FALCON: An FPGA Emulation Platform for Domain-Specific SoCs (DSSoCs)

Anish Krishnakumar
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, WI 53706 USA

Hanguang Yu
Center for Wireless Information Systems and Computational Architectures (WISCA)
Arizona State University
Tempe, AZ 85281 USA

Tutu Ajayi
Department of Electrical Engineering and Computer Science (EECS)
University of Michigan
Ann Arbor, MI 48109 USA

A. Alper Goksoy and Vishrut Pandey
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, WI 53706 USA

Joshua Mack and Sahil Hassan
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721 USA

Kuan-Yu Chen
Department of Electrical Engineering and Computer Science (EECS)
University of Michigan
Ann Arbor, MI 48109 USA

Chaitali Chakrabarti and Daniel W. Bliss
Center for Wireless Information Systems and Computational Architectures (WISCA)
Arizona State University
Tempe, AZ 85281 USA

Ali Akoglu
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721 USA

Hun-Seok Kim, Ronald G. Dreslinski, and David Blaauw
Department of Electrical Engineering and Computer Science (EECS)
University of Michigan
Ann Arbor, MI 48109 USA

Umit Y. Ogras
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, WI 53706 USA

Digital Object Identifier 10.1109/MDAT.2023.3291331
Date of publication: 30 June 2023; date of current version: 22 January 2024.

2168-2364/23©2023 IEEE Copublished by the IEEE CEDA, IEEE CASS, IEEE SSCS, and TTTC

Editor's notes:
This article presents FALCON, a full-system domain-specific system-on-chip emulation platform that enables presilicon power and performance estimation of these platforms to provide support for early functional validation and software development.

—Sudeep Pasricha, Colorado State University, USA

With the slowdown of Moore's law, the ability of traditional homogeneous processors and single instruction set architecture (ISA) heterogeneous multicore architectures to satisfy the power and performance requirements has saturated [1]. Graphics processing units (GPUs), digital signal processors (DSPs), and hardware accelerators significantly improve the efficiency metrics at the cost of user programmability. Domain-specific SoC (DSSoC) architectures, which are a specific realization of heterogeneous architectures, bridge the gap between programmability and energy efficiency by smartly combining general-purpose, special-purpose, and hardware accelerator cores. The special-purpose and hardware accelerator cores strive to maximize the energy efficiency of applications in a targeted domain and the general-purpose processors provide programming flexibility [2].

SoC architectures, particularly DSSoCs, face monumental design and verification efforts due to rapidly increasing design sizes and complexities [3]. Fabricating a fully functional DSSoC design from scratch requires significant time, effort, and resources. Functional and performance bugs in these complex chips postfabrication result in unprecedented costs. Therefore, stringent presilicon verification techniques such as RTL simulation, gate-level simulation, formal verification, FPGA emulation, and prototyping frameworks are used to detect bugs in the early design stages. Furthermore, making even incremental design changes, such as adding a new accelerator, requires new spinoffs with similar costs. FPGA emulation is specifically used to address these challenges and offers the following advantages [4]: 1) enables execution of real-world workloads on the full system (significantly faster than simulation); 2) allows early firmware and software development; and 3) facilitates faster time-to-market. While FPGAs have been used in NoC and special-purpose architecture prototyping [5], [6], [7], [8], end-to-end frameworks do not exist for the emulation and prototyping DSSoCs on a Linux-based operating system (OS).

This article proposes FALCON, an end-to-end FPGA-based emulation framework, to prototype DSSoCs for rapid design, presilicon functional validation, and performance evaluation. FALCON provides an accelerator sandbox, which uses standard advanced microcontroller bus architecture (AMBA)-based interfaces to the rest of the SoC. The accelerator sandbox improves developers' productivity by providing a plug-and-play environment to include, remove, and modify hardware accelerators. FALCON also allows designers to develop drivers for nonstandard ISA designs, software, and firmware before the chip is available. It also enables what-if analysis with different hardware configurations and domain applications can be done much faster (in our experience in a few weeks) than waiting for the final chip. Finally, FALCON interfaces with compiler-integrated extensible DSSoC runtime (CEDR) [2], a software runtime framework, to allow applications to be seamlessly executed in a DSSoC.

FALCON architecture

This section describes FALCON's full-system architecture for DSSoC design and emulation, as outlined in Figure 1. FALCON is composed of the hardware platform and the software stack. While these components are typical in any SoC, DSSoCs are highly customized to maximize the energy efficiency of domain applications. The hardware platform integrates general-purpose cores that offer programmability, hardware accelerators, and specialized processors for energy efficiency, a high-speed interconnect for low-latency on-chip data movement, last-level cache (LLC), peripherals, and debug logic. After synthesis and automatic place-and-route, the entire hardware architecture is packaged into a bitstream to program the programmable logic (PL) on the FPGA. The software stack comprises the Linux OS kernel, file system, and embedded system software components such as the boot firmware and U-boot. All components in the software stack are integrated into a software image, which is programed into the FPGA flash memory. Then, applications run on the underlying hardware of the DSSoC with the use of software runtime environments, such as CEDR and SPARTA [2], [9].

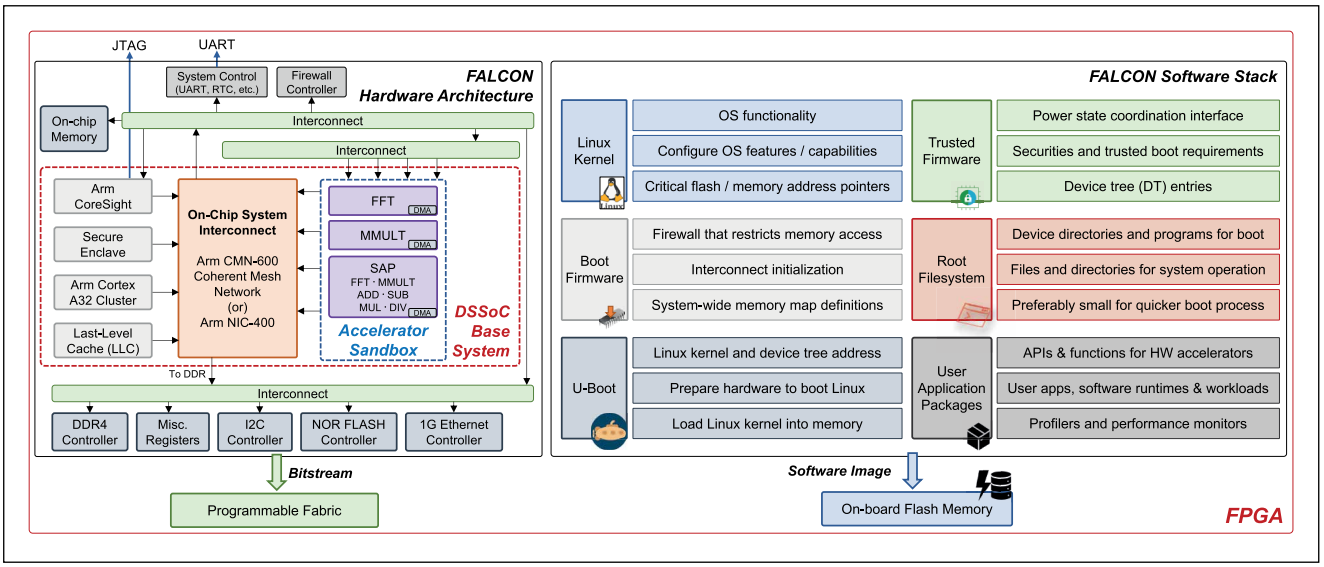

Figure 1. Overview of the key components and organization of the FALCON framework for DSSoC emulation on FPGAs. The hardware architecture and design (shown on the left) is programed as a bitstream onto the programmable fabric. The software image (shown on the right) is programed to the onboard flash memory. The accelerator sandbox includes a flexible SAP that efficiently implements operations such as FFT, matrix multiplication, and complex vector addition, subtraction, multiplication, and division.

Hardware architecture

The FALCON hardware architecture is constructed using three major components: 1) base system; 2) accelerator sandbox; and 3) miscellaneous hardware, controllers, and peripherals in the FPGA. The framework organizes the energy-efficient processors into the accelerator sandbox and the general-purpose processors with the on-chip system-level interconnect into the base system. In addition to the base system and the sandbox, FALCON includes peripherals, controllers, and other hardware, as shown in Figure 1.

Base system

The base system forms the general-purpose subsystem of the DSSoC. FALCON utilizes Arm's Corstone700 as the base system. Corstone-700 is a flexible and configurable subsystem that houses the 32-bit Arm Cortex-A32 cores as the processing cluster. It also provides easy and flexible interfaces to integrate other system components and peripherals. The number of Cortex-A32 cores is configurable between 1 and 4. An advanced extensible interface (AXI)-based interrupt controller distributes the interrupts to the different on-chip components. The secure enclave and CoreSight unit provides security and debug services. CoreSight facilitates functional and performance debugging. We note that the ratios of all clock frequencies are maintained to maintain accuracy with the final tape-out. The flexibility of the architecture allows the base subsystem to be easily swapped with other Corstone subsystems or potentially with different types of host systems.

Accelerator sandbox

The design and integration of components in a DSSoC are highly complex due to the large number and diverse processing elements. Therefore, the exploration phase involves frequent addition, modification, and removal of accelerators in a DSSoC. To address this concern, FALCON employs a modularized implementation of the interfacing of hardware accelerators with the base system. The accelerator sandbox is an independent module that uses standard AXI interfaces to connect to the system-level interconnect. The sandbox approach allows the rest of the system to observe only the AXI interfaces from the sandbox. It is oblivious to its internal architecture, providing a plug-and-play mechanism for integrating hardware accelerators. This architecture assumes that each accelerator can master the memory bus or work closely with other accelerators or

direct memory access (DMA) engines to transfer data to the system. The sandbox provides interrupt lines to the accelerators to indicate control transfer to the base subsystem. The sandbox uses four AXI initiator-responder channels. Designers map the chosen number of channels among the different hardware accelerators and their data streams based on the latency, bandwidth, and throughput requirements. A DSSoC that targets wireless communication and radar applications may integrate accelerators for fast Fourier transform (FFT), matrix multiplication, and a systolic array processor (SAP) to accelerate frequently encountered tasks as illustrated in Figure 1. The flexibility of the SAP enables us to execute several functionalities in the domain of interest with customized programming interfaces. The plug-and-play mechanism allows designers to provide intrasandbox communication between accelerators to improve data movement latencies. The sandbox can easily be extended to support multiple clocks and resets if the accelerators are required to operate at different frequencies.

On-chip system interconnect

With the diverse processing elements on the chip, data movement is critical to ensure that the hardware has the necessary inputs to perform the required computation. Developers may choose to integrate low-latency mesh NoC interconnects (such as Arm CMN-600) or low-power crossbar-based interconnects (such as Arm NIC-400).

While the base system and system-level interconnect use Arm-based components, FALCON is not limited to Arm-based systems, and developers are free to integrate processing elements and interconnects of their choice. We note that the software stack (described in the upcoming subsection) would need appropriate updates to support the hardware choices.

Software stack

DSSoCs demand an extensive software stack to exploit the full potential of the hardware architecture and provide comprehensive programming support to end-users and developers. FALCON is based on the Arm Corstone-700 base system. Hence, it utilizes the Arm reference platforms to produce the software stack [10]. The Arm reference platforms are based on the Yocto project to build customized Linux distributions. While this section describes configuring the Arm reference platforms for FALCON, the methodology is generic since the Yocto project is widely used to produce Linux distributions and software stacks. We emphasize that this software stack is fully deployable with standard security, virtualization, and the guarantees of a full-fledged Linux-based embedded system. This section focuses more on the specific configurations for the DSSoC configuration prototyped in this work.

The software stack integrates the following components to produce the entire software stack (as shown in Figure 1): 1) Linux kernel; 2) boot firmware; 3) trusted firmware; 4) U-boot; 5) root filesystem; and 6) application packages. The interactions between these components are captured in Figure 2.

Linux kernel

The primary responsibilities of the kernel include: 1) memory management; 2) process management; 3) device drivers; and 4) system calls and security. To support the hardware described in the previous subsection, FALCON makes the following modifications to the base configuration [10] for the Linux kernel.

- Enable multicore support through symmetric multiprocessing feature.
- Configure input–output memory management unit (IOMMU) for multiple cores.
- Enable kernel debugging capabilities.
- Configure power state coordination interface (PSCI) for multiple cores.
- Modify address pointers and image size of the software stack image stored in the flash memory.

Boot firmware

The boot firmware is the software for the secure enclave in the hardware architecture [10]. From the user perspective, the components that should be modified are firewall access, system-wide memory map definitions, and interconnect initialization. The firewall determines the accessible/restricted memory regions of: 1) the root file system; 2) the Linux kernel; and 3) the main memory. The boot firmware is compiled into a binary that is then built into the secure enclave hardware. Embedding the boot firmware into the hardware has a major implication in the DSSoC validation process, and this is precisely where early software development supported by FALCON plays a crucial role in developing bug-free and fully-functional SoCs (described in the upcoming section).

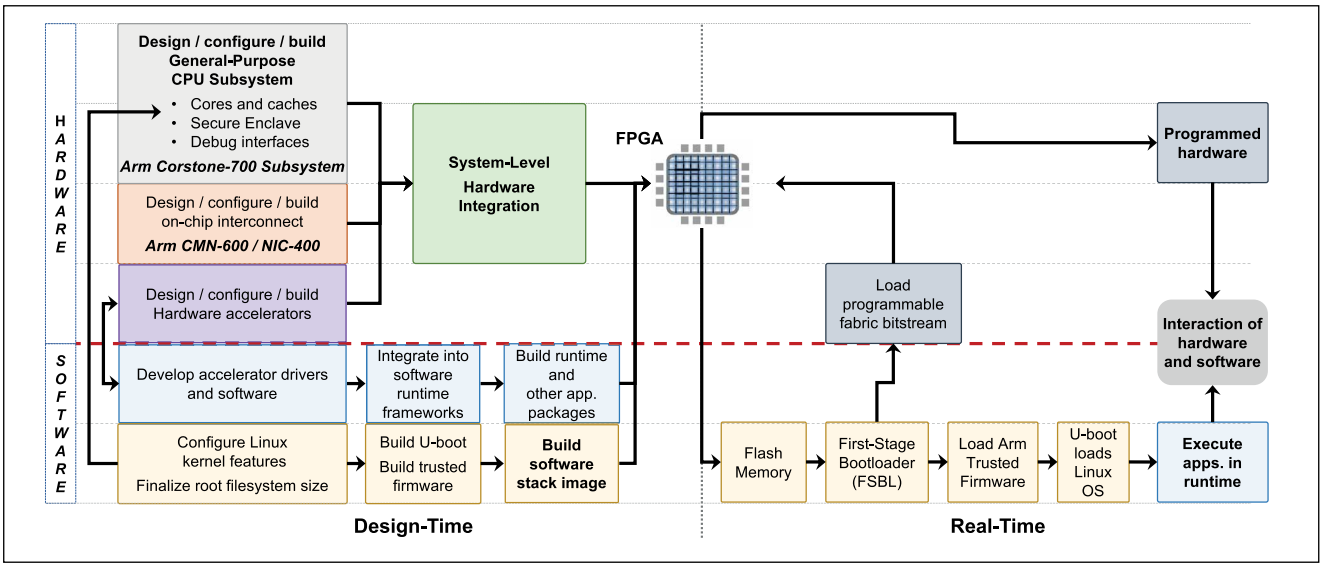

Figure 2. Timeline of the hardware development process and the boot sequence in software for a DSSoC emulated by the proposed FALCON framework.

U-Boot

This software comprises a first-stage boot loader (FSBL) and a second-stage boot loader (U-boot). It is the primary component that handles hardware initialization and control hand-off to the OS for the booting process. FALCON modifies the Linux kernel address based on its size in the software image and the device tree address in U-Boot.

Trusted firmware

The trusted firmware in FALCON comprises the critical security software for Arm-based processor systems. The default Corstone-700 stack boots only one Arm core. One of the most critical components is the PSCI which is the interface for managing the idle cores, booting the secondary cores, and system shutdown/reset. FALCON modifies the PSCI firmware to power on the secondary cores and enables realtime access to multiple cores. FALCON also adds helper threads with assembly code to initialize and boot the secondary cores. The secondary core information is also specified in the device trees as entries in 1) the PSCI interfaces and 2) central processing unit (CPU) cores. The device tree binary is built as part of the trusted firmware in FALCON's software stack.

Root filesystem

The OS's root filesystem (rootFS) contains the files and directories critical to the system's operation. By default, the Corstone-700 reference software stack provides a read-only file system. This requirement forces all the critical packages and features to be built into the rootFS during the build process. The packages to be integrated into the rootFS determine its size. It is also critical to reduce the rootFS size to minimize the boot time.

User application packages

The user applications range from libraries that include application programming interfaces (APIs) to exercise the hardware accelerators to workloads, benchmarks, profilers, and performance monitors. The domain workload and benchmark source codes are cross-compiled for the specific Arm architecture (32-bit Arm v8 architecture in FALCON) and packaged into the software stack. Additionally, performance-monitoring tools, such as *perf* that uses the performance-monitoring unit (PMU) to monitor the CPU pipelines and the interconnect, can be integrated to enable runtime performance monitoring and evaluation. The FALCON emulation framework utilizes the CEDR runtime framework, which is also deployed as a user application package.

Software runtimes

In this study for our experimental evaluations, we utilize the CEDR [2] ecosystem to conduct a design space exploration over the heterogeneous architecture emulated on the FPGA. This system allows end users to compile their applications for execution on

a heterogeneous architecture and then interact with hardware by launching a workload composed of any number and combination of different applications with user-specified arrival rates. We choose CEDR over other runtime frameworks [11], [12] since it enables the compilation and development of user applications for heterogeneous SoCs, evaluating the performance of presilicon heterogeneous hardware configurations based on dynamically arriving workload scenarios through distinct plug-and-play integration points in a unified workflow. Furthermore, CEDR offers a rich set of integrated scheduling policies, allows integration of new policies through its distinct plug-and-play interfaces, offers collecting performance counter-based performance evaluation through "perf" utility, and, more importantly, enables conducting design space exploration in the trade space of hardware composition, workload complexity, and scheduling policy over the user-defined performance metrics.

Enabling software and driver development

The efforts involved in software design and driver development for DSSoCs are substantially higher due to the presence of hardware accelerators and specialized cores. Software development after fabrication significantly delays the time-to-market. To this end, presilicon FPGA-based emulation frameworks serve as a platform for software development and hardware–software codesign cycle.

Accelerator drivers

While general-purpose cores have a well-established programming methodology in terms of programming languages and compilation toolchains, the hardware accelerator interfaces are mainly ad-hoc. They may not follow predefined protocols and languages. The interface to a hardware accelerator involves the following aspects: 1) a configuration interface that allows the user to configure the accelerator based on the application parameters; 2) a control interface that manages its initialization, starting, and completion; and 3) data interface for the inputs and outputs. It is critical to validate these interfaces and data transfer protocols in the design stage. Another aspect involves determining the optimal burst size for input and output data transfers and the memory hierarchy. Current approaches in the literature include analytical and performance models to estimate these effects, but the modeling accuracy limits them. Moreover, they are often evaluated with only portions of the system. FALCON enables evaluation in a full-system real-platform-like environment, providing highly accurate performance estimates. The precise evaluation allows designers to redesign the hardware and software architecture and interfaces as necessary to maximize metrics, including performance, throughput, bandwidth, and energy efficiency.

Enabling performance-monitoring unit

The PMU records architectural and microarchitectural events and provides key performance indicators (KPIs). KPIs allow users to profile the applications and fine-tune the system parameters and architecture to maximize performance. Enabling the PMU in FALCON requires several changes to the software stack. First, the following features are enabled in the Linux kernel: 1) CONFIG_PROFILING; 2) CONFIG_PERF_EVENTS; 3) CONFIG_ARM_PMU; and 4) CONFIG_HW_PERF_EVENTS.

The size of the Linux kernel in the software image increases when the PMU is enabled. This increase changes the address offsets and the size parameters in the boot firmware and the U-boot, as described in the previous subsection. While the above modifications described are in the software stack, they strongly affect the hardware design. As described in the previous subsection, the boot firmware includes addresses and sizes of the Linux kernel, rootFS, and the main memory, which are used in the secure enclave firewall. This information is packaged into the hardware design, making it infeasible to update these parameters after the fabrication. To this end, FALCON enables all these hardware–software codesign aspects to ensure that the fabricated chip supports all intended features and functionality.

Demonstrations Using FALCON

FALCON aids chip developers in performing functional design validation, identifying the optimal data flow for hardware accelerators, analyzing performance bottlenecks using hardware performance counters, and even performing early presilicon power evaluations. This article demonstrates the capabilities of FALCON for wireless communication and radar/signal-processing application domain. The applications in these domains frequently exercise FFT, matrix multiplication and vector addition, subtraction, multiplication, and division operations, which are extensively evaluated in this section. FALCON

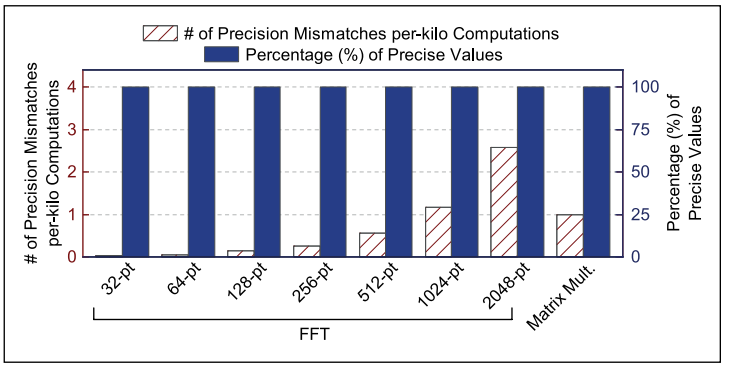

Figure 3. Number of precision mismatches (primary axis) per-kilo computations and the percentage of precise values (secondary axis) of the FFT and matrix multiplication hardware accelerators with respect to a reference software implementation in VCU128.

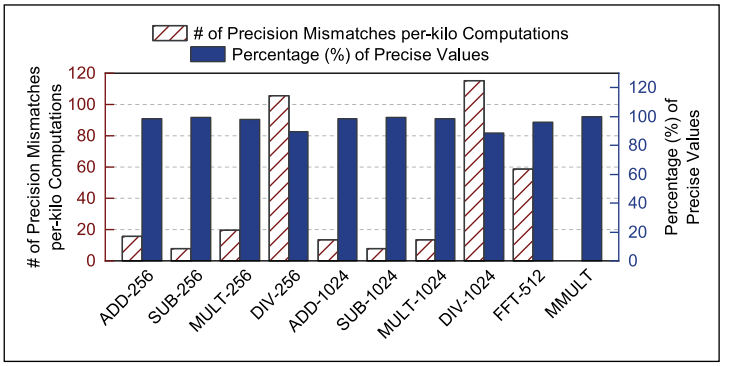

Figure 4. Number of precision mismatches (primary axis) per-kilo computations and the percentage of precise values (secondary axis) of the SAP accelerator (16-bit fixed point implementation) for vector addition, subtraction, multiplication, and division (for two sizes: 256 and 1024), FFT (512-point), and matrix multiplication operations with respect to reference software implementations in VCU128.

is evaluated using three Xilinx FPGA devices: Zynq UltraScale+ ZCU102, Virtex UltraScale+ VCU128, and Virtex UltraScale+ VU19P. Only the accelerator sandbox is deployed on the ZCU102 since it includes prebuilt Arm Cortex-A53 cores (Zynq base system). We leverage the CEDR runtime environment to launch and run the target applications.

Enabling software development and functional validation

We developed software drivers for the hardware accelerators in the sandbox, which send and receive data from the system using DMA units. We generate random stimuli as inputs to the hardware accelerators. The outputs of the accelerators are compared with a reference software implementation for functional validation and precision evaluation.

For the FFT accelerator, we evaluated transform sizes from 32 to 2,048 (in multiples of 2) as shown in Figure 3. The number of precision mismatches remains at fewer than three per-kilo computations. A larger transform size experiences higher mismatches due to a higher number of floating-point multiplication and accumulation operations. The fixed-size complex matrix multiplication accelerator experiences an average of one precision mismatch per-kilo computations. On average, more than 99.9% of the values in the computations match the software implementation within a 0.1% difference.

The SAP accelerator is implemented with 16-bit fixed-point precision to tradeoff accuracy for performance and power consumption. The goal behind this idea is to use high-precision fixed-function accelerators for precision-demanding applications such as wireless communications and flexible energy-efficient accelerators for radar system applications. Figure 4 presents the accuracy (in percent) and mismatches per-kilo computations for various operations in the SAP. Except for vector division, other operations achieve a minimum of 93% accuracy compared to the respective software implementations. Since division is a highly complex operation, it suffers from larger accuracy differences because of the fixed-point implementation (especially for larger sizes). We note that these results well align with the requirements of radar applications.

FALCON specifically allows all these explorations before fabrication to evaluate functionality, precision, and accuracy for different applications in the domain. In summary, the framework enables software driver development, functional validation of the drivers, and accelerators when exercised with the full system and evaluation of the precision requirements.

Optimizing DSSoC configuration for domain applications

Hardware accelerators introduce the notorious double-copy problem where the data must be explicitly transferred from/to them using DMA units. While fetching the data from the main memory

involves significant latency overheads, the on-chip scratchpad memory (SPM) has limited capacity. Figure 5 presents the latency (in thousands of equivalent CPU clock cycles) for 128-, 256-, and 512-point FFT operations and complex matrix multiplication computation. The latency improves substantially with the use of SPM for larger computations (larger transform sizes). Larger computations require more memory and experience significant conflicts in the cache. Therefore, they experience better speedup when data are transferred using the on-chip scratchpad. Through such analyses, FALCON allows us to optimize the number of processors, cache sizes, and memory hierarchies for the processing elements for specific domain applications to maximize energy efficiency.

Illustration of hardware performance counters

Table 1 presents the hardware counters (described in the previous subsection) for two implementations of a matrix multiplication operation. The two implementations use different loop ordering to improve data locality in the caches, reflecting fewer cache misses, and, consequently, fewer computation cycles and wall time (Table 1). The ability to utilize performance counters in FALCON enables users and developers to systematically analyze the effects of code optimization and their impact in terms of microarchitectural events.

Enabling presilicon power evaluations

Figure 6 presents the power consumption in the processing system (PS) and programmable logic (PL) in a ZCU102 FPGA using four Arm Cortex-A53 cores at 1.2 GHz, one FFT, and one complex matrix multiplication accelerator operating at 100 MHz. The power consumption is captured using TI INA226 power monitors on the board. We run 100 jobs each of *pulse Doppler* and *temporal mitigation* compiled for heterogeneous execution and managed using CEDR [2]. Figure 6 presents the idle and active PS and PL power for 100 s. At time 0, the PS consumes 1.6 W, while PL consumes 1.322 W when the system is idle. The PS power increases to 2.2 W when the applications start execution. The change in PL power remains insignificant because of its lower operating frequency and high static/idle power in the FPGA logic. The PS power reduces at 52 s when *pulse Doppler* completes and reduces to idle power of 1.6 W at 58 s when the *temporal mitigation* also ends. This capability allows developers to obtain early estimates and observe power consumption trends for preliminary analysis before the chip is taped out.

FPGA implementation strategy

Although FALCON shares a similar design as the actual chip, there are a few critical aspects that must be addressed. First, a failure to generate the intended gated clocks in the design causes the tool to use cascaded clocks instead of generating a clock tree on the FPGA. This failure can also cause long placement and routing runtimes. While the tool must automatically generate gated clocks, developers must carefully check the gated clock conversion reports after synthesis and manually specify the clock relationships as required. Second, the design may instantiate memory modules from a specific process technology library. We should replace the memory instantiations with the FPGA memory primitive or a generic memory model. Third, the synthesis,

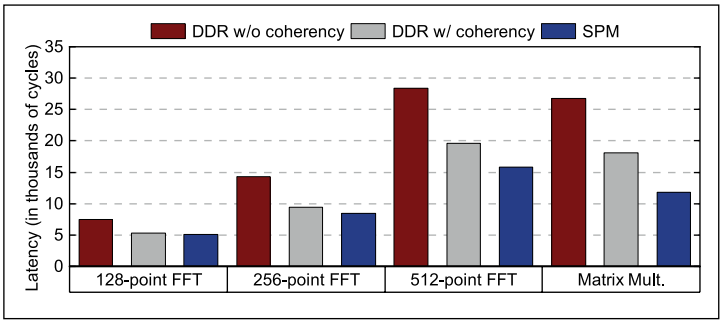

Figure 5. Comparison of the hardware accelerator latencies (in thousands of cycles) when data are transferred through DDR memory (with and without coherency) and the on-chip SPM in VCU128.

Table 1. Comparison of hardware performance counter values between two different software implementations of matrix multiplication in VCU128 operating at a frequency of 32 MHz.

HW Perf. Counter	Implementation 1	Implementation 2
Task-clock	640.2 seconds	495.6 seconds
Context-switches	687	164
Cpu-migrations	0	1
Page faults	751	996
Cycles	20.5 B	15.8 B
Instructions	14.8 B	9.5 B
Branches	1.0 B	0.14 B
Branch-misses	5.9 M (0.58%)	2.2 M (1.59%)
Cache-references	1.5 B	1.9 B
Cache-misses	125 M (8.6%)	5.3 M (0.274%)

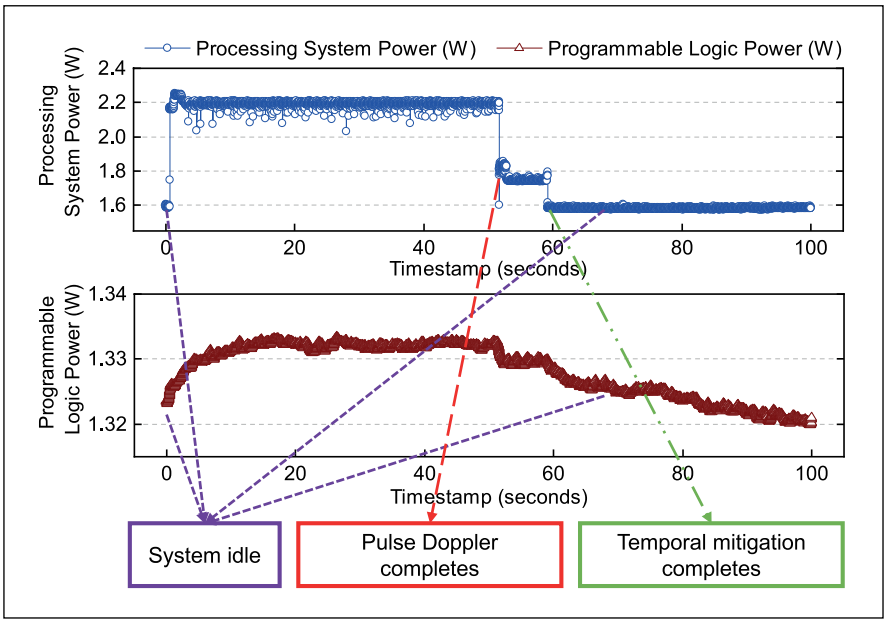

Figure 6. Illustration of power consumption in the PS (Arm Cortex-A53 cores operating at 1.2 GHz) and programmable fabric (hardware accelerators operating at 100 MHz) in a Zynq UltraScale+ ZCU102 FPGA.

placement, and routing strategy should be tuned to the specific FPGA and the size of the SoC. We use the *AlternateRoutability* for the accelerator sandbox synthesis and the *SpreadLogic* at the top level. Finally, we achieve timing closure at 32-MHz main clock frequency on the VCU128 and VU19P FPGAs.

Related work

Emulation frameworks play a crucial role in enabling presilicon functional validation, software, driver, and firmware development and are broadly classified into two categories. Virtual emulators, such as quick emulator (QEMU) and fast models, use abstracted functional models to design drivers, software, and firmware. FPGA-based platforms, representing the other class of emulators, are widely used for NoC and reconfigurable architecture prototyping [6], [8]. However, their realization of full-system emulation is limited. The cyber-physical SoC in [7] and an RISC-V accelerator-based system in [4] emulated on FPGAs are promising examples of full systems emulated on FPGAs. The framework in [7] does not support realtime power monitoring and a full-fledged Linux OS which is critical to enable full parallelism in the software runtime. The HERO framework [4] programs RISC-V-based accelerators in the FPGA but utilize prebuilt Arm cores in the FPGA and relies on the OpenMP framework for accelerator programming. Fully custom hardware accelerators in DSSoCs demand extensive function and performance validation and support for driver development. To the best of our knowledge, FALCON is the first emulation platform that supports a fully customized SoC designed from scratch with hardware accelerators evaluated on a customized Linux OS. FALCON performs end-to-end emulation of DSSoCs to support as-is tape out for fabrication, which is critical for full system and top-level functional validation.

The growing design and verification complexities in DSSoCs and the prohibitive cost of identifying postfabrication functional and performance bugs raise the need for sophisticated presilicon evaluation frameworks. Furthermore, the stringent time-to-market requirements demand the availability of firmware and software in conjunction with the OS as soon as possible after the chip becomes available. This article presented FALCON, an end-to-end full-system FPGA-based emulation framework for DSSoCs that integrates general-purpose processors, hardware accelerators, memory hierarchies, and on-chip interconnects to address these challenges.

FALCON is seamlessly integrated with a software runtime framework named CEDR to execute both domain and nondomain applications on the DSSoC efficiently. FALCON's hardware architecture and software stack, coupled with a runtime environment, enable realistic application execution in a Linux-based OS and allow full-system functional validation and early performance estimates. ■

Acknowledgments

This work was supported by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under Agreement FA8650-18-2-7860. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL and DARPA or the U.S. Government.

References

[1] J. Hennessy and D. Patterson, "A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 27–29.

[2] J. Mack et al., "CEDR: A compiler-integrated, extensible DSSoC runtime," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 2, pp. 1–34, Mar. 2023.

[3] Z. Han et al., "IP-coding style variants in a multi-layer generator framework," in *Proc. Design Verification Conf. Exhib. (DVCon)*, 2020, pp. 1–6.

[4] A. Kurth et al., "HERO: Heterogeneous embedded research platform for exploring RISC-V manycore accelerators on FPGA," 2017, *arXiv:1712.06497*.

[5] J. Zhang et al., "MEG: A RISCV-based system emulation infrastructure for near-data processing using FPGAs and high-bandwidth memory," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 4, pp. 1–24, Dec. 2020.

[6] V. Kumar, M. Mukherjee, and J. Lloret, "Reconfigurable architecture of UFMC transmitter for 5G and its FPGA prototype," *IEEE Syst. J.*, vol. 14, no. 1, pp. 28–38, Mar. 2020.

[7] S. Sarma and N. Dutt, "FPGA emulation and prototyping of a cyberphysical-system-on-chip (CPSoC)," in *Proc. 25nd IEEE Int. Symp. Rapid Syst. Prototyping*, Oct. 2014, pp. 121–127.

[8] S. Lotlikar, V. Pai, and P. V. Gratz, "AcENoCs: A configurable HW/SW platform for FPGA accelerated NoC emulation," in *Proc. 24th Int. Conf. VLSI Design*, Jan. 2011, pp. 147–152.

[9] B. Donyanavard et al., "SPARTA: Runtime task allocation for energy efficient heterogeneous manycores," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Oct. 2016, pp. 1–10.

[10] *Arm Reference Platform for Corstone-700 Subsystem*. Accessed: May 15, 2022. [Online]. Available: https://git.linaro.org/landing-teams/working/arm/arm-reference-platforms.git/tag/?h=CORSTONE-700-2020.12.10

[11] B. Donyanavard et al., "SOSA: Self-optimizing learning with self-adaptive control for hierarchical system-on-chip management," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, vol. 52. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 685–698.

[12] B. Maity et al., "SEAMS: Self-optimizing runtime manager for approximate memory hierarchies," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5, pp. 48:1–48:26, Jul. 2021.

Anish Krishnakumar is a staff engineer at Arm, Austin, TX 78735 USA. His research interests include machine-learning-based task scheduling and power modeling of heterogeneous domain-specific SoCs (DSSoCs). Krishnakumar has a PhD from the University of Wisconsin-Madison, Madison, WI, USA.

Hanguang Yu is an assistant research professor at Arizona State University (ASU), Tempe, AZ 85281 USA. His research interests include communications and high-precision positioning systems. Yu has an MSEE and a PhD from ASU.

Tutu Ajayi is a research fellow at the University of Michigan, Ann Arbor, MI 48109 USA. His research interests include primarily in cross-layer computer architecture, with an emphasis on realizing research into ASIC and FPGA platforms. Ajayi has a PhD in electrical and computer engineering from the University of Michigan.

A. Alper Goksoy is pursuing a PhD in electrical and computer engineering at the University of Wisconsin-Madison, Madison, WI 53706 USA. His research interests include task scheduling for

domain-specific SoCs (DSSoCs), in-memory computing, and on-chip learning. Goksoy has a BS in electrical and electronics engineering from Bogazici University, Istanbul, Turkey.

Vishrut Pandey is working as a senior engineer at Qualcomm, San Diego, CA 92121 USA. Pandey has a masters in electrical and computer engineering from the University of Wisconsin-Madison, Madison, WI, USA.

Joshua Mack is pursuing a PhD in the Electrical and Computer Engineering Program at the University of Arizona, Tucson, AZ 85721 USA. His research interests include reconfigurable systems, emerging architectures, and intelligent workload partitioning across heterogeneous systems.

Sahil Hassan is pursuing a PhD in the Electrical and Computer Engineering Program at the University of Arizona, Tucson, AZ 85721 USA. His research interests involve the design of reconfigurable and heterogeneous computing systems and neuromorphic architectures. Hassan has an MSc in electrical and electronic engineering from the University of Dhaka, Dhaka, Bangladesh.

Kuan-Yu Chen is pursuing a PhD at the University of Michigan, Ann Arbor, MI 48109 USA. His current research interests include digital circuit design, accelerators, and computer architecture. Chen has an MS in electrical and computer engineering from the University of Michigan. He is a Student Member of IEEE.

Chaitali Chakrabarti is a professor in the School of ECEE, at Arizona State University (ASU), Tempe, AZ 85281 USA. Her research interests include VLSI algorithm-architecture codesign of signal processing and communication systems and all aspects of low-power embedded systems. Chakrabarti has a PhD from the University of Maryland, College Park, MD, USA. She is a Fellow of IEEE.

Daniel W. Bliss is a professor in the School of Electrical, Computer, and Energy Engineering, at Arizona State University (ASU), Tempe, AZ 85281 USA. He is also the director of ASU's Center for Wireless Information Systems and Computational Architectures, Tempe. Bliss has a PhD an MS in physics from the University of California at San Diego, La Jolla, CA, USA.

Ali Akoglu is a professor in the Department of Electrical and Computer Engineering, at the University of Arizona, Tucson, AZ 85721 USA. His research focuses on high-performance computing and nontraditional computing architectures. Akoglu has a PhD in computer science from Arizona State University, Tempe, AZ, USA.

Hun-Seok Kim is an associate professor at the University of Michigan, Ann Arbor, MI 48109 USA. His research focuses on system analysis, novel algorithms, and efficient VLSI architectures for low-power/high-performance wireless communication, signal processing, computer vision, and machine-learning systems. Kim has a PhD in electrical engineering from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA. He is a Senior Member of IEEE.

Ronald G. Dreslinski is an assistant professor in the Computer Science and Engineering Department, at the University of Michigan, Ann Arbor, MI 48109 USA. His research interests include near-threshold computing (NTC), architectural simulator development, and high-radix on-chip interconnects. Dreslinski has a PhD in computer science and engineering from the University of Michigan.

David Blaauw has been on the Faculty of the University of Michigan, Ann Arbor, MI 48109 USA, where he is the Kensall D. Wise Collegiate Professor of EECS. He is the director of the Michigan Integrated Circuits Lab, Ann Arbor. Blaauw has a PhD in computer science from the University of Illinois at Urbana–Champaign, Champaign, IL, USA.

Umit Y. Ogras is an associate professor at the University of Wisconsin-Madison, Madison, WI 53706 USA. His research interests include embedded systems, heterogeneous SoCs, low-power VLSI, wearable computing, and flexible hybrid electronics. Ogras has a PhD in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA. He is a Senior Member of IEEE.

■ Direct questions and comments about this article to Anish Krishnakumar, Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706 USA; anish.n.krishnakumar@wisc.edu.