

System-Level Timing Performance Estimation Based on a Unifying HW/SW Performance Metric

Vittoriano Muttillo  

University of Teramo, Italy

Vincenzo Stoico  

Vrije Universiteit Amsterdam, The Netherlands

Giacomo Valente  



University of L'Aquila, Italy

Marco Santic  

University of L'Aquila, Italy

Luigi Pomante  

University of L'Aquila, Italy

Daniele Frigioni  

University of L'Aquila, Italy

Abstract

The rapidly increasing complexity of embedded systems and the critical impact of non-functional requirements demand the adoption of an appropriate system-level HW/SW co-design methodology. This methodology tries to satisfy all design requirements by simultaneously considering several alternative HW/SW implementations. In this context, early performance estimation approaches are crucial in reducing the design space, thereby minimizing design time and cost. To address the challenge of system-level performance estimation, this work presents and formalizes a novel approach based on a unifying HW/SW performance metric for early execution time estimation. The proposed approach estimates the execution time of a C function when executed by different HW/SW processor technologies. The approach is validated through an extensive experimental study, demonstrating its effectiveness and efficiency in terms of estimation error (i.e., lower than 10%) and estimation time (close to zero) when compared to existing methods in the literature.

2012 ACM Subject Classification Computer systems organization → Embedded systems; Computer systems organization → Embedded hardware

Keywords and phrases embedded systems, hw/sw co-design, performance estimation, lasso, machine learning

Digital Object Identifier 10.4230/OASICS.PARMA-DITAM.2025.3

Supplementary Material *Software (Source Code)*: <https://github.com/hepsycode/SLIDE-x> [22] archived at `swh:1:rev:0907cf39f7d023d0dc2e8307e1ffef6115d0377a`

Funding This research work has been funded by the Electronic Components and Systems for European Leadership Joint Undertaking (ECSEL JU) through the project AIDOaRt, grant agreement No. 101007350, and the Key Digital Technologies Joint Undertaking (KDT JU) through the project MATISSE, grant agreement No. 101140216.

1 Introduction

In the last thirty years, there has been an exponential increase in the exploitation of embedded systems in everyday life. This increase has led to a rise in the complexity of such embedded systems due to: (i) the continuous demand for additional improvements in both functional and non-functional requirements and (ii) the growing design automation with the application of embedded systems in various domains (e.g., Automotive, Aerospace) [11, 25].



© Vittoriano Muttillo, Vincenzo Stoico, Giacomo Valente, Marco Santic, Luigi Pomante, and Daniele Frigioni;

licensed under Creative Commons License CC-BY 4.0

16th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and 14th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2025).

Editors: Daniele Cattaneo, Maria Fazio, Leonidas Kosmidis, and Gabriele Morabito; Article No. 3; pp. 3:1–3:14

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Therefore, designing these systems is even more a critical task and so early-stage HW/SW performance estimation for rapid design space exploration at higher abstraction levels becomes crucial [10, 29].

In this context, numerous studies have explored the use of Machine Learning (ML) techniques for performance estimation [2, 14, 15, 17, 28, 34]. The adoption of these techniques has been driven by the challenges associated with creating an accurate analytical model of the HW/SW micro-architecture, which is often error-prone or sometimes impossible due to the lack of detailed documentation and necessary human expertise for model design [2]. Despite this scenario, the current state of the art lacks, to the best of our knowledge, of a unified HW/SW model capable of facilitating rapid performance estimation across several platforms at the system level.

For the above reasons, this study investigates how to overcome the limitations of the existing methods, particularly those restricted to specific application domains or technologies, through an approach that allows performance estimation of different HW/SW designs at the system level of abstraction. The provided approach uses the LASSO model to estimate the CC4CS performance metric presented and validated in [26]. CC4CS is a statement-level metric that can be used to quantify and, therefore, compare the performance of different processor technologies (i.e., Commercial Off-the-Shelf – COTS 8/32-bit embedded processors and HW components synthesized on FPGAs). CC4CS is defined as the ratio between the clock cycles and statements executed by a C function. In addition, this study presents a preliminary evaluation of the accuracy of the LASSO model in estimating CC4CS values. The evaluation of the accuracy of LASSO is carried out by using the **SLIDE-x**¹ framework, which executes a benchmark of well-known C functions across a set of 3 processors, namely Intel 8051, Atmega328p, Leon3, and an FPGA, i.e., the Artix7. Consequently, SLIDE-x outputs the CC4CS values that are used to train the LASSO model. Finally, the accuracy of LASSO is evaluated by comparing its predictions against the measurement profiled using SLIDE-x. The results are promising as they show a Mean Absolute Percentage Error (MAPE) of less than 10% for the Intel 8051, Leon3, and Artix7, with a maximum speed-up of up to 32x compared to the traditional HLS flow. In summary, our paper offers the following contributions: (1) formal HW/SW processor characterization through statistical analysis; (2) a detailed regression-based approach for evaluating HW/SW design performance; (3) a preliminary assessment of the accuracy of our performance predictions. This work is useful for system designers, helping them evaluate multiple HW/SW solutions and reduce design space exploration overhead.

2 Related works

In this section, we review the current state of research focused on two key areas within embedded systems design: predicting the timing performance of processors built to execute a given *Instruction Set Architecture (ISA)* (i.e., General Purpose Processors – GPPs, called SW processors), and of processors designed to directly execute application functions (i.e., Single Purpose Processors - SPPs, called HW processors) at the system level of abstraction.

To describe a SW processor and its behavior, several levels of abstraction can be considered. Accordingly, several timing estimations can be performed [27]. In such a context, the authors in [14] use a linear regression technique based on an application analysis performed at the Register Transfer Level (RTL) internal representation of the GNU GCC compiler (i.e.,

¹ SLIDE-x repository: <https://github.com/hepsycode/SLIDE-x>

needs for micro-architectural knowledge of the system). Zhang et Al. [34] use a linear regression model to estimate the performance of a given embedded software executed by the RISC-V processor, using metrics related to (assembly) instruction level. The final speed-up in comparison to the cycle-accurate simulation is up to 5x for RV32I and 4.2x for RV32IM. Finally, Amalou et Al. [2] present several approaches: (1) Ithemal: a tool that uses a Recurrent Neural Network (RNN) architecture with a hierarchical Long Short-Term Memory (LSTM) approach to predict the throughput of a set of instructions considering the opcodes and operands of instructions in a basic block (BB); (2) CATREEN: an RNN predictive algorithm able to predict the steady-state execution time of BBs in a program; (3) ORXESTRA: a tool that predicts the execution time of BBs within compiled binaries using a ML technique named Transformers XL, a recurrent variant of Transformers.

In the HW domain, the use of *High-Level Synthesis* (HLS) tools has become of vital importance [3,16]. HLS tools provide automatic transformation of C/C++/SystemC specifications into *Hardware Description Languages* (HDL) like Verilog or VHDL, significantly boosting productivity in custom hardware development²³. However, for large-scale systems, the time needed to perform HLS can often become a bottleneck [27]. Additionally, fast platform selection remains a significant challenge for developers due to the significant performance variations among platforms for the same workload [8].

To address this issues, Makrani et al. [15] introduced the Cross-Platform Performance Estimation (XPPE) tool based on ML. XPPE uses the resource usage reported by the Xilinx HLS tool and predicts application acceleration on various platforms using a Neural Network (NN) model, considering both application characteristics and FPGA platform parameters. The authors of [28] propose HLSPredict, an ML-based cross-platform estimator. Unlike XPPE, HLSPredict uses workloads as inputs to estimate performance on an FPGA by executing them on a Commercial Off-The-Shelf (COTS) host CPU. Finally, [17] presents Pyramid, a tool that uses ML to estimate optimal performance and resource usage of HLS designs, with the Random Forest (RF) outperforming other ML models.

2.1 State-of-the-Art Limitations

Table 1 compares our work with state-of-the-art ML studies by examining prediction errors. While existing studies focus on reducing errors through SW implementation or HW synthesis, none offer a unified HW/SW model with low prediction times and errors. This gap highlights the need for a unified model to compare HW and SW processors at the system level. Our paper aims to address these limitations.

3 Preliminaries

Our work introduces a method for system-level execution time estimation of C functions across different HW/SW processor technologies, using the CC4CS performance metric [26]. This metric, already used in literature for HW/SW Co-Design methodologies [21,24], simplifies performance estimation and comparison by abstracting the execution of “generic C statements”. Our approach is built on the model in [5], which defines a “generic C statement” as a combination of fundamental units, called “atoms”. Atoms are the basic components of statements, and the complexity of a statement depends on the number of atoms it con-

² Panda/Bambu Project: <https://panda.dei.polimi.it/>

³ Vitis HLS: <https://www.xilinx.com>

■ **Table 1** Comparison of literature Timing Estimation works. L/NL:= Linear/Non-Linear, LR:= Linear Regression.

Work	Target	Approach	Error (%)
[14]	ARM926EJ-S (SW) LEON3 (SW)	LR	8.25% ≤ ... ≤ 15.45% 8.03% ≤ ... ≤ 13.60%
[34]	RV32I (SW) RV32IM (SW)	LR	7.03% 5.27%
[2]	ARM Cortex M4, M7, A53, A72 (SW)	ITHEMAL CATREEN ORXESTRA	9.1% ≤ ... ≤ 18.2% 8.9% ≤ ... ≤ 13.4% 6.2% ≤ ... ≤ 8.9%
[28]	Artix7 (HW)	L/NL ML	1.88% ≤ ... ≤ 9.79%
[15]	20 Xilinx FPGA (HW)	NN	5.1% ≤ ... ≤ 9%
[17]	3 Xilinx FPGA (HW)	L/NL ML	3.5% ≤ ... ≤ 4.8%
Our Work	8051 (SW - CISC),	System-Level	4.26% ≤ ... ≤ 6.53%
	ATmega (SW - RISC),	Linear ML	6.99% ≤ ... ≤ 22.04%
	LEON3 (SW - RISC))	(Unified HW/SW	0.34% ≤ ... ≤ 2.58%
	Bambu (HW - Artix7)	Approach)	6.54% ≤ ... ≤ 11.84%

tains. Although the complexity of a C statement is not strictly predefined [26], factors like programmer experience, coding style, and standards [13] usually keep it at a “low/medium average complexity”. A “generic C statement” reflects the common way programmers write statements. When a C function runs with input data set D_k , each atom and statement executes a certain number of times, enabling the collection of profiling data, such as through tools like Gcov.

3.1 Performance Model for SW Processors

This subsection introduces a general mathematical model representing the execution time of a C function executed by a basic GPP (i.e., no advanced microarchitecture features, such as pipeline), refining the model proposed in [5]. To perform timing performance estimation, a model for the approximate (ideal) execution time \bar{T}_k is required. Therefore, in a basic GPP, the ideal execution time of a generic C function is:

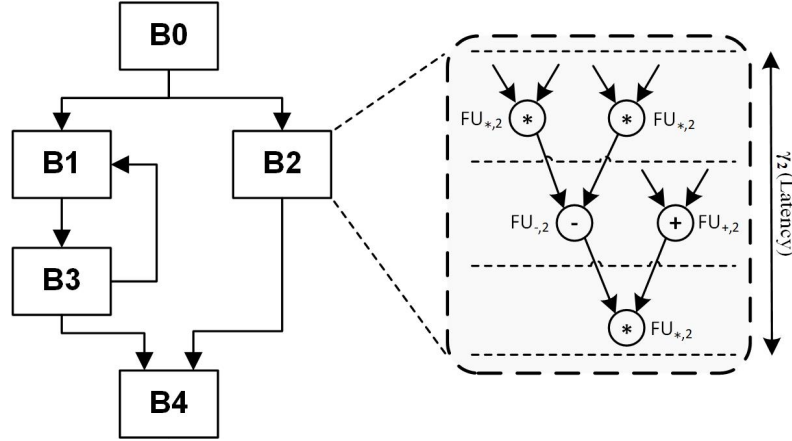
$$\bar{T}_k^{SW} = N_k^I \cdot CC_j \cdot \tau_j \quad (1)$$

where CC_j is the average number of clock cycles per statement, τ_j is the GPP processor’s clock period, and N_k^I represents the number of executions of all assembly instructions in the generic C function when run with input data set D_k .

3.2 Performance model for HW Processors

HW implementation (i.e., SPP) of a generic C function can be done using HLS tools like Bambu², LegUp [6], or Vitis HLS³. As noted in [27], common HLS practices use an intermediate representation to capture the control and data flows of the C code. Basic Blocks (BBs) represent the code control flow at the statement level.

The visual representation of control and data flow using BBs is called Control and Data Flow Graph (CDFG). Each operation is assigned to a Functional Unit (FU) capable of executing it [1], and FUs are encapsulated within the BBs of the CDFG model, as shown in Figure 1. Each datapath within the q -th BB contains $L_{j,q}^{FU}$ functional units of type $FU_{j,q,v}$



■ **Figure 1** CDFG representation. $FU_{*,2}$ is one of the FU of basic block B2, while γ_2 is the total latency of basic block B2.

(e.g., sum, mul, sub). The total propagation time (latency) depends on Integrated Circuit (IC) technologies and micro-architecture. $L_{j,q}^{FU}$ is obtained from code analysis on non-scheduled DFGs, while delays from registers and multiplexers are ignored. The actual execution time of a generic C function synthesized as an SPP is defined as:

$$T_k^{HW} = \sum_{q=1}^{N_k^{BB}} \sum_{h=1}^{N_{j,q,k}^{st}} \frac{1}{N_{j,q,k}^{st}} \sum_{v=1}^{L_{j,q}^{FU}} \omega_{j,q,v} \cdot \gamma_{j,q,v,k} \cdot \tau_j$$

where $\omega_{j,q,v} = \begin{cases} 1 & \text{if } FU_{j,q,v} \in \text{longest data path} \\ 0 & \text{otherwise} \end{cases}$ (2)

where T_k^{HW} in Eq. 2 is the execution time of the q-th BB in a generic C function synthesized as SPP p_j with input data set D_k . N_k^{BB} is the total number of executed BBs, $N_{j,q,k}^{st}$ is the number of executed statements in the q-th BB, and $\gamma_{j,q,v,k}$ is the latency of the v-th FU in the q-th BB. The clock period τ_j depends on IC technology, micro-architecture, and scheduling policy. Assuming no multi-cycling, pipelining, or chaining, τ_j is given by the following equation.

$$\tau_j = \max_{\forall \{v,q\}} t(FU_{j,q,v})$$

An average slack time (i.e., idle time of operations in a control step) can be used in multi-cycling and pipelined implementations [27] to calculate τ_j . HLS tools allow setting a desired clock period τ_j (e.g., Bambu²), and aim to minimize the difference between the desired τ_j and the actual τ_j . From Eq. 2, the simplified execution time model for a generic C function synthesized as SPP can be approximated as follows:

$$\bar{T}_{k,h}^{HW} = \sum_{h=1}^{L^{st}} \bar{T}_{k,h}^{HW} = 1/N_k^{st} \cdot \sum_{q=1}^{N_k^{BB}} \sum_{v=1}^{L_{j,q}^{FU}} \omega_{j,q,v} \cdot \gamma_{j,q,v} \cdot \tau_j^* \quad (3)$$

where $\bar{T}_{k,h}^{HW}$ is the average execution time of the h-th statement belonging to the q-th BB in a generic C function synthesized as SPP p_j with data D_k under the desired clock period τ_j^* .

3.3 Proposed Unified HW/SW Performance Model

The proposed unified HW/SW performance model integrates the timing behavior of a generic C function executed on a GPP with that of the same function synthesized as an SPP, achieved through HLS tools.

For the SW side, let $N_{h,k,j}^I$ represent the number of assembly instructions needed to execute statement h of the C function on GPP p_j with input data set D_k . This can be determined using an assembly-level execution trace [20]. The average number of executed assembly instructions \bar{N}_k^I is:

$$\bar{N}_k^I = 1/N_k^{st} \cdot \sum_{h=1}^{L^{st}} N_{h,k,j}^I \quad \text{and} \quad N_k^I = \sum_{h=1}^{L^{st}} N_{h,k,j}^I = \bar{N}_k^I \cdot N_k^{st}$$

N_k^I is the total number of executed assembly instructions, and N_k^{st} is the total number of executed statements for a generic C function with D_k . Therefore, Eq. 1 can be redefined:

$$\bar{T}_k^{SW} = \bar{N}_k^I \cdot N_k^{st} \cdot CC_j \cdot \tau_j = N_k^{st} \cdot \frac{\sum_{h=1}^{L^{st}} N_{h,k,j}^I \cdot CC_j}{N_k^{st}} \cdot \tau_j \quad (4)$$

According to Eq. 4, the expressions for the approximate (ideal) \bar{T}_k can be redefined for basic GPP processors as follows:

$$\bar{T}_k^{SW} = N_k^{st} \cdot \bar{t}(ST_k)^{SW} \quad (5)$$

$$\bar{t}(ST_k)^{SW} = 1/N_k^{st} \cdot \sum_{h=1}^{L^{st}} N_{h,k,j}^I \cdot CC_j \cdot \tau_j \quad (6)$$

$$CC_{j,k}^{SW} = \sum_{h=1}^{L^{st}} N_{h,k,j}^I \cdot CC_j \quad (7)$$

Eq. 7 shows the total clock cycles $CC_{j,k}^{SW}$ needed to execute a generic C function on GPP p_j with input D_k . This value is normalized in Eq.6 to the total number of C statements executed with input D_k .

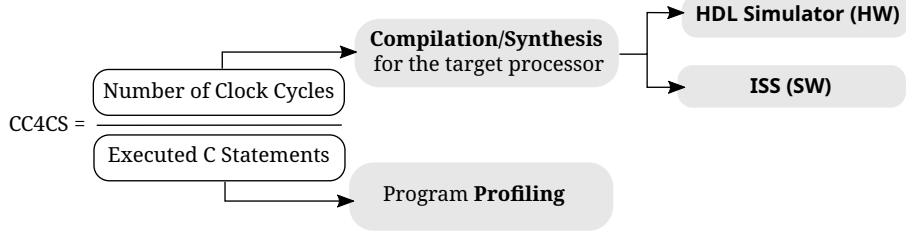
For the HW side, based on Eq. 3, the number of clock cycles $CC_{j,k}^{HW}$ needed to execute a generic C function synthesized as an SPP can be evaluated using HLS tools². These tools generate HDL files (Verilog or VHDL) and provide the required clock cycles for executing C functions with data D_k , as follows:

$$CC_{j,k}^{HW} = \sum_{q=1}^{N_k^{BB}} \sum_{v=1}^{L_{j,q}^{FU}} \omega_{j,q,v} \cdot \gamma_{j,q,v} \quad (8)$$

According to Eq. 5 and Eq. 8, a generic C function with input data D_k executed by a GPP or by an SPP requires an execution time of:

$$\bar{T}_k^{HW} = N_k^{st} \cdot \bar{t}(ST_k)^{HW} = N_k^{st} \cdot \left(\frac{CC_{j,k}^{HW}}{N_k^{st}} \right) \cdot \tau_j \quad (9)$$

The fraction within parentheses in Eq. 9 represents the unified metric *Clock Cycles for C statements* (CC4CS) [26]. The CC4CS metric, at the statement level of abstraction, encompasses both atoms (SW) and blocks (HW). According to Eq. 9, the empirical evaluation of the CC4CS metric across a set of HW/SW processor technologies requires a well-defined methodology for automated and repeatable operations, as shown in Figure 2.



■ **Figure 2** CC4CS Evaluation Methodology.

For a specific processor p_j , each C function is taken from the benchmark. Random input data $D_{s,k}$ is generated and uniformly distributed within a set range. Two parallel processes then determine the clock cycles required by the target HW or SW processor to execute the function ($CC_{j,k}^{\frac{HW}{SW}}$) and the number of C statements executed (N_k^{st}), which depends only on the input data and function, not on the processor. To evaluate CC4CS across HW/SW processors, the process involves: (a) selecting target processors p_j ; (b) choosing benchmark C functions; (c) generating input data sets D_k ; (d) profiling C functions to find the number of executed statements N_k^{st} (using tools like Gcov); (e) compiling/synthesizing C functions for each processor; (f) performing cycle-accurate simulations to extract the real execution time $\bar{t}(ST_k)$. This is done through ISS or HDL simulations. Each processor p_j will then have a Cumulative Distribution Function (CDF) of $CC4CS_j$. Different compiler optimization flags can be applied, though in this work, the -O0 flag is used as proof of concept, leaving other flags for future exploration.

3.4 Performance Estimation Approach

This work addresses the challenge of determining an estimator, $\hat{T}_{s,k}$, for the actual (real) execution time of a given C function z_s implemented or synthesized through both HW/SW processor technologies. The total actual (real) estimation time of a generic C function is expressed as follows:

$$T_k = \sum_{h=1}^{L^{st}} N_{h,k}^{st} \cdot t(ST_{h,k}) \simeq f(N_k^{st}, \hat{t}(ST_k)) = \hat{T}_k \quad (10)$$

where $\hat{t}(ST_k)$ is the estimated average time to execute a statement ST_k in a generic C function. The error to be minimized over functions and input data sets is:

$$\min_{\forall \{D_k\}} \epsilon_k^2 = \min_{\forall \{D_k\}} (T_k - \hat{T}_k)^2 \quad (11)$$

According to Eq. 10, our proposed solution uses the Least Absolute Shrinkage and Selection Operator (LASSO) [7] to exploit an approach called the CC4CS LASSO Regression Approach (CLRA), as follows:

$$\begin{aligned} \hat{T}_k &= \beta_0 + N_k^{st} \cdot \hat{t}(ST_h) \cdot \hat{\beta} + \lambda \cdot |\hat{\beta}| \\ \hat{t}(ST_h, z_s) &= \bar{t}(ST_k) + \delta \\ \bar{t}(ST_k) &= g(CC4CS_j, \tau_j) \text{ AND } \delta = h(\theta, CC4CS_j, \tau_j) \end{aligned} \quad (12)$$

where θ depends on correction functions like, e.g., the affinity value defined in [4]. LASSO regression performs an L1 regularization that adds a penalty equal to the absolute value of the magnitude of the coefficients. LASSO solutions are quadratic programming problems best solved with dedicated software tools (e.g., Matlab). According to Eq. 11 and Eq. 12, we define the final estimation problem as follows:

$$\min_{\beta_0, \hat{\beta}} 1/2d \sum_{k=1}^d [T_k - \beta_0 - N_k^{st} \cdot \hat{t}(ST_h) \cdot \hat{\beta}]^2 + \lambda \cdot |\hat{\beta}| \quad (13)$$

where d is the number of observations (i.e., number of C function executions), $T_{s,k}$ is the execution time with input data set D_k , N_k^{st} is the number of executed C statements with input data set D_k , λ is a non-negative regularization parameter. The parameters β_0 and $\hat{\beta}$ are scalar values.

4 Experimental Activities

This section outlines the experimental activities used to validate the proposed processor characterization and performance estimation approach. Based on Eq. 9 and Figure 2, we developed the **SLIDE-x** (System-Level Infrastructure for HW/SW Dataset **E**-xtraction) framework to evaluate CC4CS across various processors. While implementation details are beyond the scope of this paper, the source code is freely available on GitHub¹. All experiments were performed on a PC with an Intel® Xeon CPU E3-1225 v5 @ 3.30 GHz, 32 GB memory, and 128KB L1, 1 MB L2, and 8 MB L3 caches.

The benchmark includes 15 control- and data-dominated C functions from well-established HW/SW benchmarks [27]. Each function was tested with various data types (namely: *int8*, *int16*, *int32*, *int64* from `stdint` library, single precision IEEE 754 floating point data types) and randomly generated input files. A total of $6 * 10^4$ inputs were generated via uniform random distribution, with additional tests using $6 * 10^5$ and $6 * 10^6$ inputs showing no significant difference. The benchmark avoids function calls, recursion, external files, or library routines, and input ranges were set to prevent overflows.

CC4CS was evaluated for specific HW/SW processor technologies. For GPPs, we considered: (1) Intel 8051 CISC microcontroller⁴; (2) Microchip ATmega328/P⁵, a low-power CMOS 8-bit microcontroller; and (3) LEON3⁶, a 32-bit SPARC V8-compatible soft processor. The 8051 was simulated using Dalton ISS⁴, ATmega328/P with SimulAVR ISS⁷, and LEON3 with Cobham Gaisler TSIM ISS⁶. For SPPs, FPGA synthesis for the Xilinx Artix7 XC7A35T-1CPG236C was done using Bambu HLS².

4.1 Processor Characterization Results

In our work, we aimed to identify which classical probability distribution best fits the empirical cumulative *CC4CS_j* distributions obtained via the SLIDE-x framework (e.g., *Normal Gaussian*, *Lognormal*, *Beta*, *Weibull*). These distributions were evaluated using *Goodness-Of-Fit* (GOF) metrics, including *NLogN*, *BIC*, *AIC*, and *AICc*. The analysis revealed that the Lognormal distribution is best for GPPs, while the Normal distribution suits SPPs, as shown in Figure 3.

We then outlined an approach to characterize GPPs and SPPs, focusing on estimating distribution parameters (mean μ , standard deviation σ) for specific processors. To derive the values for GPPs, it has been applied the *Moment Matching Approximation* (MMA) method [33], which approximates the statistics of an empirical distribution function, with mean $\hat{\mu}$ and square mean $\hat{\mu}_2$, with a Lognormal random variable $Z = e^x$ such that $X \sim N(\mu_x, \sigma_x^2)$.

⁴ U. of California, Dalton Project: <https://newit.gsu.by>

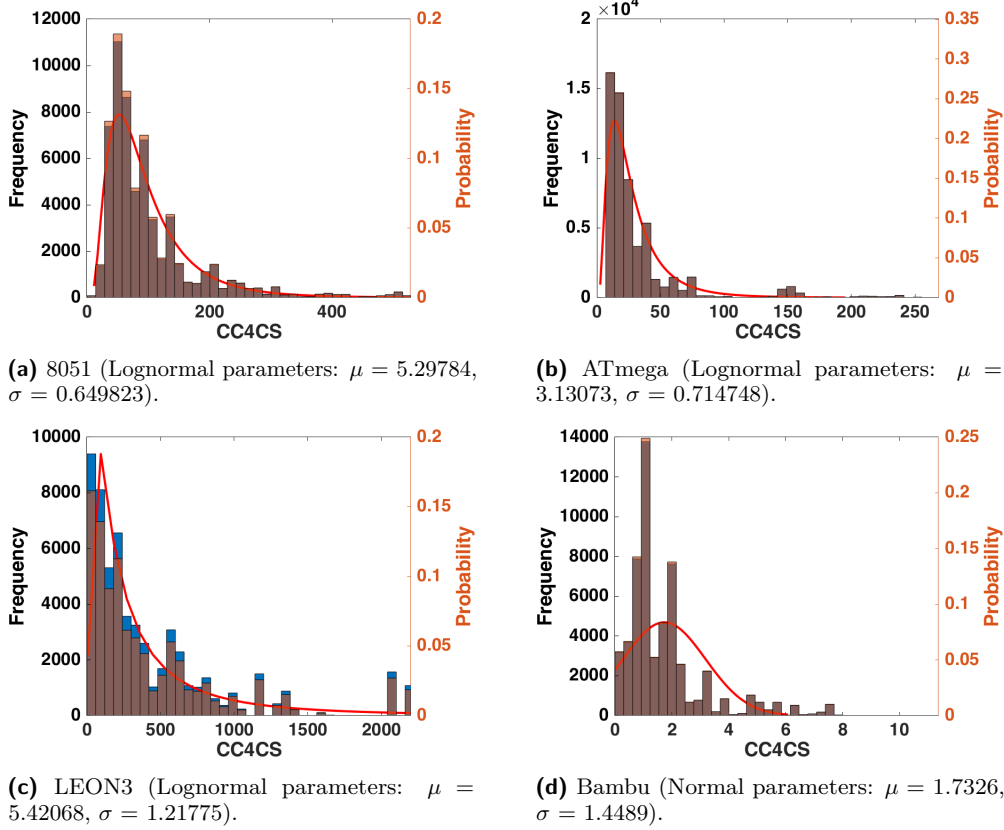
⁵ M. Technology, ATmega328/P: <https://www.microchip.com>

⁶ Gaisler Website: <https://www.gaisler.com/>

⁷ SimulAVR: <http://savannah.nongnu.org>

$$\begin{cases} \hat{\mu} \triangleq E[Z] = E[e^x] \\ \hat{\mu}_2 \triangleq E[Z^2] = E[e^{2x}] \end{cases} \quad \begin{cases} \hat{\mu} = e^{\mu_x + \frac{1}{2} \cdot \sigma_x^2} \\ \hat{\mu}_2 = e^{2 \cdot \mu_x + 2 \cdot \sigma_x^2} \end{cases} \quad \begin{cases} \mu_x = \log \frac{\hat{\mu}^2}{\hat{\mu}_2} \\ \sigma_x^2 = \log \frac{\hat{\mu}_2}{\hat{\mu}^2} \end{cases} \quad (14)$$

The μ and σ parameters for SPPs were set to the arithmetic mean $\hat{\mu}$ and standard deviation $\hat{\sigma}$ of the empirical distribution, with the fitted distribution being the Normal distribution $N(\hat{\mu}, \hat{\sigma}^2)$. These parameters are shown in Figure 3. This approach allows for the performance characterization of any processor technology using $CC4CS_j$. Such characterizations can be included in datasheets or other relevant materials and made available for further analysis.



■ **Figure 3** $CC4CS_j$ sampling distribution and fitted probability density function.

4.2 CLRA Performance Prediction Results

The predictive equations are given in Eq. 12, where τ_j represents the clock period of the HW/SW processor. We use the cumulative distribution function from Section 4.1 to estimate each function's execution time as follows:

$$\bar{t}(ST_{s,k}) = Q_2 \cdot \tau_j \quad \text{AND} \quad \delta(z_s) = 0 \quad \text{for GPPs} \quad (15)$$

$$\bar{t}(ST_{s,k}) = \mu \cdot \tau_j \quad \text{AND} \quad \delta(z_s) = 0 \quad \text{for SPPs} \quad (16)$$

Q_2 represents the median of the lognormal distribution for the 8051, Atmega328/P, and LEON3 processors, while μ is the mean of the normal distribution for the Bambu SPP. To build the CLRA model, the dataset was split into 80% for training ($48 \cdot 10^3$ inputs) and 20% for testing ($12 \cdot 10^3$ inputs). We then used Matlab R2022b's LASSO function with 10-fold cross-validation and the elastic net method, with $\alpha = 1.0$.

3:10 System-Level Timing Performance Estimation

Defining the estimation error as $\epsilon_{s,k} = T_{s,k} - \hat{T}_{s,k}$, we finally evaluate the errors for the different processors p_j using Percentage Error (PE) and Mean Absolute Percentage Error (MAPE) defined as follows [7]:

$$PE_j = \left(\frac{1}{n * d} \sum_{i=1}^n \sum_{k=1}^d \frac{\epsilon_{s,k}}{T_{s,k}} \right) \cdot 100, \quad MAPE_j = \left(\frac{1}{n * d} \sum_{i=1}^n \sum_{k=1}^d \frac{|\epsilon_{s,k}|}{T_{s,k}} \right) \cdot 100 \quad (17)$$

In this work, we have used the R^2 measure of goodness of fit metric and defined an additional reliability metric as follows:

$$REL_j = \frac{2 \cdot \sum_{s=1}^n \sum_{k=1}^d \sum_{r=k+1}^d \mu_{s,k,r}}{n \cdot d \cdot (d-1)}, \quad \text{where } \mu_{s,k,r} = \begin{cases} 0 & \text{if } (\hat{T}_k > \hat{T}_r \text{ and } T_k < T_r) \\ & \text{or if } (\hat{T}_k < \hat{T}_r \text{ and } T_k > T_r) \\ 1 & \text{otherwise} \end{cases}$$

Table 2 shows Pearson correlation and slope values between clock cycles and executed C statements. Correlations for 8051 and ATmega328/P are lower (< 0.9), while LEON3 is close to 1. The slope indicates estimation uncertainty increases with input data bits for Atmega and 8051 but remains stable for LEON3 and Artix-7. The table also shows that 8051 performs worst with float data types due to the lack of an FPU, while Bambu has the lowest correlation ($< 50\%$).

■ **Table 2** CC4CS HW/SW Statistical Analysis results (p-value $\ll 0.001$ for every value).

p_j	Data Type Corr. ¹				Data Type Slope ³			
	int8	int16	int32	float	int8	int16	int32	float
LEON3	0.993	0.919	0.9280	0.973	341.086	335.759	343.400	335.705
ATmega	0.849	0.905	0.976	0.934	8.633	10.755	14.582	24.624
8051	0.994	0.987	0.928	0.747	85.829	106.111	129.371	247.771
Artix7	0.424	0.372	0.362	0.408	2.250	2.300	2.289	3.273

a) ¹ Corr.: Pearson Correlation; ² Slope: Regression Slope Parameter;

Table 3 shows the results from the CLRA approach. Generally, the table reports high reliability and R^2 values for most input types, with some exceptions due to underfitting caused by data inconsistency or imbalance (e.g., Atmega int32 or Bambu int8). Despite lower reliability and R^2 compared to other SW processors, LEON3 has the lowest mean error in PE (from -1.37% to 6.08%) and MAPE (from 0.34% to 2.58%) due to caches and pipelines creating a stronger linear link between executed statements and clock cycles. Atmega shows the largest errors and p-values, while 8051 has smaller errors due to its CISC architecture, which has a more linear dependency between statements and clock cycles. Artix 7 shows a smaller PE range (from -2.12% to 0.36%) but higher MAPE (from 6.54% to 11.84%) and stronger R^2 and reliability compared to SW processors. This is because synthesis in HW depends on data size rather than input values. Despite Bambu's low correlation and higher errors for 8-bit types, the approach performs well for HW processors, with errors consistently below 10% for data types larger than 8 bits. Based on the works listed in Table 1, our approach consistently outperforms other works for SW processors like the 8051 and LEON3. For HW processors, other techniques may give better results but at the cost of longer execution times and greater resource demands [17]. Our approach, leveraging the CC4CS metric, provides accurate estimations for both HW and SW technologies at the system level. Errors are always below 10% for 8-bit CISC and 32-bit RISC processors with caches and pipelines (e.g., 67% more accurate than [14] for LEON3 with -O0 flag). Our approach also eliminates the need to compile/run target code for each architecture, using

a linear LASSO model to represent processor behavior. Although computing the CC4CS metric for new processors can take hours, once completed, estimation time for new data is minimal. Solving the CLRA optimization takes around 1 minute, with execution time estimation negligible.

■ **Table 3** CLRA Performance results across various data type sizes and architectural targets. H_0 : MAPE \geq 10%.

Target	Intel MCS51					AVR Atmega328/P				
Metrics	int8	int16	int32	float	AVG	int8	int16	int32	float	AVG
PE (%)	-16.03	6.06	8.16	-6.10	-1.98	-2.97	-1.39	-14.04	-2.82	-5.30
MAPE (%)	5.13	6.53	6.88	4.26	5.70	9.10	12.15	22.04	6.99	12.57
p-value	0.0211	0.0397	0.0161	2.7E-04	2.7E-06	0.9041	0.9622	0.9889	0.8598	0.9994
Rel	0.925	0.924	0.931	0.925	0.926	0.929	0.930	0.935	0.937	0.933
R^2	0.969	0.963	0.981	0.982	0.974	0.975	0.980	0.986	0.989	0.982
Target	Sparc-V8 LEON3					Xilinx Artix7 (XC7A35T)				
Metrics	int8	int16	int32	float	AVG	int8	int16	int32	float	AVG
PE (%)	6.08	0.18	3.90	-1.37	2.20	-2.12	-0.15	-0.19	0.36	-0.52
MAPE (%)	1.40	2.34	2.58	0.34	1.66	11.84	6.54	7.09	6.65	8.03
p-value	7.2E-07	1.6E-05	2.1E-05	8.9E-20	1.1E-22	0.727	8.6E-04	0.001	9.3E-04	0.0033
Rel	0.922	0.923	0.919	0.935	0.925	0.939	0.937	0.940	0.946	0.940
R^2	0.892	0.906	0.930	0.960	0.922	0.985	0.999	0.999	0.999	0.996

While extracting the initial dataset is time-consuming, our approach greatly reduces prediction times compared to Bambu, lowering prediction errors after data collection and training. The overall speed-up reaches up to 32x (\approx 97%). In comparison, state of the art report a 17% reduction using NNs [15] and 43.78% with RFs [28], both lower than the LASSO model's speed-up. Thus, our approach offers a significant speed-up over traditional HLS methods. For SW processors, the key advantage is model portability and the ability to evaluate performance across various inputs and code complexities.

5 Threats to validity

The *internal validity* may be influenced by how the CLRA model is trained, as MAPE values in Table 3 are based on a dataset where 80% was used for training and 20% for testing. This could reduce the observed error. In the future, we plan to introduce a control group for training. For *external validity*, the main concern is generalizability. Results may vary with different HW/SW environments or workloads outside the training set. To address this, we validated the approach using well-known benchmarks, though future work may need to include more diverse inputs and benchmarks. *Construct validity* may be affected by the characteristics of the selected processors and FPGA (Intel 8051, Atmega328p, LEON3, Artix7). Factors like cache, virtual memory, and external memory could influence performance, and our small set of simple HW limits the generalization of our claims. Additionally, our assumption that CC4CS values follow lognormal and normal distributions may not hold for more complex platforms. *Conclusion validity* concerns the reliability of our findings. We used appropriate statistical tests to avoid biases and errors, and we have made our repository available to reproduce and validate our work¹.

6 Conclusion and Future Work

This work presents a system-level performance estimation approach using the CC4CS, a unified HW/SW metric for early performance estimations. The paper formalizes this metric and proposes an estimator based on statistical analysis. Experiments validate the approach, showing effectiveness with an estimation error below 10% and an estimation time close to 0. As shown in Table 1, our method enables system-level estimation without compiling and running the code on each architecture. It uses statistical analysis and linear regression to model different HW/SW processors. While calculating the CC4CS metric for a new processor may take hours, the subsequent estimates for new C functions are immediate. Furthermore, the estimator provides reliable predictions of execution times with limited error. Future work will (1) increase the amount of data extracted through also the usage of advanced observability mechanisms [32] and generate models for common compiler configurations; (2) integrate more HLS tools and ISSs (e.g., RISC-V, ARM, Vitis HLS), targeting various FPGA families [23], heterogeneous targets [31], and SW processors with advanced micro-architectural features (e.g., pipelines); (3) enrich the reference benchmark by considering different sources [9] across various application domains [12, 30]; (4) improving the exploitation of non-linear ML models (e.g., SVM, Regression Trees, Random Forest, Neural Networks) [19]; (5) extend the approach with additional statistics (e.g., Kolmogorov-Smirnov tests, ANOVA, t-tests) [18].

References

- 1 Vikas Agrawal, Anand Pande, and Mahesh M. Mehendale. High level synthesis of multi-precision data flow graphs. In *VLSI Design 2001. Fourteenth International Conference on VLSI Design*, pages 411–416, 2001. doi:10.1109/ICVD.2001.902693.
- 2 Abderaouf Nassim Amalou, Elisa Fromont, and Isabelle Puaut. Fast and accurate context-aware basic block timing prediction using transformers. In *Proceedings of the 33rd ACM SIGPLAN International Conference on Compiler Construction*, CC 2024, pages 227–237, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3640537.3641572.
- 3 Yunsheng Bai, Atefeh Sohrabizadeh, Zongyue Qin, Ziniu Hu, Yizhou Sun, and Jason Cong. Towards a comprehensive benchmark for high-level synthesis targeted to fpgas. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024. Curran Associates Inc.
- 4 Carlo Brandolese, William Fornaciari, Luigi Pomante, Fabio Salice, and Donatella Sciuto. Affinity-driven system design exploration for heterogeneous multiprocessor soc. *IEEE Transactions on Computers*, 55(5):508–519, May 2006. doi:10.1109/TC.2006.66.
- 5 Carlo Brandolese, William Fornaciari, Fabio Salice, and Donatella Sciuto. Source-level execution time estimation of c programs. In *Proceedings of the Ninth International Symposium on Hardware/Software Codesign*, pages 98–103, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/371636.371694.
- 6 Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H. Anderson, Stephen Brown, and Tomasz Czajkowski. Legup: high-level synthesis for fpga-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '11, pages 33–36, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1950413.1950423.
- 7 Steve Dai, Yuan Zhou, Hang Zhang, Ecenur Ustun, Evangeline F.Y. Young, and Zhiru Zhang. Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 129–132, 2018. doi:10.1109/FCCM.2018.00029.

- 8 P. Derler, E. A. Lee, and A. Sangiovanni Vincentelli. Modeling cyber–physical systems. *Proceedings of the IEEE*, 100(1):13–28, 2012.
- 9 Tania Di Mascio, Luigi Laura, and Marco Temperini. A framework for personalized competitive programming training. In *2018 17th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–8, 2018. doi:10.1109/ITHET.2018.8424620.
- 10 Daniele Di Pompeo, Emilio Incerto, Vittoriano Muttillio, Luigi Pomante, and Giacomo Valenete. An efficient performance-driven approach for hw/sw co-design. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE '17*, pages 323–326, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3030207.3030239.
- 11 Christof Ebert and Capers Jones. Embedded software: Facts, figures, and future. *Computer*, 42(4):42–52, 2009. doi:10.1109/MC.2009.118.
- 12 Paolo Giammatteo, Federico Vincenzo Fiordigigli, Luigi Pomante, Tania Di Mascio, and Federica Caruso. Age & gender classifier for edge computing. In *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, 2019. doi:10.1109/MECO.2019.8760160.
- 13 José Andrés Jiménez, José Amelio Medina Merodio, and Luis Fernández Sanz. Checklists for compliance to do-178c and do-278a standards. *Computer Standards & Interfaces*, 52:41–50, 2017. doi:10.1016/j.csi.2017.01.006.
- 14 Marco Lattuada and Fabrizio Ferrandi. Performance modeling of embedded applications with zero architectural knowledge. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 277–286, 2010. doi:10.1145/1878961.1879010.
- 15 Hosein Mohammadi Makrani, Hossein Sayadi, Tinoosh Mohsenin, Setareh rafatirad, Avesta Sasan, and Houman Homayoun. Xppe: Cross-platform performance estimation of hardware accelerators using machine learning. In *ASPDAC '19, ASPDAC '19*, pages 727–732, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3287624.3288756.
- 16 Dimosthenis Masouros, Aggelos Ferikoglou, Georgios Zervakis, Sotirios Xydis, and Dimitrios Soudris. Late breaking results: Language-level qor modeling for high-level synthesis. In *Proceedings of the 61st ACM/IEEE Design Automation Conference, DAC '24*, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3649329.3663500.
- 17 Hosein Mohammadi Makrani, Farnoud Farahmand, Hossein Sayadi, Sara Bondi, Sai Manoj Pudukotai Dinakarrao, Houman Homayoun, and Setareh Rafatirad. Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 397–403, 2019. doi:10.1109/FPL.2019.00069.
- 18 Vittoriano Muttillio, Claudio Di Sipio, Riccardo Rubei, Luca Berardinelli, and MohammadHadi Dehghani. Towards synthetic trace generation of modeling operations using in-context learning approach. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE '24*, pages 619–630, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3691620.3695058.
- 19 Vittoriano Muttillio, Paolo Giammatteo, and Vincenzo Stoico. Statement-level timing estimation for embedded system design using machine learning techniques. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering, ICPE '21*, pages 257–264, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3427921.3450258.
- 20 Vittoriano Muttillio, Paolo Giammatteo, Vincenzo Stoico, and Luigi Pomante. An early-stage statement-level metric for energy characterization of embedded processors. *Microprocessors and Microsystems*, 77:103200, 2020. doi:10.1016/J.MICPRO.2020.103200.
- 21 Vittoriano Muttillio, Luigi Pomante, Marco Santic, and Giacomo Valente. Systemc-based co-simulation/analysis for system-level hardware/software co-design. *Computers and Electrical Engineering*, 110:108803, 2023. doi:10.1016/j.compeleceng.2023.108803.

- 22 Vittorioano Muttillio and Vincenzo Stoico. SLIDE-x (System-Level Infrastructure for HW/SW Dataset E-xtraction). Software, version 1.0., swHId: swh:1:rev:0907cf39f7d023d0dc2e8307e1ffef6115d0377a (visited on 2025-02-12). URL: <https://github.com/hepsycode/SLIDE-x>, doi:10.4230/artifacts.22912.
- 23 Vittorioano Muttillio, Vincenzo Stoico, Marco Santic, Giacomo Valente, Luigi Pomante, and Daniele Frigioni. Slide-x-ml: System-level infrastructure for dataset e-xtraction and machine learning framework for high-level synthesis estimations. In *2024 IEEE 42nd International Conference on Computer Design (ICCD)*, pages 616–619, 2024. doi:10.1109/ICCD63220.2024.00098.
- 24 Vittorioano Muttillio, Giacomo Valente, Daniele Ciabrone, Vincenzo Stoico, and Luigi Pomante. Hepsycode-rt: a real-time extension for an esl hw/sw co-design methodology. In *Proceedings of the Rapido'18 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '18, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3180665.3180670.
- 25 Vittorioano Muttillio, Giacomo Valente, Fabio Federici, Luigi Pomante, Marco Faccio, Carlo Tieri, and Serenella Ferri. A design methodology for soft-core platforms on fpga with smp linux, openmp support, and distributed hardware profiling system. *Eurasip Journal on Embedded Systems*, 2016(1), 2017. doi:10.1186/s13639-016-0051-9.
- 26 Vittorioano Muttillio, Giacomo Valente, Luigi Pomante, Vincenzo Stoico, Fausto D'Antonio, and Fabio Salice. Cc4cs: An off-the-shelf unifying statement-level performance metric for hw/sw technologies. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE '18*, pages 119–122, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3185768.3186291.
- 27 Razvan Nane, Vlad-Mihai Sima, Christian Pilato, Jongsok Choi, Blair Fort, Andrew Canis, Yu Ting Chen, Hsuan Hsiao, Stephen Brown, Fabrizio Ferrandi, Jason Anderson, and Koen Bertels. A survey and evaluation of fpga high-level synthesis tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(10):1591–1604, October 2016. doi:10.1109/TCAD.2015.2513673.
- 28 Kenneth O'Neal, Mitch Liu, Hans Tang, Amin Kalantar, Kennen DeRenard, and Philip Brisk. Hlspredict: Cross platform performance prediction for fpga high-level synthesis. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2018. doi:10.1145/3240765.3240816.
- 29 Jürgen Teich. Hardware/software codesign: The past, the present, and predicting the future. *Proceedings of the IEEE*, 100(Special Centennial Issue):1411–1430, 2012. doi:10.1109/JPROC.2011.2182009.
- 30 Walter Tiberti, Federica Caruso, Luigi Pomante, Marco Pugliese, Marco Santic, and Fortunato Santucci. Development of an extended topology-based lightweight cryptographic scheme for iee 802.15.4 wireless sensor networks. *International Journal of Distributed Sensor Networks*, 16(10):1550147720951673, 2020. doi:10.1177/1550147720951673.
- 31 Giacomo Valente, Gianluca Brilli, Tania Di Mascio, Alessandro Capotondi, Paolo Burgio, Paolo Valente, and Andrea Marongiu. Fine-grained qos control via tightly-coupled bandwidth monitoring and regulation for fpga-based heterogeneous socs. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–15, 2024. doi:10.1109/TPDS.2024.3513416.
- 32 Giacomo Valente, Tiziana Fanni, Carlo Sau, Tania Di Mascio, Luigi Pomante, and Francesca Palumbo. A composable monitoring system for heterogeneous embedded platforms. *ACM Trans. Embed. Comput. Syst.*, 20(5), July 2021. doi:10.1145/3461647.
- 33 R. Valentini, P.D. Marco, R. Alesii, and F. Santucci. Cross-layer analysis of multi-static rfid systems exploiting capture diversity. *IEEE Transactions on Communications*, 69(10):6620–6632, 2021. doi:10.1109/TCOMM.2021.3096541.
- 34 Weiyan Zhang, Mehran Goli, and Rolf Drechsler. Early performance estimation of embedded software on risc-v processor using linear regression. In *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 20–25, 2022. doi:10.1109/DDECS54261.2022.9770144.