DS3: A System-Level Domain-Specific System-on-Chip Simulation Framework

Samet E. Arda, Anish NK, A. Alper Goksoy, Nirmal Kumbhare, Joshua Mack, Anderson L. Sartor, Ali Akoglu, Radu Marculescu and Umit Y. Ogras

Abstract— Heterogeneous systems-on-chip (SoCs) are highly favorable computing platforms due to their superior performance and energy efficiency potential compared to homogeneous architectures. They can be further tailored to a specific domain of applications by incorporating processing elements (PEs) that accelerate frequently used kernels in these applications. However, this potential is contingent upon optimizing the SoC for the target domain and utilizing its resources effectively at runtime. To this end, system-level design - including scheduling, power-thermal management algorithms and design space exploration studies - plays a crucial role. This paper presents a system-level domain-specific SoC simulation (DS3) framework to address this need. DS3 enables both design space exploration and dynamic resource management for power-performance optimization of domain applications. We showcase DS3 using six real-world applications from wireless communications and radar processing domain. DS3, as well as the reference applications, is shared as open-source software to stimulate research in this area.

Index Terms—Heterogeneous computing, System-on-Chip (SoC), domain-specific SoC, simulation framework, scheduling, dynamic thermal-power management (DTPM), design space exploration.

1 Introduction

Homogeneous general-purpose processors provide flexibility to implement a variety of applications and facilitate programmability. However, these platforms cannot take advantage of the domain knowledge to optimize the energy efficiency for specific application domains, such as machine learning, communication protocols, and autonomous driving [1], [2], [3]. In contrast, heterogeneous systems-on-chip (SoCs) that combine general purpose and specialized processors (e.g., audio/video codecs and communication modems) offer great potential to achieve higher efficiency [4]. In particular, domain-specific SoCs (DSSoCs) - a class of heterogeneous architectures - optimize the architecture, computing resources and design flows by exploiting the characteristics of applications in a domain. For a given target domain, DSSoCs can provide three orders of magnitude higher energy-efficiency in comparison to general-purpose processors [5].

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

S. E. Arda, A. NK, A. A. Goksoy and U. Y. Ogras are with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA. E-mail: {sarda1, anishnk, aagoksoy, umit}@asu.edu

N. Kumbhare, J. Mack and A. Akoglu are with the Electrical and Computer Engineering Dept., University of Arizona, Tucson, AZ 85719 USA. E-mail: {nirmalk, jmack2545, akoglu}@email.arizona.edu

A. L. Sartor is with the Electrical and Computer Engineering Dept., Carnegie Mellon University, Pittsburgh, PA 15213 USA. E-mail: {asartor}@cmu.edu

R. Marculescu is with the Electrical and Computer Engineering Dept., University of Texas at Austin, Austin, TX 78712 USA. E-mail: {radum}@utexas.edu

Harvesting the full potential of DSSoCs depends critically on the integration of optimal combination of computing resources and their effective utilization and management at runtime. Hence, the first step in the design flow includes analysis of the domain applications to identify the commonly used kernels [6]. This analysis aids in determining the set of specialized hardware accelerators for the target applications. For example, DSSoCs targeting wireless communication applications obtain better performance with the inclusion of Fast-Fourier Transform (FFT) accelerators. Similarly, SoCs optimized for autonomous driving applications integrate deep neural network (DNN) accelerators [7]. Then, a wide range of design- and run-time algorithms are employed to schedule the applications to the processing elements (PEs) in the DSSoC [8], [9], [10], [11]. Finally, dynamic power and thermal management (DTPM) techniques optimize the SoC for energy efficient operations at runtime. Throughout this process, evaluation frameworks, ranging from analytical models and hardware emulation, are needed to explore the design space and ensure that the DSSoC achieves performance, power and energy targets [5].

Full-system simulators, like gem5 [12], have the ability to perform instruction-level cycle-accurate simulation. However, this level of detail leads to long execution times, in the order of hours to simulate a few milliseconds of workloads [13]. Hence, they are not suitable for rapid design space exploration (DSE). It is also important to note that the level of detail provided by cycle-accurate simulations is beyond the requirements of high-level design space exploration. The most critical system-level questions are *where tasks should run* and *how fast PEs should operate* to satisfy the design requirements, e.g., maximizing performance per Watt (PPW) or energy-delay product (EDP). On the other hand, hardware emulation using

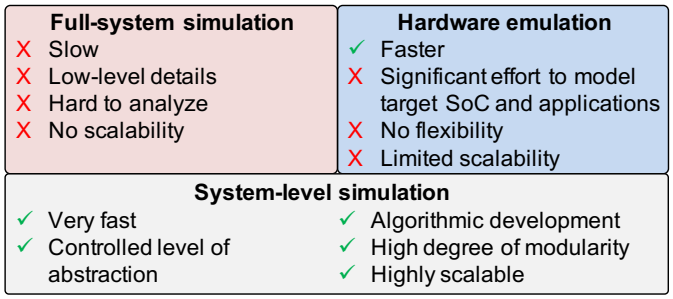


Fig. 1: Common DSE methodologies with advantages (✓) and disadvantages (X).

Field-Programmable Gate Array (FPGA) prototypes are substantially faster [14]. However, they involve significantly higher development effort to implement the target SoC and applications (see Figure 1). Given the design complexities and the cost of considering a large design space, there is a strong need for a simulation environment which allows rapid, high-level, simultaneous exploration of scheduling algorithms and power-thermal management techniques, both of which can significantly influence energy efficiency.

In this paper, we present DS3, a system-level domain-specific system-on-chip simulation framework. DS3 framework enables (1) run-time scheduling algorithm development, (2) DTPM policy design, and (3) rapid design space exploration. To this end, DS3 facilitates plug-and-play simulation of scheduling algorithms; it also incorporates built-in heuristic and table-based schedulers to aid developers and provide a baseline for users. DS3 also includes power dissipation and thermal models that enable users to design and evaluate new DTPM policies. Furthermore, it features built-in dynamic voltage and frequency scaling (DVFS) governors, which are deployed on commercial SoCs. Besides providing representative baselines, this capability enables users to perform extensive studies to characterize a variety of metrics, PPW and EDP for a given SoC and set of applications. Finally, DS3 comes with *six reference applications* from wireless communications and radar processing domain. These applications are profiled on heterogeneous SoC platforms, such as Xilinx ZCU102 [15] and Odroid-XU3 [16], and included as a benchmark suite in DS3 distribution. The benchmark suite enables realistic design space explorations, as we demonstrate in this paper.

The major contributions of this work include:

- A unified, high-level DSSoC simulator, DS3, that enables design space exploration together with scheduling and DTPM strategies,
- A benchmark suite of real-world applications and their reference hardware implementations and
- Extensive design space exploration studies for fine-grained architecture tuning.

The rest of this paper is organized as follows. The related work is reviewed in Section 2. The goals and architectural details of DS3 are presented in Section 3. Section 4 elaborates the implementation details of DS3 and Section 5 presents the built-in capabilities. Section 6 describes the validation of the framework against a real platform while Section 7 presents the case studies using real-world applications. Finally, the conclusions and future work are discussed in Section 8.

2 Related Work

As the use cases for this environment intersect with a number of distinct research areas, we break the related work into three parts. First, we discuss existing work in the area of scheduling, power, and thermal optimization algorithms, and we motivate a need for a unified framework that integrates these with rich design space exploration capabilities. Second, we discuss existing work in the area of design space exploration for embedded systems, and we note the lack of rich support for thermal/power models or plug-and-play scheduling frameworks. Third, for completeness, we give a brief overview of related works in the scope of high performance computing or non-embedded environments. Together, this set of related works serves to motivate the need for an environment such as DS3 that unifies all of these aspects into a single, open-source framework for embedded DSSoC development.

Starting with works on scheduling, power, and thermal optimization algorithms, one of the most important goals of heterogeneous SoC design is to optimize energy-efficiency while satisfying the performance constraints. To this end, a variety of offline and runtime algorithms have been proposed to schedule applications to PEs in multi-core architectures [8], [9], [10], [11]. Similarly, DVFS policies, such as HiCAP [17], power management governors, such as *ondemand* [18], and thermal management techniques [19] have been proposed to efficiently manage the power and temperature of SoCs. However, existing approaches are typically evaluated in isolated environments and different in-house tools. Hence, there is a strong need for a unified simulation framework [20] to compare and evaluate various scheduling algorithms in a common environment.

Next, there are a large number of works on design space exploration for embedded systems, but they are found to be lacking in support for rich scheduling, thermal, and power optimization algorithms. Khalilzad et al. [21] consider a heterogeneous multiprocessor platform along with applications modeled as synchronous dataflow graphs and periodic tasks. The design space exploration problem is solved using a constraint programming solver for different objectives such as deadline, throughput, and energy consumption. ASpmT [22] proposes a multi-objective tool using Answer Set Programming (ASP) for heterogeneous platforms with a grid-like network template and applications specified as directed acyclic graphs (DAGs). Trčka et al. [23] utilize the Y-chart [24] philosophy for design space exploration and introduces an integrated framework using the Octopus toolset [25] as its kernel module. Then, for different steps in the exploration process (i.e., modeling, analysis, search, and diagnostics), different languages and tools such as Ptolemy, Uppaal, and OPT4J are employed. Target platforms and applications are modeled in the form of an intermediate representation to support translation from different languages and to different analysis tools. Artemis [26] aims to evaluate embedded-systems architecture instantiations at multiple abstraction levels. Later, authors extend the work and introduce the Sesame framework [27] in which target multimedia applications are modeled as Kahn Process Network (KPN) written in C/C++. Architecture models, on the other hand,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2986963, IEEE Transactions on Computers

include components such as processor, buffers, and buses and are implemented in SystemC. The framework supports different schedulers such as first in, first-out (FIFO), round-robin, or customized. A trace-driven simulation is applied for cosimulation of application and architecture models.

Finally, ReSP [28] is a virtual platform targeting multiprocessor SoCs focusing on a component-based design methodology utilizing SystemC and transaction-level modeling libraries. ReSP adopts lower-level instruction set based simulation approach and is restricted to applications implemented in SystemC. All aforementioned frameworks or tools lack accurate power and thermal models, and do not support for exploration of scheduling algorithms and power-thermal management techniques.

Outside of embedded systems, there has also been a large body of work on design space exploration via heterogeneous runtimes at the desktop or HPC scale, with StarPU [29] being one of the most prominent examples of such a runtime. StarPU is a comprehensive framework that provides the ability to perform run-time scheduling and execution management for DAG based programs on heterogeneous architectures. Although, the framework allows users to develop new scheduling algorithms, StarPU lacks power-thermal models and DVFS techniques to optimize power and energy consumption. A recent work [30] targets domain-specific programmability of heterogeneous architectures through intelligent compile-time and run-time mapping of tasks across CPUs, GPUs, and hardware accelerators. In the proposed approach, the authors employ four different simulators, more specifically, Contech to generate traces, MacSim to model CPU/GPU architectures, BookSim2 to model the networks-on-chip, and McPat to predict energy consumption. The proposed DS3 simulator integrates the above features in a unified framework to benefit similar studies in the future.

To the best of our knowledge, DS3 is the first open-source framework to integrate all of these distinct elements into a unified simulation environment targeting embedded DSSoCs. It includes built-in analytical models, scheduling algorithms, DTPM policies, and six reference applications from wireless communication and radar processing domain.

3 Overall Goals and Architecture

The goal of DS3 is to enable rapid development of scheduling algorithms and DTPM policies, while enabling extensive design space exploration. To achieve these goals, it provides:

- **Scalability:** Provide the ability to simulate instances of multiple applications simultaneously by streaming multiple jobs from a pool of active domain applications.
- **Flexibility:** Enable the end-users to specify the SoC configuration, target applications, and the resource database swiftly (e.g., in minutes) using simple interfaces.
- **Modularity:** Enable algorithm developers to modify the existing scheduling and DTPM algorithms, and add new algorithms with minimal effort.
- **User-friendly Productivity Tools:** Provide built-in capabilities to collect, report and plot key statistics, including power dissipation, execution time, throughput, energy consumption, and temperature.

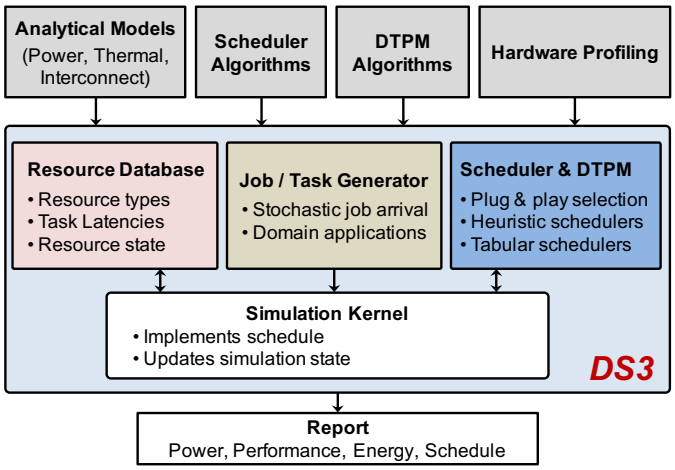


Fig. 2: Organization of DS3 framework describing the inputs and key functional components to perform rapid design space exploration and validation.

The organization of the DS3 framework designed to accomplish these objectives is shown in Figure 2. The resource database contains the list of PEs, including the type of each PE, capacity, operating performance points (OPP), among other configurations. By exploiting the deterministic nature of domain applications, the profiled latencies of the tasks are also included in the resource database. The simulation is initiated by the job generator, which generates application representative task graphs. The injection of applications in the framework is controlled by a random exponential distribution. The DS3 framework invokes the scheduler at every scheduling decision epoch with the list of tasks ready for execution. Then, the simulation kernel simulates task execution on the corresponding PE using execution time profiles based on reference hardware implementations. Similarly, DS3 employs analytical latency models to estimate interconnect delays on the SoC [31]. After each scheduling decision, the simulation kernel updates the state of the simulation, which is used in subsequent decision epochs. In parallel, DS3 estimates power, temperature and energy of each schedule using power models [32]. The framework aids the design space exploration of dynamic power and thermal management techniques by utilizing these power models and commercially used DVFS policies. DS3 also provides plots and reports of schedule, performance, throughput and energy consumption to help analyze the performance of various algorithms.

DS3 is released to public as a companion to this paper [1]. Following two sections present the implementation details and capabilities for new developers and users, respectively.

4 Developer View: DS3 Implementation

This section describes the implementations of the DS3 components (Figure 2) from a developer's perspective.

4.1 Resource Database

DS3 enables instantiating a wide range of SoC configurations with different types of general- and special-purpose PEs. A list of PEs and its characteristics are stored

1. https://github.com/segemena/DS3

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2986963, IEEE Transactions on Computers

TABLE 1: List of PE attributes in resource database

	Attribute	Description
Static	**Type**	Defines type of PE Example: CPU, accelerator etc.
	Capacity	Number of simultaneous threads a PE can execute
	DVFS policy	Policy which controls PE frequency and voltage at runtime
	Operating performance point	Operating frequencies and corresponding voltages for a PE
	Execution time profile	Defines the execution time of supported tasks on each PE
	Power consumption profile	Provides the power consumption profile of each PE
Dynamic	**Utilization**	Defines active time of a PE for a particular time window
	Blocking	Probability that a PE is busy when a task is ready
	State	Indicates whether a PE is busy or idle

in the resource database. Each PE in the database has the *static* and *dynamic attributes* described in Table 1.

The *static* and *dynamic* attributes are determined based on the current industry practice and upon a careful examination of available systems on the market. For example, CFS scheduler, the default Linux kernel scheduler [33], makes task mapping decisions based on the utilization of PEs. In addition, ARM big.LITTLE architecture [32], combining Cortex-A15 cluster with energy-efficient Cortex-A7 cluster, supports different operating frequencies for each cluster. The voltage level and thus energy consumption depend on the operating point and DS3 takes these effects into account. Finally, commercial SoCs utilize DVFS policies [18] to control power and performance of PEs. For this reason, we integrated these policies into DS3 and assigned the current DVFS policy as an attribute to a PE. This list in Table 1 can be extended either by defining a new parameter in the corresponding SoC file and parsing it, or assigning as an attribute in the *PE* class of DS3 framework.

4.2 Job Generator

Figure 3 presents block diagrams for a WiFi transmitter (WiFi-TX) and receiver (WiFi-RX) both of which are composed of multiple tasks. The tasks and dependencies in an application are represented using a DAG. The job generator produces the tasks shown in Figure 3, for a WiFi-TX job along with the dependencies (see Appendix A for real DAG implementations). The basic unit of data processed by this chain is a frame, which is 64 bits in this work. DS3 defines each new input frame of an application as a job. Hence, each job is a 64-bit frame streaming through the WiFi-TX chain.

The job generator produces the tasks and DAG for each active application following a user-specified job injection model. DS3 currently models the traffic by injecting jobs based on an exponential distribution. The framework provides the flexibility to model other distributions as well. DS3 is scalable in terms of job generation and is capable of spawning jobs from multiple applications. For example, suppose that both WiFi-TX and WiFi-RX applications are active and the corresponding injection ratio is 0.8:0.2. On an average, DS3 generates 4 WiFi-TX jobs for every WiFi-RX job. This capability plays a crucial role in exploring mix of multiple workloads, as demonstrated in Section 7.

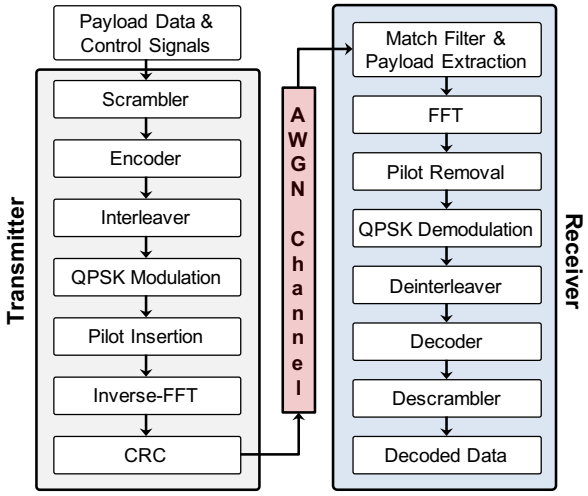


Fig. 3: Block diagrams for WiFi-TX and WiFi-RX applications.

4.3 Scheduling and DTPM Algorithms

DS3 provides a plug-and-play interface to choose between different scheduling and DTPM algorithms. Hence, developers can implement their own algorithms and easily integrate them with the framework. To achieve this, developers define the new scheduling algorithm as a member function of the *Scheduler* class. Then, the new scheduler is invoked from the *run* method of the simulation core. Scheduling algorithms vary significantly in their complexities and hence, require different inputs to map tasks to PEs. To support this, DS3 provides a loosely defined interface to specify inputs to the schedulers as required. The framework supports list schedulers (such as HEFT [34] as well as table-based schedulers (such as constraint programming), where schedule for all the tasks in a job is generated at the time of job injection.

DS3 provides built-in DTPM policies and facilitates the design of new DTPM algorithms. The policy is invoked periodically at every control epoch, which is parameterizable by the user. To minimize the run-time and power overhead of DTPM decisions, we use 10ms–100ms range following the common practice [32]. A policy of low complexity may use only the power state information of the PEs. On the other hand, advanced algorithms may use PE utilization and more detailed performance metrics, such as number of memory accesses and retired instructions. In addition, the DTPM policies have access to the resource management, including the power consumption and performance profiles therein. The decisions of the DTPM policy are evaluated and applied to the PEs at every control epoch.

Developers can add new scheduling and DTPM algorithms without modifying the rest of DS3. Modular design enables both maintaining existing interfaces and expanding them to support radically different algorithms.

4.4 Simulation Kernel

The life cycle of a task in DS3 is shown in Figure 4. The job generator constructs a task graph as described in Section 4.2. The tasks that are ready to execute (i.e.

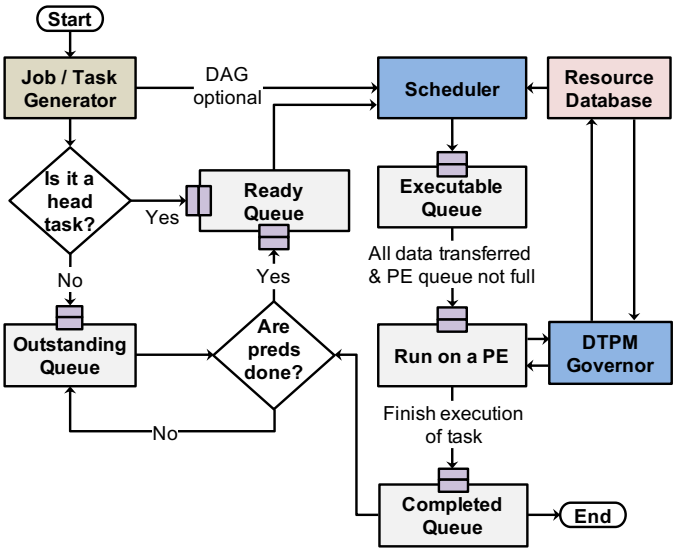


Fig. 4: Life-cycle of a task in DS3 queues.

free of dependencies) are moved to a *Ready Queue*. The other tasks that are waiting for predecessors to complete execution are held in the *Outstanding Queue* before being moved to the *Ready Queue*. The scheduler, an algorithm either built-in or user-defined, uses the resource database and produces PE assignments for ready tasks. Then, the simulation kernel migrates the tasks to the *Executable Queue* until communication requirements from predecessors are met. Finally, the task is simulated on the PE and retired after execution. The simulation kernel clears the dependencies imposed by these tasks and removes them from the system. If all the predecessors of a task waiting in the *Outstanding Queue* retire, the kernel moves them to the *Ready Queue*. This triggers a new scheduling decision and the tasks experience a similar life cycle in the framework, as described above.

Memory and network are shared resources in an SoC. The communication fabric performing high-speed data transfers among the various resources of the platform is assumed to be a mesh-based network-on-chip (NoC). We integrate analytical models to compute the latency at a given traffic load in a priority-aware mesh-based industrial NoC [31]. Executing multiple applications simultaneously leads to higher traffic in the network, as compared to the standalone execution. Hence, we account for the effect of a congestion in the network on execution time of applications. To model memory communication in the SoC, we include a bandwidth-latency model for memory latency modeling based on DRAMSim2 [35]. DRAMSim2 is used to obtain memory latencies at varying bandwidth requirements as shown in Figure 5. DS3 models the transactions between the various communicating elements and keeps track of outstanding memory requests in a sliding window. We compute the memory bandwidth based on outstanding requests and then utilize the bandwidth-latency curve as a look-up table to obtain the average latency for the current memory bandwidth and add it to the execution time of the application(s). Hence, we account for contention of shared resources using the described network and memory models.

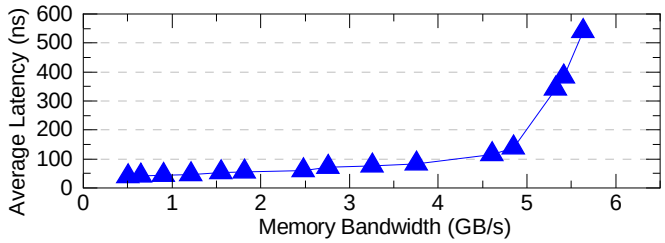


Fig. 5: Bandwidth-Latency curve used to model DRAM latency in DS3 framework.

The simulation kernel also calls the DTPM governor periodically at every decision epoch. The DTPM governor determines the power states of the PEs as a function of their current load and information provided by the resource database. Subsequently, the simulation kernel updates the power states of the PEs and the decisions are retained until the next evaluation at the next epoch.

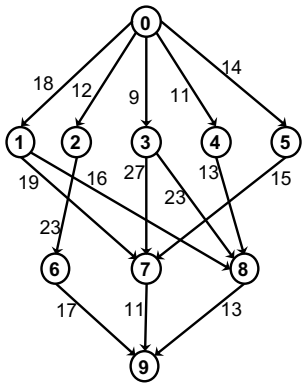
5 User View: DS3 Capabilities

This section presents the built-in scheduling and DTPM algorithms provided by the framework.

5.1 Scheduling Algorithms

DS3 provides a set of commonly used built-in scheduling algorithms, which can be specified by the users in the main configuration file. The framework generates Gantt charts to visualize the schedulers (see Figure 7). This allows the end-user to understand the dynamics of the scheduler under evaluation. We describe the built-in scheduling algorithms in DS3 using one of the most commonly used canonical task graphs [34] shown in Figure 6. Since this task graph is used commonly as reference for many list-scheduling studies, it serves as a representative example before analyzing the results from real-world applications in Section 7.3.

Minimum Execution Time (MET) Scheduler: The MET scheduler assigns a ready task to a PE that achieves the minimum expected execution time following a FIFO policy [36]. If there are multiple PEs that satisfy the minimum execution criterion, the scheduler then reads the current state information of all these PEs and assign the tasks to one of the most idle PEs. Figure 7(a) shows the schedule generated by the MET scheduler in DS3 for the DAG shown in Figure 6. All tasks are assigned to their best-performing PEs as expected.



Task	P0	P1	P2
0	14	16	9
1	13	19	18
2	11	13	19
3	13	8	17
4	12	13	10
5	13	16	9
6	7	15	11
7	5	11	14
8	18	12	29
9	21	7	16

Fig. 6: A canonical task flow graph [34] with 10 tasks. Each node represents a task and each edge represents average communication cost across the available pool of PEs for the pair of nodes sharing that edge. The computation cost table on the right indicates the execution time for each of the PEs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2986963, IEEE Transactions on Computers

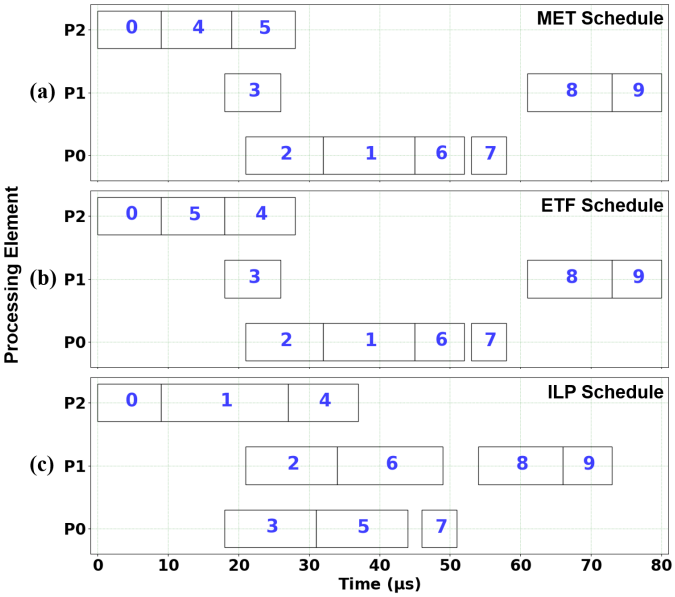


Fig. 7: Schedule of task graph in Figure 6 with (a) MET, (b) ETF, and (c) CP.

Earliest Task First (ETF) Scheduler: The ETF scheduler utilizes the information about the communication cost between tasks and the current status of all PEs to make a scheduling decision [37]. Figure 7(b) shows the schedule produced by DS3 for a single instance of DAG shown in Figure 6 with ETF scheduler. Although the execution times are the same for both MET and ETF based schedules, the mapping decisions are different as shown in Figure 7(a) and 7(b). This difference becomes evident when multiple applications are executed together, as shown in Section 7.3.

Table-based Scheduler: DS3 also provides a scheduler which stores the scheduling decisions in a look-up table. This allows users to utilize any offline schedule, such as an assignment generated by an constraint programming (CP) solver, with the help of a small look-up table. For example, we use IBM ILOG CPLEX Optimization Studio [38] to generate the CP solution for a job. The schedule from the CP solution depicted in Figure 7(c) outperforms the other two schedulers for a single job instance as expected. However, we note that the schedule stored in the table guarantees optimality only if there is a single job in the system. Hence, its performance degrades when multiple jobs overlap, as shown in Section 7.3.

5.2 DTPM Policies

State-of-the-art SoCs support multiple voltage-frequency domains and DVFS, which enables users to optimize for various power-performance trade-offs. To support this capability, DS3 allows each PE to have a range of OPPs, configurable in the resource database. The OPPs are voltage-frequency tuples that represent all supported frequencies of a given PE, which can be exploited by DTPM algorithms to tune the SoC at runtime. In addition, DS3 integrates analytical power dissipation and thermal models for Arm Cortex-A15 and Cortex-A7 cores [32], and power profiles for FFT [39], scrambler encoder, and Viterbi accelerators [40].

The power models capture both dynamic and static power consumption. The dynamic power consumption ($P = CV^2Af$) varies according to the load capacitance (C), supply voltage (V), activity factor (A), and operating frequency (f). Voltage and frequency are modeled through the OPPs, while the load capacitance and activity factors are modeled using measurements on real devices and published data. Static power consumption depends mainly on the current temperature and voltage, and DS3 uses thermal models obtained from measurements in the Odroid-XU3 SoC to accurately model both power and temperature.

The integrated power, performance, and temperature models enable us to implement a wide range of DTPM policies using DS3. To provide a solid baseline to the user, we also provide built-in DVFS policies that are commonly used in commercial SoCs. More specifically, users use the input configuration file to set the DTPM policy to *ondemand*, *performance* and *powersave* [18] governors, or to a custom DTPM governor.

Ondemand Governor: The *ondemand* governor controls the OPP of each PE as a function of its utilization. The supported voltage-frequency pairs of a given PE are given by the following set:

$$\mathcal{OPP} = \{(V_1, f_1), (V_2, f_2), \ldots, (V_k, f_k)\} \quad (1)$$

where k is the number of operating points supported by that PE. Suppose that the PE currently operates at (V_2, f_2). If the utilization of the PE is less than a user-defined threshold, then the *ondemand* governor decreases the frequency and voltage such that the new OPP becomes (V_1, f_1). If the utilization is greater than another user-defined threshold, the OPP is increased to the maximum frequency. Otherwise, the OPP stays at the current value, i.e., (V_2, f_2).

Performance Governor: This policy sets the frequency and voltage of all PEs to their *maximum* values to minimize execution time.

Powersave Governor: This policy sets the frequency and voltage of all PEs to their *minimum* values to minimize power consumption.

User-Specified Values: This policy enables users to set the OPP (i.e., frequency and voltage) of each PE individually to a constant value within the permitted range. It enables thorough power-performance exploration by sweeping the OPPs. Finally, developers can also define custom DTPM algorithms in the *DTPM* class, similar to the scheduler.

6 Simulator Validation

Simulation frameworks serve as powerful platforms to perform rapid design space exploration, evaluation of scheduling algorithms and DTPM techniques. However, the fidelity of such simulation frameworks is questionable. Particularly, the level of abstraction in high-level simulators is significant. Hence, estimations from simulations may diverge due to differences in modeling, ineffective representation and limitations to capture overheads observed on hardware platforms. In this section, we comprehensively evaluate our DS3 framework in terms of performance, power and temperature estimations with two commercial SoC platforms — Odroid-XU3 and Zynq Ultrascale+ ZCU102.

6.1 Validation with Odroid-XU3

We choose Odroid-XU3 as one of the platforms for validation because of its abilities to measure

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2986963, IEEE Transactions on Computers

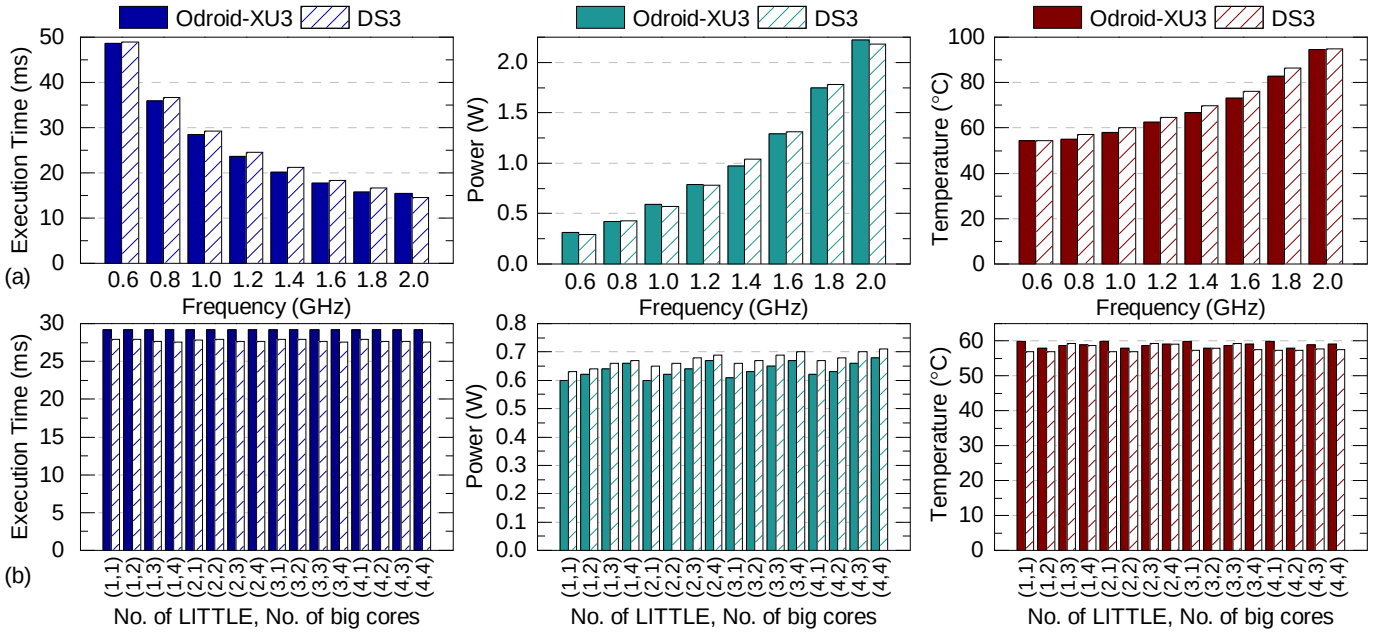


Fig. 8: Comparison of execution time, power and temperature between DS3 and Odroid-XU3 for single-threaded applications when (a) Freq-Sweep: Number of cores is constant, frequencies of the cores are varied (b) Core-Sweep: Frequencies of cores are constant, number of cores is varied.

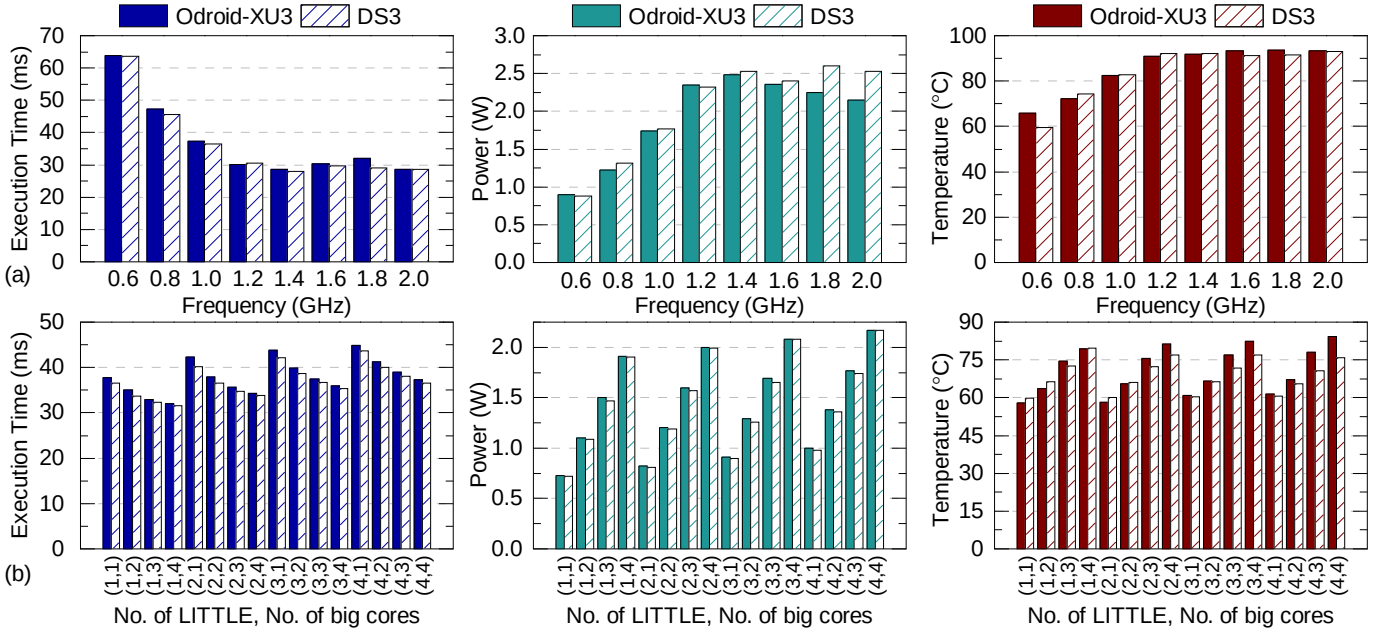


Fig. 9: Comparison of execution time, power and temperature between DS3 and Odroid-XU3 for multi-threaded applications when (a) Freq-Sweep: Number of cores is constant, frequencies of the cores are varied (b) Core-Sweep: Frequencies of cores are constant, number of cores is varied.

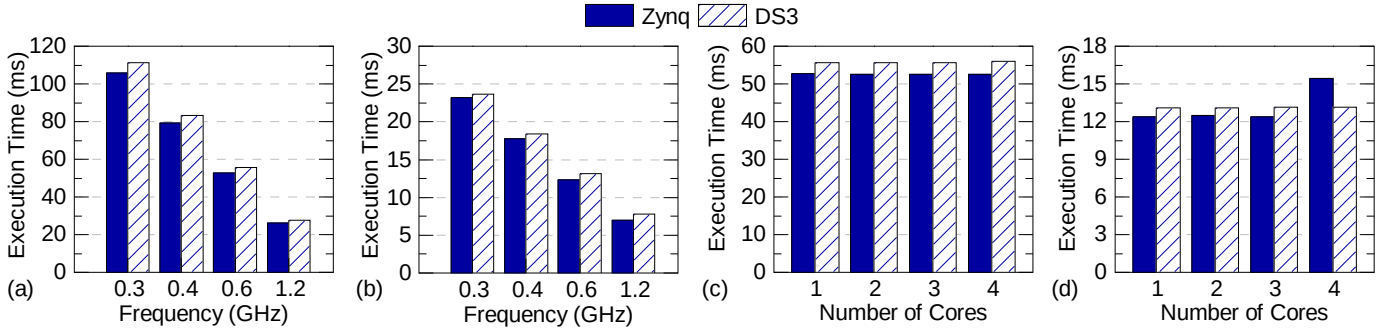


Fig. 10: Comparison of execution time between DS3 and Zynq MPSoC when (a) Freq-Sweep with only Cortex A53 cores, (b) Freq-Sweep with Cortex A53 cores and hardware accelerators (c) Core-Sweep with only Cortex A53 cores, (d) Core-Sweep with Cortex A53 cores and hardware accelerators.

TABLE 3: Error percentages in comparison of execution time, power and temperature between DS3 and Odroid-XU3

Application Type	Scenario	Execution Time	Power	Temperature
Single Threaded	Freq-Sweep	3.6%	3.1%	2.7%
	Core-Sweep	5.3%	5.1%	2.1%
Multi Threaded	Freq-Sweep	2.8%	6.1%	2.4%
	Core-Sweep	2.7%	1.3%	3.8%

power, performance and temperature. This platform comprises in-built current and temperature sensors enabling us to measure power, performance and temperature simultaneously and accurately at runtime. Since the design space comprising the number of cores, frequency levels and applications is very large, we choose representative configurations and applications for validation against the Odroid-XU3, as shown in Figure 8 and Figure 9. For a comprehensive validation, we consider two cases: (1) Freq-Sweep: the number of cores is fixed, frequencies of the cores are varied and (2) Core-Sweep: the frequencies of the cores are fixed, the number of cores is varied. To ensure completeness in validation, we validate both single-threaded and multi-threaded applications in both scenarios. For single-threaded applications, Figure 8(a) describes the execution time, power and temperature for Freq-Sweep scenario and Figure 8(b) for Core-Sweep. For Freq-Sweep, we vary the frequency from 0.6-2.0 GHz. Odroid-XU3 has four LITTLE cores and four big cores, leading to 16 possible combinations of configurations of active cores. Core-Sweep compares the parameters of DS3 and Odroid-XU3 for all 16 combinations. Similarly, Figures 9(a) and (b) show the comparisons of DS3 and Odroid-XU3 for multi-threaded applications. Table 3 shows the error percentages in comparison of execution time, power and temperature for single-threaded and multi-threaded applications in both Freq-Sweep and Core-Sweep configurations.

In Freq-Sweep scenario, the frequency of the LITTLE and big cores are varied in unison until we reach the maximum frequency of the LITTLE cores, which is 1.4 GHz. Thereafter, the frequency of the big cores are swept until the maximum frequency of 2.0 GHz. From Table 3, we observe that the average error in performance, power and temperature estimates are *3.6%*, *3.1%*, and *2.7%*, respectively, for single-threaded applications. Similarly, the errors are *2.8%*, *6.1%*, and *2.4%* for multi-threaded applications.

In Core-Sweep scenario, the number of active cores is varied in all combinations while the frequencies of the LITTLE and big cores are retained at 1.0 GHz. The performance, power and temperature mean absolute errors are *5.3%*, *5.1%*, and *2.1%*, respectively, for single-threaded applications while multi-threaded applications have an average error of *2.7%*, *1.3%*, and *3.8%*.

On an average, the error in accuracy is mostly less than 6%. We note that the platform experiences frequency throttling when the temperature reaches trip points (95°C). The throttling behavior is modeled in DS3 and hence, we obtain highly correlated estimates for execution time, power and temperature even when the platform is throttled by the on-board thermal management agent.

In summary, the estimates from DS3 closely match real-time measurements obtained by the execution of similar workloads on the platform, as summarized in Table 3. The strong validation results aid in reinforcing the fidelity of the framework in simulating DSSoCs with high accuracy.

6.2 Validation with Zynq Ultrascale+ ZCU102

The second platform used to validate the results of the DS3 framework is Zynq Ultrascale+ ZCU102 FPGA SoC. Zynq serves as a crucial platform for validation as it supports the implementation of hardware accelerators, unlike Odroid-XU3. The support for hardware accelerators aids in validating DS3 against highly heterogeneous SoCs. However, lack of on-board sensors prevent us from accurately measuring power and temperature. Hence, we chose to validate the execution time of Zynq and DS3 in the presence of hardware accelerators in various scenarios.

We pick multiple scenarios to validate the execution time. First, we sweep the frequencies across the four supported frequencies on the Zynq board, which is the Freq-Sweep scenario. We then measure the execution times when applications are executed only on Cortex A53 cores on the platform and then with both A53 cores and hardware accelerators. Secondly, we measure the execution times in Core-Sweep scenario with only A53 cores, and with A53 cores and hardware accelerators. Figure 10 shows high correlation between the measurements obtained from the Zynq board and DS3. However, when four A53 cores and hardware accelerators are enabled, we observe an anomaly with 15% error between DS3 and Zynq.

Current Linux kernels do not support scheduling and enablement of hardware accelerators in the operating system. Hence, we implement the scheduling mechanism for accelerators in user-space and identify the anomaly as an overhead incurred due to the user-space implementation. This overhead is expected to be significantly minimized when operating systems include support for accelerators. Finally, the average error in execution time is 6.85%.

7 Application Case Studies

This section presents case studies and experiments for design space exploration of dynamic resource management, power-thermal management, and architecture configurations. We base our studies on the benchmark applications, which are presented in the following section.

7.1 Benchmark Applications

DS3 comes with *six reference applications* from wireless communications and radar processing domain:

- **WiFi-TX/RX**,
- **Low-power single-carrier TX/RX**, and
- **Radar and Pulse Doppler**.

The WiFi protocol consists of transmitter and receiver flows as shown in Figure 3. It has compute-intensive blocks, such as FFT, modulation, demodulation, and Viterbi decoder (see Table 4), which require a significant amount of system resources. When the bandwidth and latency requirements are small, one can use a simpler single carrier protocol to achieve lower power consumption. Finally, we include two applications from the radar domain as part of the benchmark application suite - (1) range detection and (2) pulse Doppler (see Table 4). Figure 11(a) and Figure 11(b) represents block diagrams of the range detection and pulse Doppler applications, respectively.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2986963, IEEE Transactions on Computers

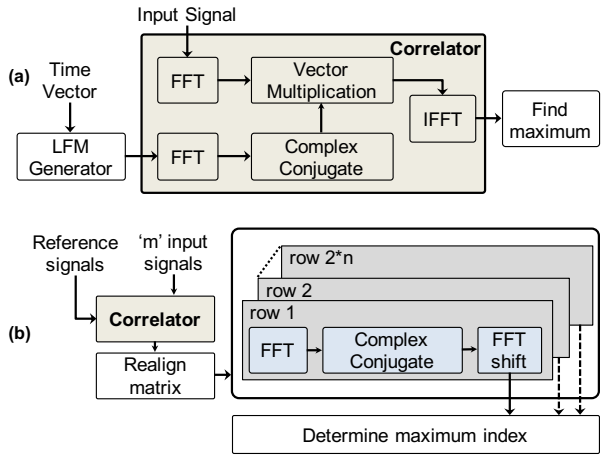


Fig. 11: Block diagram of (a) range detection application, (b) pulse Doppler application where m is number of signals and n is number of samples for a signal.

TABLE 4: Execution time profiles of applications on Arm A53 core in Xilinx ZCU102, Arm A7/A15 cores in Odroid-XU3, and hardware accelerators

Application	Task	Latency (μs)			
		Zynq A53	Odroid A7	Odroid A15	HW Acc.
WiFi TX	Scrambler-Encoder	22	22	10	8
	Interleaver	8	10	4	
	QPSK Modulation	15	15	8	
	Pilot Insertion	4	5	3	
	Inverse-FFT	225	296	118	16
	CRC	5	5	3	
WiFi RX	Match Filter	15	16	5	
	Payload Extraction	5	8	4	
	FFT	218	290	115	12
	Pilot Extraction	4	5	3	
	QPSK Demodulation	79	191	95	
	Deinterleaver	10	16	9	
	Decoder	1983	1828	738	2
	Descrambler	2	3	2	
Pulse Doppler	FFT	30	35	15	6
	Vector Multiplication	30	100	35	
	Inverse-FFT	30	35	15	6
	Amplitude Computation	25	70	40	
	FFT Shift	6	7	3	
Range Detection	LFM Waveform Generator	20	90	60	
	FFT	68	150	60	30
	Vector Multiplication	52	75	60	
	Inverse-FFT	68	150	60	30
	Detection	10	20	20	

The benchmark applications enable various algorithmic optimizations and realistic design space exploration studies, as we demonstrate in this paper. We will continuously include applications from other domains to the benchmarks.

7.2 Reference Design of Applications

We developed a reference design for each of the applications described in Section 7.1 on two popular commercial heterogeneous SoC platforms: Xilinx Zynq ZCU102 UltraScale MpSoC [15] and Odroid-XU3 [16] which has Samsung Exynos 5422 SoC. The nodes of the DAG, i.e., the tasks that constitute the target application, are scheduled on the processing elements. Since the tasks in the DAG determine the scheduling and acceleration granularity, they should be coarse enough to offset the overheads and produce benefits. In the scope of DS3, we implement hardware accelerators in the programmable logic (PL) of the Xilinx Zynq SoC. Depending on the size of data transfers required for the accelerator, we use either memory-mapped (AXI4-Lite) or streaming interfaces (AXI-Stream). To be specific, we use memory-mapped interfaces to communicate with scrambler-encoder accelerators and stream interfaces for the FFT accelerators. A direct memory access (DMA) unit facilitates data transfers between user-space mappable memory buffers and the accelerators using a streaming interface. In this regard, we profiled the computation and communication times in a Linux environment running on the Zynq SoC. The speedup of scrambler-encoder accelerator is 2.75× in comparison to the performance on Arm A53 of Zynq SoC. On the other hand, the speedup of the FFT accelerator is ∼19× when comparing the total latency (computation and communication) with that of Arm A53 on Zynq SoC. The streaming interface allows for significantly improved data transfer latencies. The transfer of data by the DMA unit through the user-space mappable memory buffers can further be improved by enabling caching and higher-bandwidth interfaces in the Zynq SoC. Hence, the Viterbi decoder accelerator is assumed to have a highly efficient data transfer mechanism and consequently the speedup is three orders of magnitude (see Table 4).

The latency of each task in every application on different resource types of these two platforms is profiled. At runtime, the schedulers decide to allocate the task to either hardware accelerators or general-purpose cores based on their corresponding function of communication- and computation-times along with other system parameters. In addition to latency profiling, we used the power consumption and temperature sensors on the Odroid-XU3 board for power consumption profiling. DS3 power/performance models used in the resource manager incorporate these performance profiles for each task-resource pair.

7.3 Scheduler Case Studies

This section provides an extension to our previous work in [41], using built-in DS3 schedulers and applications in the benchmark suite. For the simulations, we use a typical heterogeneous SoC with a total of 16 general-purpose cores and hardware accelerators: 4 big Arm Cortex-A15 cores, 4 LITTLE Arm Cortex-A7, and 2 scrambler, 4 FFT, and 2 Viterbi decoder accelerators.

We schedule and execute the WiFi TX/RX, range detection and pulse Doppler task flow graphs using DS3 and plot the average job execution time trend with respect to the job injection rate, as shown in Figure 12. We use the parameters p_{RX}, p_{TX}, p_{range}, and p_{pulse} representing the probabilities for the new job being WiFi-RX, WiFi-TX, range detection and pulse Doppler, respectively.

Figures 12(a) and (b) depict the results with WiFi applications for a download and upload intensive workload, independently. To understand the performance of scheduling algorithms, we analyze the average execution time at varying rates of job injection. MET uses a naive

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2986963, IEEE Transactions on Computers

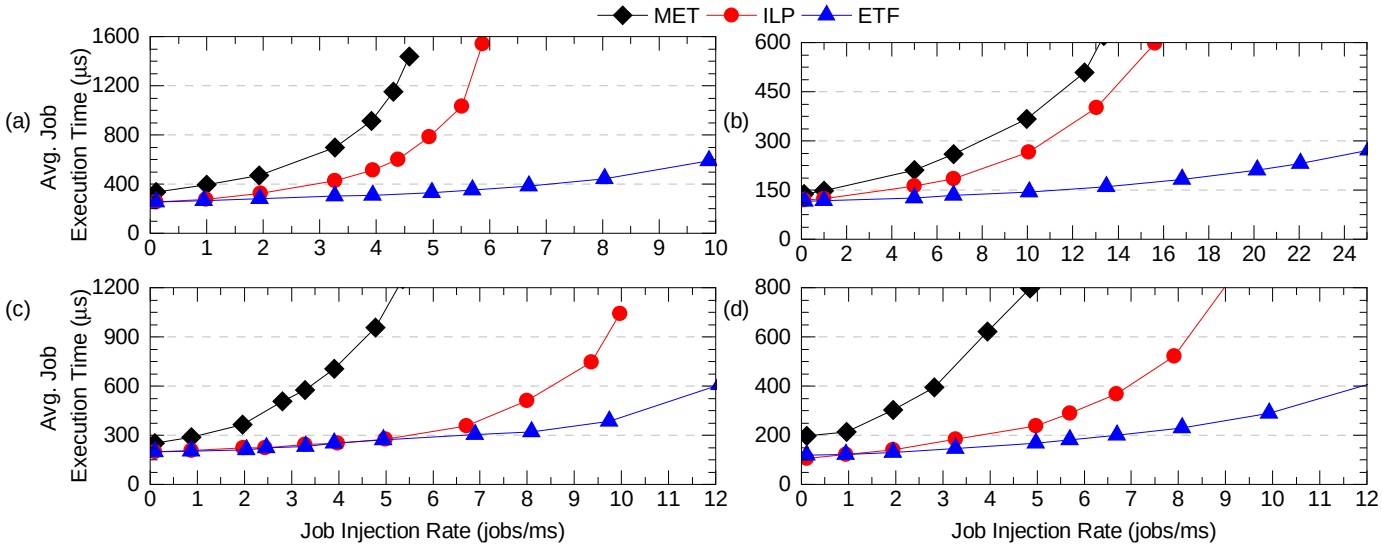


Fig. 12: Results from different schedulers with a workload consisting of (a) WiFi-TX (p_{TX}=0.2) and WiFi-RX (p_{RX}=0.8), (b) WiFi-TX (p_{TX}=0.8) and WiFi-RX (p_{RX}=0.2), (c) range detection (p_{range}=0.8) and pulse Doppler (p_{pulse}=0.2), (d) WiFi-TX (p_{TX}=0.3), WiFi-RX (p_{RX}=0.3), range detection (p_{range}=0.3), and pulse Doppler (p_{pulse}=0.1).

representation of the system state for scheduling decisions (described in Section 5.1), which results in higher execution time. On the other hand, CP uses a static table-based schedule which is optimal for one job instance. At low injection rates (less than 1 job/ms), CP is suitable as jobs do not interleave. However, as the injection rate increases, the CP schedule is not optimal. ETF scheduler is superior in comparison to the others, as observed in Figures 12(a),(b).

Figure 12(c) demonstrates the results for a workload comprising radar benchmarks. This workload uses p_{range} = 0.8 and p_{pulse} = 0.2, owing to the difference in execution times of the two applications. The performance of ETF and CP schedulers are similar until 5 jobs/ms, following which performance of ETF is superior in comparison to CP. Although the trend in execution time for radar benchmarks is similar to WiFi, the job injection rate at which ETF and CP diverge is different because of the differences in execution times of these applications, as shown in Table 5. At an injection rate lower than 5 jobs/ms, the level of interleaving of jobs is low which aligns with the CP solution.

Finally, we construct a workload comprising of all four applications and Figure 12(d) shows the corresponding results. The performance trend of the schedulers with all applications is similar to WiFi and radar workloads. MET considers only the best performing PEs for mapping and CP is sub-optimal at high injection rates whereas ETF utilizes the state information of all PEs for mapping decisions.

In summary, the experiments presented in Figure 12 demonstrate the capabilities of the simulation environment. DS3 allows the end user to evaluate workload scenarios exhaustively by sweeping the p_{TX}, p_{RX}, p_{range} and p_{pulse} configuration space to determine the scheduling algorithm that is most suitable for a given SoC architecture and set of workload scenarios.

TABLE 5: Execution time of applications in benchmark suite with different schedulers

	Execution Time of Single Job (μs)			
	WiFi-TX	WiFi-RX	Range Detection	Pulse Doppler
MET	69	389	177	1665
ETF	69	301	177	1045
CP	69	288	177	1000

7.4 SoC Design Space Exploration

This section illustrates how DS3 can be utilized to identify the number and types of PEs during early design space exploration. We employ the WiFi-TX and WiFi-RX applications to explore different SoC architectures. All configurations in this study have 4 big Arm Cortex-A15 and 4 LITTLE Arm Cortex-A7 cores to start with and DS3 guides the user to determine the number of configurable hardware accelerators in the architecture. We choose accelerators for FFT and Viterbi decoder. FFT is a widely used accelerator among all applications. We also choose Viterbi decoder because its execution cost on a general-purpose core is significantly high.

7.4.1 DSE Using Grid Search

We vary the number of instances of FFT (0, 1, 2, 4, 6) and Viterbi decoder (0, 1, 2, 3) in this grid search study. Table 6 lists the representative configurations out of 20 configurations. Each row in the table represents the configuration under investigation with an estimated SoC area, and average execution time and average energy consumption per job. The DS3 framework provides metrics that aid the user in choosing a configuration that best suits power, performance, area and energy targets.

Figure 13 plots the energy consumption per job as a function of the SoC area. We find the area of a given

TABLE 6: Area, performance and energy for different SoC configurations with varying number of accelerators

Configuration			Area (mm^2)	Average Job Execution (μs)	Energy per Job (μJ/job)
ID	FFT	Viterbi			
1	0	0	14.94	2606	1744
2	0	1	14.94	1824	1244
3	2	1	15.82	293	589
4	4	0	16.29	1212	957
5	4	1	16.56	274	584
6	6	3	19.29	264	582

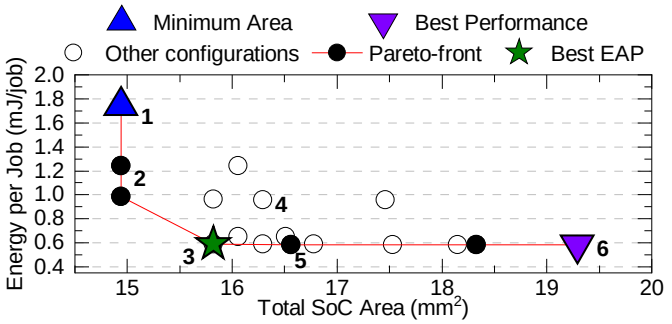


Fig. 13: Design space exploration studies showing energy per job vs. SoC area with pareto-frontier.

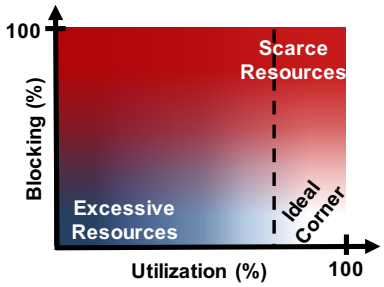


Fig. 14: PE blocking vs utilization (2-D performance plane)

configuration using a built-in floorplanner that takes the areas of PEs and other components such as system level cache and memory controllers. The energy consumption per job is computed as the ratio of total energy consumption for the entire workload with the number of completed jobs.

As the accelerator count increases in the system, the energy consumption per operation decreases. This comes at the cost of larger SoC area, as shown in Figure 13 and Table 6. For this workload, *configuration-3*, i.e., an SoC with two FFT and one Viterbi decoder accelerators, provides the best trade-off. Removing any of the accelerators leads to a significant increase in energy per operation with a small area advantage. In contrast, any further increase in the number of accelerators does not result in significant improvement in energy per job for this workload. As a result, *configuration-3* is the best configuration in terms of energy-area product (EAP). This configuration leads to an EAP gain of almost 65% (an energy reduction of 67% with an increase in area by only less than 6%) compared to *configuration-1*. After this point, the improvement in the overall performance by adding more FFT accelerators and Viterbi decoders does not overcome the cost of increase in the total area as seen in both Figure 13 and Table 6.

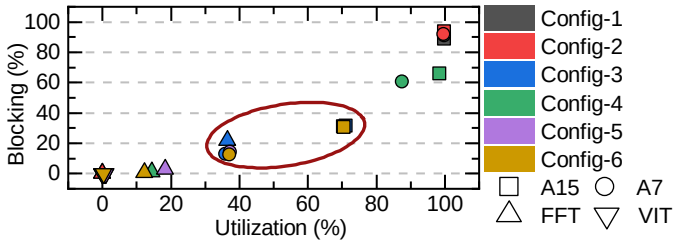


Fig. 16: Results for representative configurations on 2-D performance plane for PE clusters.

7.4.2 DSE Using Guided Search

DS3 also supports a guided search in the design space. Figure 14 depicts a 2-D performance plane for a PE (or a PE cluster) where *x*-axis and *y*-axis are utilization and blocking, both as percentages, respectively. Ideally, a PE should be on the lower-right corner where utilization is high, and blocking is low. If both utilization and blocking are high, upper-right corner, then it means that there is a need for more resources in the system. If, however, the opposite is true, the utilization of a PE is low, and it also does not block tasks very often. In this case, resources in the system are abundant. Finally, a PE should never be on the upper-left corner, low utilization and high blocking, which is unrealistic.

Considering the case study in Section 7.4.1 where we explore 20 different configurations, the guided search will converge on the best configuration faster. Figure 15 shows how utilization, blocking, and average job execution time differ for six representative configurations. *Configuration-1*, with no FFT and Viterbi accelerator, yields a high utilization and blocking for both Arm clusters, hence the SoC requires hardware accelerators. The results with *configuration-2* (addition of one Viterbi accelerator) indicate that the Viterbi accelerator is a critical component for the system since it provides a huge gain in average job execution time (a reduction from 2606 μs to 1824 μs) although the utilization for this accelerator is very small (0.61%). *Configuration-2* also suggests that one Viterbi accelerator is enough for the system since both utilization and blocking is low. Based on this observation, we directly eliminate configurations with no and more than one Viterbi accelerator (i.e., *configuration-4* and *-6*, see Table 6). The comparison between *configuration-3* and *-5* based on utilization, blocking, and estimated SoC area draws a conclusion that *configuration-3* is the best configuration for this case study.

Figure 16 depicts the same results for the representative configurations on the 2-D plane. As seen, *configuration-3* is closest to the ideal region and provides the best trade-off.

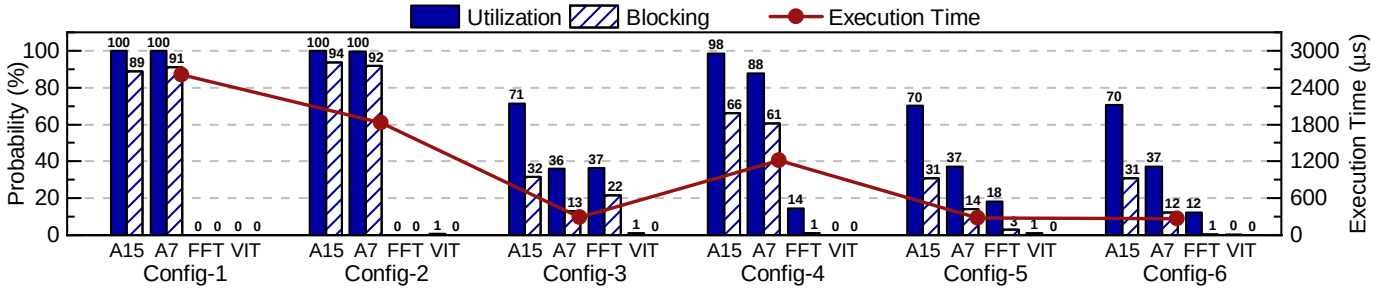


Fig. 15: Utilization vs blocking for PE clusters in representative configurations with average job execution times.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.2986963, IEEE Transactions on Computers

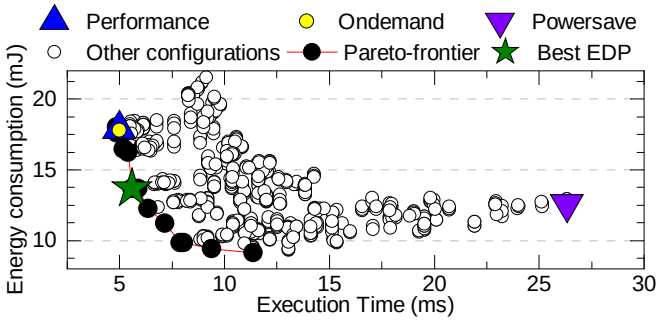


Fig. 17: Pareto frontier in the energy-performance curve for an SoC with 16 processing elements (PEs) executing a representative workload.

These approaches can be used to utilize DS3 for design space exploration of SoC architectures by analyzing energy efficiency, area, and developmental effort involved in the development of specialized cores for hardware acceleration. This approach can also be extended to explore the effect of modifying the number of general-purpose cores and hardware accelerators in a DSSoC architecture.

7.5 DTPM Design Space Exploration

In this section, we evaluate a subset of five applications from our benchmark set: WiFi-RX/TX, single-carrier RX/TX, and range detection on an SoC with 16 heterogeneous PEs. We explore 8 frequency points for the big cluster (0.6-2.0GHz) and 5 for the LITTLE (0.6-1.4GHz), using a 200MHz step. All possible DVFS modes were evaluated, i.e., all possible combinations of power states for each PE in addition to *ondemand*, *powersave*, and *performance* [18] modes.

Figure 17 presents the Pareto frontier for all aforementioned configurations. The *ondemand* and *performance* policies provide low latency with high energy consumption, while *powersave* minimizes the power at the cost of high latency, which results in sub-optimal energy consumption due to the increased execution time. The best configuration in terms of EDP uses 1.6GHz and 4 active cores for the big cluster, and 600MHz and 3 active cores for the LITTLE cluster, achieving 5.6ms and 13.7mJ. This figure also shows a $5\times$ variation in execution time and energy consumption between different configurations. The best EDP configuration achieves up to $4\times$ better EDP than default DTPM algorithms. This indicates that there is opportunity for users to propose their own power management mechanisms to improve the energy efficiency of the system and integrate those mechanisms into DS3.

In addition to the Pareto frontier, DS3 also provides energy, performance, and EDP histograms to aid the design space exploration. Due to space constraints, Figure 18 depicts only the EDP histogram. This histogram shows that only a small fraction of configurations achieve an EDP below 80mJ*ms. While the *performance* and *ondemand* governors are in the range of 90mJ*ms, the *powersave* mode gets 330mJ*ms EDP. Therefore, users can use these visualization tools to quickly identify the most promising configurations for the SoC.

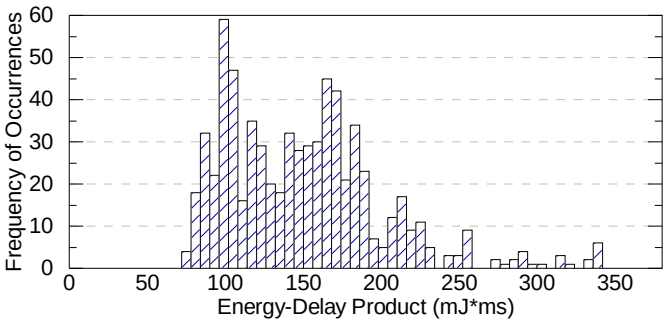


Fig. 18: Energy-Delay Product (EDP) histogram.

7.6 Scalability Analysis

This section illustrates the scalability of DS3 as a function of simulated number of jobs, SoC size, and the number of tasks in a single job (application). To maximize the load, we run four applications (WiFi TX/RX, range detection and pulse Doppler) simultaneously while sweeping the job injection rates and number of PEs. For the last case, however, we fix the job injection rates and SoC configuration while running applications different in size, separately.

Figure 19(a) shows the total simulation time as a function of the number of jobs injected throughout the simulation. As the relation between two metrics is linear, DS3 simulation run-time increases linearly with the workload size. Similarly, Figure 19(b) presents the simulation time when the number of PEs increase. This relationship is also linear, leading to 6ms per simulation cycle (1μs) for a 56-core configuration. Finally, in Figure 19(c), the simulation time with respect to application size (number of tasks in a single job) is depicted. As application size grows, the runtime to simulate of 1μs also increases linearly.

The scalability analysis provided in this section demonstrates that DS3 runtime is a linear function of workload, SoC and application size. As a side note, we obtain a simulation speedup of $600\times$ when running 1675 jobs of WiFi-TX in comparison to gem5. Hence, DS3 facilitates rapid design space exploration with relatively short turn-around times. This feature makes DS3 very powerful and helps users with extensive design space exploration in relatively short time while avoiding unnecessary simulation of low-level details.

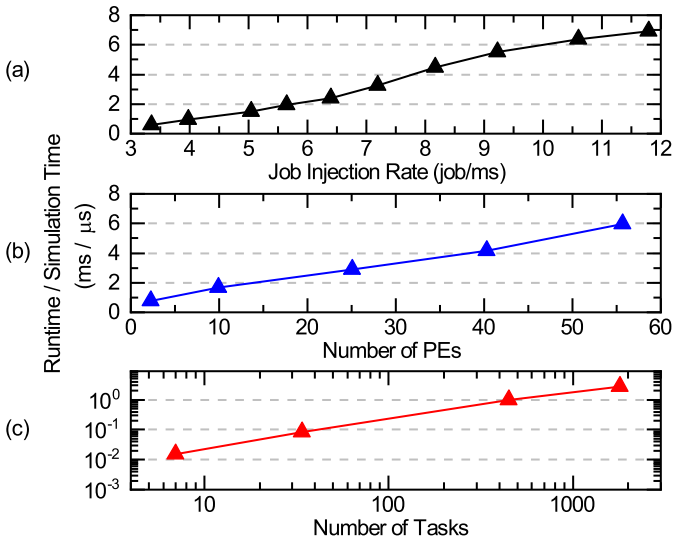


Fig. 19: Results for scalability analysis showing DS3 runtime versus (a) Different job injection rates, (b) Varying SoC configurations and (c) Number of tasks executed.

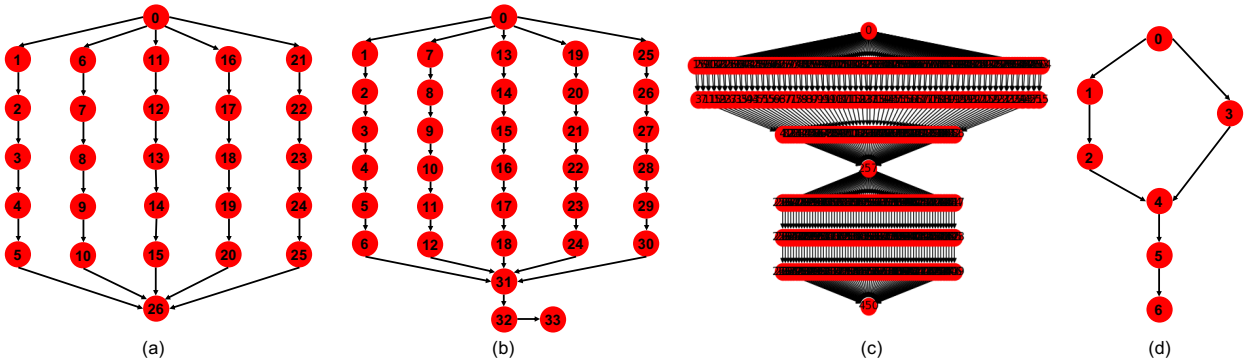


Fig. 20: DAG representations for (a) WiFi-TX, (b) WiFi-RX, (c) pulse Doppler, and (d) range detection application.

8 Conclusion and Future Work

The performance and energy efficiency potential of DSSoCs remains untapped unless we employ efficient run-time resource management techniques. This paper presented DS3, a Python-based open-source system-level framework for rapid design space exploration of domain specific SoCs. We developed a scalable, modular, and flexible simulation framework to evaluate scheduling algorithms, dynamic power-thermal management techniques and architecture exploration. We also presented benchmark applications, built-in scheduling algorithms and DVFS policies which can be used as reference by users and developers. Finally, the framework is thoroughly validated against a commercial SoC, asserting the fidelity of the simulator to successfully simulate domain-specific SoCs.

We shared DS3, built-in schedulers, DVFS governors, and benchmark applications to the public to encourage research in the domain of DSSoCs. The integration of learning-based schedulers and expanding the domain beyond wireless communications and radar systems are identified as part of future scope of this work. In the future, we envision extending DS3 to other domains with appropriate case studies and validation.

Appendix A
DAG Representations of Benchmarks

Applications from wireless communication and radar domains in the benchmark suite offer a variety of DAG representation ranging from simple one to very complex ones. Figure 20 shows DAG representations for WiFi TX/RX, pulse Doppler, and range detection applications. WiFi-TX and WiFi-RX are implemented as five parallel chains of tasks as seen in Figure 20(a),(b). In addition, range detection application only contains seven tasks whereas pulse Doppler is composed of 451 tasks since there are n samples for a single signal (see Figures 11(b), 20(d)).

References

[1] T. Chen *et al.*, "Diannao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning," in *ACM Sigplan Notices*, vol. 49, no. 4. ACM, 2014, pp. 269–284.

[2] O. Reiche *et al.*, "Generating FPGA-based Image Processing Accelerators with Hipacc," in *2017 IEEE/ACM Int. Conf. on Comput.-Aided Design (ICCAD)*. IEEE, 2017, pp. 1026–1033.

[3] A. R. Buzdar *et al.*, "Area and Energy Efficient Viterbi Accelerator for Embedded Processor Datapaths," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 3, pp. 402–407, 2017.

[4] J. L. Hennessy and D. A. Patterson, "A New Golden Age for Computer Architecture," *Commun. of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[5] J. Cong *et al.*, "Architecture Support for Domain-Specific Accelerator-Rich CMPs," *ACM Trans. on Embedded Computing Syst. (TECS)*, vol. 13, no. 4s, p. 131, 2014.

[6] R. Uhrie *et al.*, "Machine understanding of domain computation for domain-specific system-on-chips (dssoc)," in *Open Architecture/Open Business Model Net-Centric Systems and Defense Transformation 2018*, vol. 11015. Int. Society for Optics and Photonics, 2019.

[7] S. Liu *et al.*, "Computer Architectures for Autonomous Driving," *Computer*, vol. 50, no. 8, pp. 18–25, 2017.

[8] C.-L. Chou, U. Y. Ogras, and R. Marculescu, "Energy-and performance-aware incremental mapping for networks on chip with multiple voltage levels," *IEEE Trans. Comput.-Aided Des. Integr. Syst.*, vol. 27, no. 10, pp. 1866–1879, 2008.

[9] J. Castrillon *et al.*, "Communication-Aware Mapping of KPN Applications onto Heterogeneous MPSoCs," in *Proc. of the 49th Annu. Design Automat. Conf.* ACM, 2012, pp. 1266–1271.

[10] L. T. Smit, J. L. Hurink, and G. J. Smit, "Run-time Mapping of Applications to a Heterogeneous SoC," in *2005 Int. Symp. on System-on-Chip*. IEEE, 2005, pp. 78–81.

[11] E. L. de Souza Carvalho, N. L. V. Calazans, and F. G. Moraes, "Dynamic Task Mapping for MPSoCs," *IEEE Design & Test of Computers*, vol. 27, no. 5, pp. 26–35, 2010.

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

[13] A. Sandberg *et al.*, "Full Speed Shead: Detailed Architectural Simulation at Near-native Speed," in *2015 IEEE Intl. Symp. on Workload Characterization*, pp. 183–192.

[14] M. Pellauer *et al.*, "HAsim: FPGA-based High-detail Multicore Simulation Using Time-division Multiplexing," in *2011 IEEE 17th Int. Symp. on High Perf. Comput. Arch.* IEEE, 2011, pp. 406–417.

[15] "ZCU102 Evaluation Board," https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf, Accessed 3 Mar. 2019.

[16] "ODROID-XU3," https://wiki.odroid.com/old_product/odroid-xu3/odroid-xu3 Accessed 3 Mar. 2019.

[17] J.-G. Park *et al.*, "HiCAP: Hierarchical FSM-based Dynamic Integrated CPU-GPU Frequency Capping Governor for Energy-Efficient Mobile Gaming," in *Proc. of the 2016 Int. Symp. on Low Power Electronics and Design*. ACM, 2016, pp. 218–223.

[18] D. Brodowski *et al.*, "Linux CPUFreq Governors." [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[19] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Temperature Management in Multiprocessor SoCs Using Online Learning," in *2008 45th ACM/IEEE Design Automation Conf.* IEEE, pp. 890–893.

[20] A. K. Maurya and A. K. Tripathi, "On Benchmarking Task Scheduling Algorithms for Heterogeneous Computing Systems," *The J. of Supercomputing*, vol. 74, no. 7, pp. 3039–3070, 2018.

[21] N. Khalilzad, K. Rosvall, and I. Sander, "A modular design space exploration framework for multiprocessor real-time systems," in *2016 Forum on Specification and Design Languages (FDL)*. IEEE, 2016, pp. 1–7.

[22] K. Neubauer *et al.*, "Exact multi-objective design space exploration using aspmt," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 257–260.

[23] N. Trčka *et al.*, "Integrated model-driven design-space exploration for embedded systems," in *2011 Int. Conf. on Embedded Comput. Syst. Arch., Modeling and Simulation*. IEEE, 2011, pp. 339–346.

[24] B. Kienhuis *et al.*, "An approach for quantitative analysis of application-specific dataflow architectures," in *Proc. IEEE Int. Conf. on Application-Specific Systems, Architectures and Processors*. IEEE, 1997, pp. 338–349.

[25] T. Basten *et al.*, "Model-driven design-space exploration for embedded systems: The octopus toolset," in *Int. Symp. On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2010, pp. 90–105.

[26] A. D. Pimentel *et al.*, "Exploring embedded-systems architectures with artemis," *Computer*, vol. 34, no. 11, pp. 57–63, 2001.

[27] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. on Computers*, vol. 55, no. 2, pp. 99–112, 2006.

[28] G. Beltrame, L. Fossati, and D. Sciuto, "Resp: a nonintrusive transaction-level reflective mpsoc simulation platform for design space exploration," *IEEE Trans. Comput.-Aided Des. Integr. Syst.*, vol. 28, no. 12, pp. 1857–1869, 2009.

[29] C. Augonnet *et al.*, "Starpu: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[30] Y. Xiao, S. Nazarian, and P. Bogdan, "Self-Optimizing and Self-Programming Computing Systems: A Combined Compiler, Complex Networks, and Machine Learning Approach," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, pp. 1–12, 2019.

[31] S. K. Mandal *et al.*, "Analytical performance models for NoCs with multiple priority traffic classes," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, p. 52, 2019.

[32] G. Bhat *et al.*, "Algorithmic Optimization of Thermal and Power Management for Heterogeneous Mobile Platforms," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 3, pp. 544–557, 2018.

[33] "CFS Scheduler," https://www.kernel.org/doc/html/latest/scheduler/sched-design-CFS.html Accessed 10 Mar. 2020.

[34] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. on Parallel and Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar 2002.

[35] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE computer architecture letters*, vol. 10, no. 1, pp. 16–19, 2011.

[36] T. D. Braun *et al.*, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, Jun 2001.

[37] J. Blythe *et al.*, "Task Scheduling Strategies for Workflow-based Applications in Grids," in *Proc. of the IEEE Int. Symp. on Cluster Computing and the Grid*, vol. 2, 2005, pp. 759–767.

[38] "V12.8: User's Manual for CPLEX," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[39] X. Chen *et al.*, "A Variable-size FFT Hardware Accelerator based on Matrix Transposition," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 99, pp. 1–14, 2018.

[40] J. D. Tobola and J. E. Stine, "Low-Area Memoryless optimized Soft-Decision Viterbi Decoder with Dedicated Paralell Squaring Architecture," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE, pp. 203–207.

[41] S. E. Arda *et al.*, "A simulation framework for domain-specific system-on-chips: work-in-progress," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis Companion*, 2019, pp. 1–2.

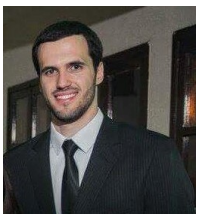
Samet E. Arda received his M.S. and Ph.D. degrees in Electrical Engineering from Arizona State University in 2013 and 2016. He is currently an Assistant Research Scientist at ASU. His Ph.D. thesis focused on development of dynamic models for small modular reactors. His current research interests include system-level design and optimization techniques for scheduling in heterogeneous SoCs.

Anish NK received his B.Tech degree in Electrical and Electronics Engineering from National Institute of Technology, Tiruchirappalli, India and Masters degree from Birla Institute of Technology, Pilani, India. Anish worked as a Physical Design Engineer at Qualcomm and as a Research Scientist at Intel Labs in India. He is currently pursuing Ph.D in Electrical Engineering at Arizona State University.

A. Alper Goksoy received his B.S. degree in Electrical and Electronics Engineering from Bogazici University, Istanbul, Turkey. He is currently pursuing his Ph.D. in Electrical Engineering at Arizona State University.

Nirmal Kumbhare Nirmal Kumbhare received his Ph.D. in computer engineering from the University of Arizona. His research interests involve reconfigurable and heterogeneous systems, high performance computing, and power-aware resource management. He has worked with Intel India prior to joining Ph.D.

Joshua Mack Joshua Mack is a doctoral student in the University of Arizona Electrical and Computer Engineering program. His research interests include the intersection of high performance computing and reconfigurable systems; emerging architectures; and intelligent and/or autonomous workload partitioning across heterogeneous systems.



Anderson L. Sartor received his B.Sc. and Ph.D. in Computer Engineering from Universidade Federal do Rio Grande do Sul (UFRGS), Brazil, in 2013 and 2018, respectively. He is currently a Postdoctoral Researcher at Carnegie Mellon University (CMU). His primary research interests include embedded systems design, machine learning for energy optimization, SoC modeling, and adaptive processors.

Ali Akoglu received his Ph.D. degree in Computer Science from the Arizona State University in 2005. He is an Associate Professor in the Department of Electrical and Computer Engineering and the BIO5 Institute at the University of Arizona. He is the site-director of the National Science Foundation Industry-University Cooperative Research Center on Cloud and Autonomic Computing. His research focus is on high performance computing and non-traditional computing architectures.

Radu Marculescu is the Laura Jennings Turner Chair in Engineering and Professor in the Electrical and Computer Engineering department at The University of Texas at Austin. He received his Ph.D. in Electrical Engineering from the University of Southern California in 1998. Radu's current research focuses on developing methods and tools for modeling and optimization of embedded systems, cyber-physical systems, social networks, and biological systems.

Umit Y. Ogras received his Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University, Pittsburgh, PA, in 2007. From 2008 to 2013, he worked as a research scientist at the Strategic CAD Laboratories, Intel Corporation. He is an Associate Professor at the School of Electrical, Computer and Energy Engineering, and the Associate Director of WISCA Center.