–203.

[3] S. Wallentowitz, A. Lankes, A. Zaib, T. Wild, and A. Herkersdorf, "A framework for open tiled manycore system-on-chip," in *Proc. 22nd Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2012, pp. 535–538.

[4] S. Bandara, A. Ehret, D. Kava, and M. A. Kinsy, "BRISC-V: An open-source architecture design space exploration toolbox," 2019, *arXiv:1908.09992*.

[5] M. A. Kinsy, M. Pellauer, and S. Devadas, "Heracles: A tool for fast RTL-based design space exploration of multicore processors," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2013, pp. 125–134.

[6] K. Asanovic *et al.*, "The rocket chip generator," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2016-17, 2016.

[7] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[8] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2009, pp. 33–42.

[9] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2009, pp. 469–480.

[10] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling energy optimizations in GPGPUs," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 487–498, 2013.

[11] T. Nowatzki, J. Menon, C.-H. Ho, and K. Sankaralingam, "Architectural simulators considered harmful," *IEEE Micro*, vol. 35, no. 6, pp. 4–12, Nov./Dec. 2015.

[12] Y. Xiao, Y. Xue, S. Nazarian, and P. Bogdan, "A load balancing inspired optimization framework for exascale multicore systems: A complex networks approach," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 217–224.

[13] Y. Xiao, S. Nazarian, and P. Bogdan, "Plasticity-on-chip design: Exploiting self-similarity for data communications," *IEEE Trans. Comput.*, vol. 70, no. 6, pp. 950–962, Jun. 2021.

[14] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2009, pp. 163–174.

[15] (2021). *System-On-Chip Benchmarks*. [Online]. Available: https://cadforassurance.org/soc-platform/soc-benign-benchmark/system-on-chip-benchmarks/

[16] A. Basak, S. Bhunia, and S. Ray, "A flexible architecture for systematic implementation of SoC security policies," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 536–543.

[17] A. Basak, S. Bhunia, and S. Ray, "Exploiting design-for-debug for flexible SoC security architecture," in *Proc. 53rd Annu. Design Autom. Conf.*, Jun. 2016, pp. 1–6.

[18] A. Basak, S. Bhunia, T. Tkacik, and S. Ray, "Security assurance for system-on-chip designs with untrusted IPs," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 7, pp. 1515–1528, Jul. 2017.

[19] A. P. D. Nath, S. Boddupalli, S. Bhunia, and S. Ray, "Resilient system-on-chip designs with NoC fabrics," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 2808–2823, 2020.

[20] X. Meng, K. Raj, A. P. D. Nath, K. Basu, and S. Ray, "SoCCAR: Detecting system-on-chip security violations under asynchronous resets," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, p. 309.

[21] H. Witharana and P. Mishra. (Aug. 2021). *Scalable Validation of System-On-Chip Vulnerabilities Using Security Assertions*. [Online]. Available: https://esl.cise.ufl.edu/Assert.html

[22] Y. Cao, H. Zheng, H. Palombo, S. Ray, and J. Yang, "A post-silicon trace analysis approach for system-on-chip protocol debug," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Nov. 2017, pp. 177–184.

[23] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: Threat analysis and countermeasures," *Proc. IEEE*, vol. 102, no. 8, pp. 1229–1247, Aug. 2014.

[24] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware Trojans: Lessons learned after one decade of research," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, pp. 1–23, Dec. 2016.

[25] J. Cruz and S. Bhunia. (2021). *Mimi: Automatic Hardware Trojan Insertion in a Design*. [Online]. Available: https://cadforassurance.org/tools/ic-trust-verification/mimi/

[26] R. S. Chakraborty and S. Bhunia, "HARPOON: An obfuscation-based SoC design methodology for hardware protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1493–1502, Oct. 2009.

[27] M. Goli, M. Hassan, D. Große, and R. Drechsler, "Security validation of VP-based SoCs using dynamic information flow tracking," *Inf. Technol.*, vol. 61, no. 1, pp. 45–58, Feb. 2019.

[28] M. Goli and R. Drechsler, "VIP-VP: Early validation of SoCs information flow policies using SystemC-based virtual prototypes," in *Proc. Forum Specification Des. Lang. (FDL)*, Sep. 2021, pp. 1–8.

[29] C. D. Systems, "Jaspergold formal verification platform," Tech. Rep., Aug. 2021.

Atul Prasad Deb Nath (Member, IEEE) received the B.Sc. degree from the Khulna University of Engineering and Technology (KUET), Khulna, Bangladesh, in 2011, the M.Sc. degree from the University of Toledo, Toledo, OH, USA, in 2016, and the Ph.D. degree from the University of Florida, Gainesville, FL, USA, in 2021, with a focus on investigating major aspects of SoC security and developing novel architectural features to protect assets, firmware, and ON-chip communication against various adversarial models.

He has published four book chapters, over 15 peer-reviewed journals and premiere conference papers, and filed three patents. His research interest includes system-on-chip (SoC) platform security and CAD for security and trust.

Kshitij Raj (Member, IEEE) is currently working toward the Ph.D. degree at the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA, as part of the Rising Laboratory, with a focus on the domain of automated integration and synthesis of secure silicon.

His research has been published in Design Automation Conference (DAC), Design, Automation and Test in Europe Conference (DATE), Asian HOST Conference, and so on. His research interests lie in the field of silicon architecture and design.

Swarup Bhunia (Senior Member, IEEE) received the B.E. degree (Hons.) from Jadavpur University, Kolkata, India, in 1995, the M.Tech. degree from IIT Kharagpur, Kharagpur, India, in 1997, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA, in 2005.

He was appointed as the T. and A. Schroeder Associate Professor of Electrical Engineering and Computer Science at Case Western Reserve University, Cleveland, OH, USA. He is currently the Director of the Warren B. Nelms Institute for the Connected World and a Semmoto Endowed Chair Professor of IoT with the University of Florida, Gainesville, FL, USA. He has over 250 publications in peer-reviewed journals and premier conferences. His current research interests include hardware security and trust, adaptive nanocomputing, and novel test methodologies.

Dr. Bhunia received the IBM Faculty Award, the NSF CAREER Award, the SRC Inventor Recognition Award, the SRC Technical Excellence Award, and several best paper awards/nominations. He has been serving as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS, the *ACM Journal of Emerging Technologies*, and the *Journal of Low Power Electronics*.

Sandip Ray (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA.

He is a Professor at the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA, where he holds an Endowed IoT Term Professorship. Before joining the University of Florida, he was a Senior Principal Engineer at NXP Semiconductors, and prior to that, a Research Scientist with the Intel Strategic CAD Laboratories. During his industry tenure, he led industrial research and research and development in pre-silicon and post-silicon validation of security and functional correctness of SoC designs, design-for-security and design-for-debug architectures, and security validation for automotive and the Internet-of-Things applications. He is the author of three books and over 100 publications in international journals and conferences. His current research targets correct, dependable, secure, and trustworthy computing through cooperation of specification, synthesis, architecture, and validation technologies.

Dr. Ray served as the Technical Program Committee Member of over 50 international conferences, as the Program Chair of ACL2 2009, FMCAD 2013, and IFIP IoT 2019, a Guest Editor for *IEEE Design & Test*, IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS, and *ACM Transactions on Design Automation of Electronic Systems*, and an Associate Editor for the *Journal of Hardware and Systems Security* (Springer) and IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS.