A New HW/SW Co-Design Approach for Monitored Systems-on-Chip Development

GIACOMO VALENTE, University of L'Aquila, L'Aquila, Italy
VITTORIANO MUTTILLO, University of Teramo, Teramo, Italy
LUIGI POMANTE, University of L'Aquila, L'Aquila, Italy
DANIELE FRIGIONI, University of L'Aquila, L'Aquila, Italy
TANIA DI MASCIO, University of L'Aquila, L'Aquila, Italy

As embedded systems are required to satisfy increasing functional and non-functional requirements, heterogeneous systems-on-chip architectures are progressively adopted. While these complex systems-on-chip deliver high performance, they require efficient coordination of the tasks they carry out. To tackle this challenge, designers often resort to runtime mechanisms allowing the dynamic alignment of application requirements with platform services. In turn, runtime mechanisms require the adoption of on-chip monitoring systems. The integration of on-chip monitoring systems into a system-on-chip results in a monitored system-on-chip. Notwithstanding, this integration risks driving a re-design and a re-implementation of the whole system-on-chip, potentially driving to a time-to-market deadline miss. In the literature, HW/SW co-design approaches for monitored systems-on-chip have been proposed to overcome the problem. However, the existing HW/SW co-design approaches prevent performing a system-level design-space exploration that involves all the monitoring requirements, and they also prevent adequate reuse of existing on-chip monitoring systems. This paper proposes an approach for efficient HW/SW co-design of monitored systems-on-chip, aiming to comprehensively capture all monitoring requirements at the system-level and to perform a system-level design-space exploration to satisfy them, enforcing the reuse of existing on-chip monitoring systems. The proposed approach is validated through two experimental activities, which demonstrate a 24% reduction in total development time for a monitored system-on-chip implemented on FPGA, compared to the customary approach. The results also show that the approach reduces the risk of missing time-to-market deadlines, supports flexible design choices, and enables the reuse of on-chip monitoring systems.

CCS Concepts: • **Hardware** → **Methodologies for EDA**; **Methodologies for EDA**; • **Computer systems organization** → *System on a chip*.

Additional Key Words and Phrases: Design methodologies, HW/SW co-design, On-Chip Monitoring, Heterogeneous systems-on-chip.

1 Introduction

As recently highlighted by several industrial associations (e.g., see [17]), next-generation computing systems, especially in the embedded domain, need to be connected, intelligent, autonomous, and evolvable. Designers have started to address such needs by exploiting heterogeneous Systems-on-Chip (SoCs) architectures (e.g., see [42]). These complex SoCs provide high performances, but designers face a significant challenge in simultaneously

Authors' Contact Information: Giacomo Valente, University of L'Aquila, L'Aquila, Italy; e-mail: giacomo.valente@univaq.it; Vittoriano Muttillo, University of Teramo, Teramo, Abruzzo, Italy; e-mail: vmuttillo@unite.it; Luigi Pomante, University of L'Aquila, L'Aquila, Italy; e-mail: luigi.pomante@univaq.it; Daniele Frigioni, University of L'Aquila, L'Aquila, Italy; e-mail: daniele.frigioni@univaq.it; Tania Di Mascio, University of L'Aquila, L'Aquila, Italy; e-mail: tania.dimascio@univaq.it.

ACM 1558-3465/2025/9-ART
https://doi.org/10.1145/3769075

satisfying the evolving demand in terms of *Functional Requirements* – the functionality required to a system – and *Non-Functional Requirements* – the constraints on the design metrics to be satisfied by the system.

A common way to face such a challenge is the adoption of runtime mechanisms, which allow the matching of the evolving functional requirements and non-functional requirements of the applications with the offered platform services [13, 31]. Runtime mechanisms deal, among others, with phenomena and events to be observed, that can be formalized through *Monitoring Requirements* (MREQs).

MREQs refer to requirements for monitoring actions; specifically, MREQs *can be of two types*: those referring to the behavior of the system and those referring to the structure of the system [31]. For instance, an MREQ referring to system behavior may require the tracking of power dissipation during the execution of a real-time image task to meet energy constraints, as commonly done in vision-based automotive applications [66]. Conversely, a structural MREQ may require monitoring the number of cache misses experienced by a processor core executing a real-time operating system, as frequently found in safety-critical platforms [9].

MREQs are satisfied through the integration of *on-Chip Monitoring Systems* (oCMSs), which are HW or SW elements able to observe the system within the SoC, obtaining a *monitored SoC*. Examples of HW-oCMSs include HW performance monitors such as ARM ETM [2] or Intel Performance Monitoring Units [26], which offer non-intrusive visibility into processor events like instruction count or cache activity. SW-based oCMSs, like Linux perf [43] or instrumentation libraries [18], are often used for monitoring task scheduling latencies or I/O bandwidth in general-purpose embedded processors.

Monitored SoCs are customarily obtained as follows: first, the SoC is developed; after, one or more oCMSs are integrated into it [31]. An example illustrating the oCMSs integration within a SoC developed through a HW/SW co-design flow is depicted in the timing diagram of Fig. 1 (see [52]): there, the SoC is completed by month 17. Then, MREQs are met through oCMSs integration (e.g., source-code instrumentation). However, if the integration of oCMSs within the SoC results in the monitored SoC failing to meet its functional or non-functional requirements (a situation not uncommon, as evidenced in [31]), it may necessitate a complete redesign and re-implementation. Fig. 1 illustrates precisely this case, where the re-design and re-implementation extend the completion of the monitored SoC to month 28 (i.e., a development time increase of nearly 65%).

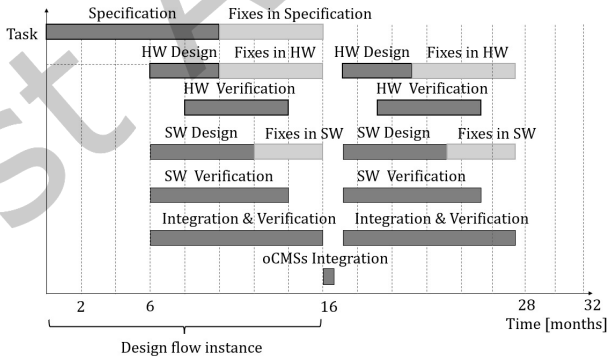


Fig. 1. Timing diagram showing the oCMSs integration in a SoC developed through a HW/SW co-design flow, showcasing the impact on design time (adapted from [52]).

Consequently, there is an increased risk of missing the time-to-market deadline.

Coherently with similar problems (e.g., design for throughput [53] and for power [29]), *a way to overcome the time-to-market deadline miss is to reconsider both the development process of the whole SoC and the integration*

process of oCMSs into it. In which way to reconsider the development process is guided by seminal works (e.g., [30]), highlighting that design methodologies for complex SoCs have to include the following aspects:

(i) the guarantee that all the requirements are captured at the **highest level** of abstraction, to be able to exploit all the degrees of freedom that are available; such a level of abstraction should not make any distinction between HW and SW, since such a distinction is the consequence of a design decision;
(ii) the possibility for designers to trade off among different design metrics by exploring different SoC platform solutions – **design space exploration** – before system implementation;
(iii) the capability to favor **reuse**, mainly to reduce the time-to-market pressure.

In general, HW/SW co-design methodologies for developing heterogeneous SoCs include all three key aspects. According to seminal papers on HW/SW co-design methodologies (e.g., see [52]), these methodologies start with specifying functional and non-functional requirements at the system-level through a formal behavioral model. Next, a design-space exploration is conducted to develop a system structural model that meets the requirements. Together with the design-space exploration, co-estimation and co-simulation are used to early assess the SoC capability (i.e., the first 10 months in Fig. 1). Lastly, a low-level HW/SW implementation activity leads to the production of the final SoC (i.e., from month 11 to month 17 in Fig. 1).

Actually, in the case of monitored SoCs, **the existing HW/SW co-design methodologies do not yet consider simultaneously the (i), (ii), and (iii) aspects**. In particular, there are:

- works that capture only a type of MREQs (i.e., those referring to the behavior of the system or those referring to the structure of the system) [14, 19, 23, 24, 33, 38, 47–49, 58, 65]. In turn, this forces to capture only specific requirements, hence *preventing to capture all the requirements at system-level*;
- works that address MREQs forcing designers to use specific oCMSs [14, 33, 47–49], *limiting the possibility of designers to perform a design space exploration*;
- works that address MREQs with ad-hoc created oCMSs [14, 23, 24, 38, 65], *preventing the reuse of existing oCMSs*.

To include all three aspects (i), (ii), and (iii) in designing monitored SoCs, in this paper, we propose an approach **enabling a HW/SW co-design dedicated to monitored SoCs**. The proposed approach:

- allows capturing both types of MREQs at system-level (i.e., the ones referred to the behavior and to the structure of the system), taking care of the aspect (i) of capturing all the requirements at system-level;
- allows performing a design-space exploration of monitored SoCs, enabled by the definition of a new intermediate representation at system-level, taking care of the aspect (ii); we will refer to this representation as *proposed monitoring model*;
- allows satisfying MREQs by reusing existing oCMSs, taking care of the aspect (iii).

In order to evaluate the effectiveness of the proposed approach, in this paper we also report two experimental activities. The first deals with the design of a monitored SoC to execute a synthetic application. This activity serves to demonstrate the effectiveness of the proposed approach in capturing and managing at system-level MREQs targeting both the system behavior and the system structure – aspect (i). Moreover, the experiment also highlights how the proposed approach supports the oCMSs reuse – aspect (iii). The second experimental activity deals with the design of a monitored SoC for a pacemaker with reliability requirements. This activity serves to demonstrate the effectiveness of the proposed approach in allowing a design-space exploration to trade-off among the design metrics of response time and occupied resources – aspect (ii).

This paper gets over the previous work in progress [57] that proposed a design methodology for HW/SW co-design of monitored embedded SoCs. In fact, in [57] we only defined an innovative topic referred to as Design for Monitorability, providing a line to exploit HW/SW co-design techniques to satisfy MREQs by considering

them since the initial steps of the design flow. Going more in depth, in this paper we propose and detail an overall approach and we demonstrate its effectiveness through structured experimental results.

The paper is organized as follows: Section 2 illustrates the related works and how their approaches only partially include aspects (i), (ii), and (iii); Section 3 reports the necessary preliminaries about HW/SW co-design. These preliminaries serve as underpinning knowledge for the proposed monitoring model, reported in Section 4, while Section 5 introduces the proposed approach for HW/SW co-design of monitored SoCs. Section 6 discusses the experimental evaluation, while Section 7 concludes the paper.

2 Related Works

Being the goal of this paper to provide an approach to perform the HW/SW co-design of monitored SoCs, this section reports, to the best of our knowledge, the most significant works about design approaches for monitored SoCs.

We report on the works in the literature that captured MREQs at system-level (i.e., the highest abstraction level in the design process), and we analyze the type of system *view* they refer to. Specifically, MREQs can be categorized based on whether they concern the behavior or the structure of the system [31]. This distinction is orthogonal to the abstraction level: both behavioral and structural views may be adopted at different stages of the design process (e.g., functional, algorithmic, system-level).

- **Behavioral MREQs**: these MREQs allow the designer to express the need to monitor the actions of the system during execution, abstracting from the actual implementation. The behavioral view typically targets tasks, dataflows, and functional interactions without requiring details of how they are implemented on HW or SW components. For example, a designer may request the monitoring of power consumption for a specific data processing task or the end-to-end latency across a logical communication path between tasks.
- **Structural MREQs**: these MREQs allow the designer to express the need to monitor the components of the system architecture. The structural view refers to concrete platform elements such as processors, memories, buses, interconnects, and other HW components. For instance, a designer may require monitoring the number of cache misses on a specific processor core or the bandwidth usage on a particular communication bus.

The work [14] presents a framework that captures functional requirements, non-functional requirements, and MREQs for runtime verification at system-level, using temporal logic equations. The MREQs considered in this work refer to the behavioral view of the system, i.e., it is possible to express behavioral MREQs. The system is then generated by producing SW to be executed on a general-purpose processor; additionally, oCMSs based on code instrumentation are automatically generated. Notwithstanding, [14] does not support the specification of MREQs referring to the structural view of the system, i.e., it is not possible to express structural MREQs. Moreover, the nature of its framework prevents the use of alternative oCMSs, thereby limiting both the design space exploration and the reuse of existing oCMSs.

Similarly, the works [33, 47–49] allow to capture MREQs at system-level and, at the same time, enforce the reuse of oCMSs. These works are all based on the same methodology, with some differences among each other. The methodology captures MREQs at system-level, together with functional requirements and non-functional requirements. Then, a runtime monitoring graph is extracted and used to configure a reference oCMS. In particular, the works [33, 48, 49] allow to express MREQs using three different modeling languages (periodic state machines, UML sequence diagrams, and UML activity diagrams, respectively), and all of them configure the same reference oCMS. Instead, the work [47] captures MREQs through timed automata and configures a Micron Automata processor as the final oCMS. Notwithstanding, these four works again allow to express behavioral MREQs but not structural MREQs, and they do not allow performing a design-space exploration.

Table 1. Comparison of the approach proposed in this paper with state-of-the-art works.

Aspects	Features	[14]	[33]	[49]	[48]	[47]	[24]	[23]	[38]	[65]	[19]	[58]	This Work
(i)	MREQs at system-level	✓	✓	✓	✓	✓	✓						✓
	Behavioral MREQs	✓	✓	✓	✓	✓	✓						✓
	Structural MREQs							✓	✓	✓	✓	✓	✓
(ii)	Design Space Exploration						✓	✓	✓	✓	✓	✓	✓
(iii)	Reuse of oCMSs		✓	✓	✓	✓					✓	✓	✓

The only work that allows a design space exploration after capturing the MREQs at system-level is the framework proposed in [24], as during the HW/SW co-design flow, it allows choosing between HW and SW elements to observe the system. However, [24] allows expressing only behavioral MREQs, and it does not support reuse of oCMSs.

Complementary to the works discussed until now, the works [19, 23, 38, 58, 65] capture the structural MREQs. However, these MREQs are not captured at system-level, as they expect designers to provide their MREQs when the system has distinct parts in SW and HW; in turn, this prevents working at the highest level of abstraction. Despite this, we here report these works since they allow to perform a design-space exploration of monitored SoCs, with some of them allowing the reuse.

For example, the works [23, 38, 65] provide a design flow where MREQs are captured when the system is described at the register transfer level ([38, 65]), or at the algorithmic level ([23]). In both cases, they do not target the system-level. In their approaches, oCMSs are automatically generated through high-level synthesis based on input MREQs, in turn preventing the reuse of existing oCMSs.

On the other hand, in the works [19, 58], after providing MREQs, they both allow a design-space exploration and reusing existing oCMSs to satisfy MREQs.

Table 1 summarizes how the related works in the literature behave with respect to the aspects (i), (ii), and (iii) mentioned in the Introduction. The second column reports the features required to ensure that the corresponding aspect is covered. The first row highlights if the considered work allows capturing *MREQs at system-level*. The second and third rows refer to works that consider Behavioral MREQs and Structural MREQs, respectively. The fourth row indicates the works allowing a *monitored SoC design space exploration*. Finally, the fifth row points out if the considered work supports the *reuse of oCMSs*. What is clear from Table 1 is that, to the best of our knowledge, no works include simultaneously all three aspects. By analyzing the reported related works, it is clear that the road to follow to include the three aspects consists of:

- developing a sufficiently detailed model able to capture both types of MREQs at system-level;
- allowing designers to satisfy the captured MREQs through oCMSs of different types (HW or SW elements), enabling a design-space exploration;
- allowing designers to search oCMSs among existing ones, enabling reuse.

3 Preliminaries

This section outlines the fundamental concepts of HW/SW co-design methodology and describes the Gajski-Kuhn chart [22], which illustrates the various levels of abstraction for system modeling. Table 2 summarizes the main symbols used in this paper.

A HW/SW co-design flow [52] consists of two main phases: system development and system deployment. The system development phase includes three key activities: requirements specification, system-level synthesis, and low-level implementation. Fig. 2 presents the flowchart of the reference HW/SW co-design, which is detailed below. It is worth noting that, with reference to Fig. 1, the first two activities can be mapped to the initial 10 months, while the third activity corresponds to the period from month 11 to month 17.

Table 2. Table of symbols used in this paper, grouped according to the sections in which they appear throughout the paper.

Symbol	Description
$MREQs$	Monitoring requirements
$oCMSs$	On-chip monitoring systems
Γ, Λ	Set of tasks and logical links
τ_r, λ_t	*r-th* task and *t-th* logical link
FR	Behavioral model of functional requirements
NFR	Non-functional requirements
C	Set of constraints on design metrics
P, Φ, M	Set of processors, links, and memories
n_p, n_ϕ, n_m	Number of items in P, Φ, and M
p_i, ϕ_j, m_q	*i-th* processor, *j-th* link, *q-th* memory
rt	Max Response time
res	Max Resource usage on FPGA
pwd	Max Power consumption

Symbol	Description
Ω	Set of predefined metrics
ω	A selected monitoring metric
Γ_M, Λ_M	Set of monitoring tasks and links
$\mu\tau_f, \mu\lambda_s$	*f-th* monitoring task and *s-th* monitoring link
ic, dc	If true, p_i has instruction/data cache
$ct, prot$	Communication type and protocol of ϕ_j
pd	Timing overhead of an oCMS (in clock cycles)
fx, rcf	If true, implementation is fixed or reconfigurable
tpi	Type of implementation
pt, isa	Processor type and instruction set related to p_i
$IBMREQ$	Implicit behavioral MREQs
$EBMREQ$	Explicit behavioral MREQs
$SMREQ$	Structural MREQs
f^{comp}, f^{comm}	Computation and communication synthesis
FR^+	Model given by $(\Gamma \cup \Gamma_M, \Lambda \cup \Lambda_M)$
Γ^+, Λ^+	Extended sets of tasks and links
f_{p_i}, f_{ϕ_j}	Frequency of involvement for p_i, ϕ_j
N^{DB}	Number of oCMSs in the database
Q^ω	Matrix containing the frequencies of involvement
$oCMS^\omega$	Matrix collecting oCMSs compatible with the MREQs
Selected$^+$	Structural model produced by design space exploration step
n^*_p, n^*_ϕ, n^*_m	Components part of Selected$^+$
Selected^{end}	Structural model produced by the proposed approach
FFG	FIR-FIR-GCD
f^{HR}	Heart-rate input to pacemaker

The **requirements specification activity** involves collecting, analyzing, and specifying the needs to be satisfied by the system, resulting in a formal behavioral model of functional requirements, alongside non-functional requirements annotated on the behavioral model.

Without loss of generality, we express the formal behavioral model of functional requirements as follows:

$$FR = (\Gamma, \Lambda)$$

where: $\Gamma = \{\tau_1, \cdots, \tau_r, \cdots, \tau_{n_\tau}\}$ is a set of n_τ tasks and $\Lambda = \{\lambda_1, \cdots, \lambda_t, \cdots, \lambda_{n_\lambda}\}$ is a set of n_λ logical links. A link λ_t connects an ordered pair of tasks. If λ_t starts from τ_r and ends on τ_{r+1}, τ_r is referred to as *Tail* and τ_{r+1} is referred to as *Head*. Fig. 3a shows an example of *FR*.

Non-functional requirements are hard to represent in a common way [11].

Being the goal of this paper to provide an approach that, starting from some basic components, supports designers in HW/SW co-design of monitored SoCs able to satisfy constraints on some design metrics, without loss of generality we express non-functional requirements as:

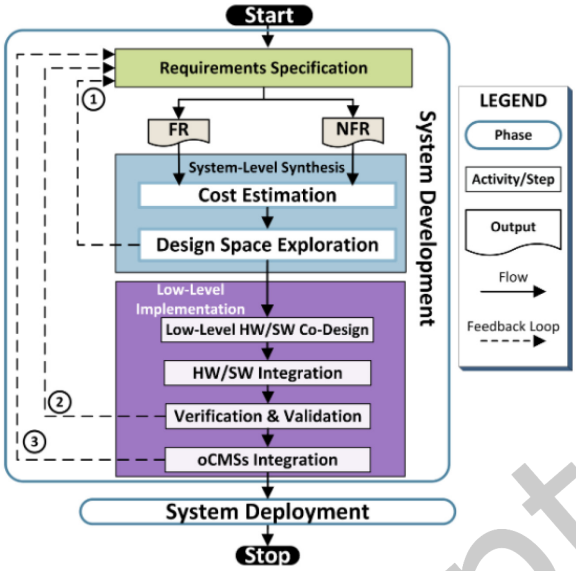


Fig. 2. Flow-chart of the reference HW/SW co-design flow.

$$NFR = \{C, P, \Phi, M\}$$

where C indicates the required constraints on design metrics, while P, Φ, M indicates the basic components to be used to build the system structural model. In more detail:

- C expresses the constraints given in relation to design metrics; customarily [52], they are:

$$C = \{rt, area, res, pwd, energy\} \quad (1)$$

  in which $rt \in \mathbb{R}$ is the response time, expressed in milliseconds; $area \in \mathbb{R}$ is the physical space the system occupies on a silicon die, and it is expressed in squared millimeters; $res = \{lut \in \mathbb{N}, ff \in \mathbb{N}, dsp \in \mathbb{N}, bram \in \mathbb{N}\}$ represents the resources of a Field Programmable Gate Array (FPGA), expressed as number of look-up tables (*lut*), flip-flops (*ff*), digital-signal processors (*dsp*), and block RAM (*bram*); $pwd \in \mathbb{R}$ is the power dissipation, expressed in mW; $energy \in \mathbb{R}$ is the energy consumption, expressed in mJ.
- $P = \{p_1, \cdots, p_i, \cdots, p_{n_p}\}$ is a set of n_p processing units, $\Phi = \{\phi_1, \cdots, \phi_j, \cdots, \phi_{n_\phi}\}$ is a set of n_ϕ physical links, and $M = \{m_1, \cdots, m_q, \cdots, m_{n_m}\}$ is a set of n_m memories. A processor p_i accesses a memory m_q through a physical link ϕ_j to get instructions and data. Two or more p_i can communicate either through a shared m_q (in this case, both p_i access the same m_q using the same ϕ_j), or through a point to point ϕ_j. p_i, ϕ_j, and m_q are characterized through performance parameters, such as instruction per cycle, energy performance, maximum workload, and maximum bandwidth. Fig. 3b shows an example of a system structural model.

The **system-level synthesis activity** comprises two steps: cost estimation and design space exploration. Cost estimation considers *FR* and *NFR* and estimates design metrics for implementing $\tau_r \in \Gamma$ as SW-tasks or HW-tasks [10, 16, 37], given the available $p_i \in P$, $\phi_j \in \Phi$, and $m_q \in M$.

Design space exploration involves HW/SW partitioning & mapping [52] to propose a candidate system structural model, followed by HW/SW co-simulation to verify the satisfaction of *FR* and *NFR*. If the requirements

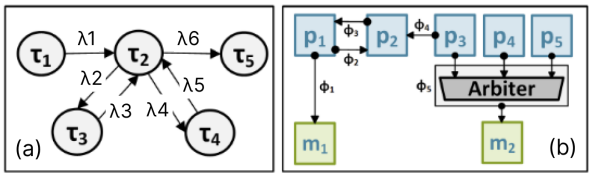


Fig. 3. Example of *FR* (a) and system structural model (b)

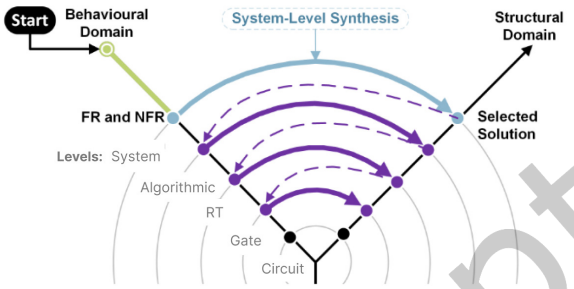


Fig. 4. Gajski-Kuhn chart with the HW/SW co-design flow.

are met, the candidate becomes the selected solution; otherwise, the process iterates with adjustments (the feedback loop ① in Fig. 2).

The **low-level implementation activity** covers the low-level HW/SW co-design, integration, and verification & validation of the selected solution. If the final system fails to meet the requirements, the flow will be feedback to one of the previous steps (in the worst case, to the Requirements Specification activity, as represented by the feedback loop ② in Fig. 2).

Customarily, at this point, **oCMSs integration** is initiated to address MREQs (for example, through code instrumentation). However, if, after the oCMSs integration, the monitored SoC fails to comply with *FR* and *NFR*, there is a potential need for a complete redesign and reimplementation of the SoC, represented by feedback loop ③ in Fig. 2. This potential need, that, accordingly to Fig.1 can led to an increase of the development time up to 65% (especially if some ASIC prototyping is involved), carries the inherent risk of missing the time-to-market deadline, a concern highlighted in the Introduction.

The discussed HW/SW co-design flow can be represented on the Gajski-Kuhn chart to emphasize the various levels of abstraction involved in system modeling. The Gajski-Kuhn chart [22] serves to describe the development of digital systems. It has also been employed to map HW/SW co-design flows across different abstraction levels (e.g., system level, algorithmic level) within three design domains: behavioral, structural, and physical (see Fig. 4). By retracing the HW/SW co-design flow over the Gajski-Kuhn chart, the output of requirements specification activity (i.e., *FR* and *NFR*) is identified by a point on the behavioral domain axes. Also, the system-level synthesis is positioned over the arc of the system-level circle, starting from the behavioral domain to the structural domain (blue arrow). Finally, the low-level implementation is positioned along several arcs, which start from the selected solution dot on the structural domain axes and end on the final implementation on the structural domain axes; in the general case, this happens through backward and forward moves between structural and behavioral domains (dashed and solid purple arrows).

4 The Proposed Monitoring Model

This section begins by analyzing the common way to describe MREQs and oCMSs, identifying the gaps in these descriptions that prevent the satisfaction of MREQs using oCMSs at the system-level (Subsection 4.1). Then, to overcome these gaps, a *monitoring model* is proposed (Subsection 4.2). Thanks to this relation, we will then be able to provide an approach that matches MREQs and oCMSs at the system-level, which will be described in Section 5.

4.1 The Necessity of a Monitoring Model

MREQs are typically elicited by designers from various domains such as automotive and avionics. Reviews [39, 51, 59] outcome that the structure of a single MREQ can be represented as:

$$\text{MREQ} = \langle \textit{Metrics, Intrusiveness, Portability} \rangle \tag{2}$$

where:

- **Metrics** – Monitoring is defined as the process of continuously or periodically observing a SoC and collecting relevant data. This is formalized by defining monitoring as the application of metrics to the system, each defined as a function that takes the system as input and returns a scalar output [59].
- **Portability** – The Portability refers to which architectures the monitoring actions is required to be operational [59].
- **Intrusiveness** – The Intrusiveness is the allowed impact for monitoring actions on processing resources [51].

Despite the clear structure of MREQs, to the best of our knowledge, the literature has predominantly focused on proposing methods to express metric requirements (e.g., [33, 48]), while the portability and intrusiveness dimensions are often left to be described informally in natural language. Furthermore, as highlighted in several works (e.g., [31, 59]), the portability dimension inherently depends on the system view adopted, either behavioral or structural, which influences how monitoring requirements are expressed and where they apply. However, no existing work provides designers with a formal means to specify both behavioral and structural MREQs within a unified model (see Section 2).

On the other side, there are oCMSs: oCMSs can be used to evaluate different metrics, targeting various SW components of a system such as application and operating system, or various HW components of a system such as physical links and processors [31]. Reviews [5, 15, 31, 32, 40, 54] provided a taxonomy of oCMSs properties. Analogously to MREQs, oCMSs metrics are well defined, while intrusiveness and portability are left to be expressed in natural language.

The absence of a standardized and formalized model to express the portability and intrusiveness of both MREQs and oCMSs severely impacts the possibility of matching MREQs with oCMSs at system-level. This motivates the need for a monitoring model that enables precise reasoning and decision-making early in the design process.

4.2 Parameters of the Monitoring Model

The monitoring model is composed of a set of parameters that serve to describe both MREQs and oCMSs at the system level. These parameters originate from the understanding of MREQs commonly adopted in the literature [39, 51, 59], where an MREQ is characterized by the triplet: ⟨*Metrics*, *Portability*, *Intrusiveness*⟩.

In this work, we adopt the key assumption that, *since oCMSs are designed to satisfy MREQs, their properties can be described using the same triplet structure.*

Building on this assumption and based on state-of-art works about oCMSs (for an exhaustive list, please refer to systematic reviews in [5, 15, 31, 32, 40, 54]), we identified the parameters needed to describe each of the three dimensions *Metrics*, *Portability*, and *Intrusiveness* in a precise way. These parameters are used not only to

characterize oCMSs, but also to express MREQs in a formal manner at system-level. In essence, the monitoring model proposed in this work is the set of such parameters.

4.2.1 METRICS. In the proposed model, we describe the metrics that an oCMS allows to evaluate through four strings:

$$Metrics = \omega \in \Omega$$
$$\Omega = \{cache, bandwidth, timestamp, power\}$$

where:

- *cache* refers to monitoring the number of cache misses, meaning that the oCMS counts how many times a p_i attempts to retrieve data from cache memory but finds the requested item absent. Cache misses are typically captured through oCMSs comprising dedicated HW counters integrated into processor architectures, as such events are not directly visible to SW. These oCMSs are widely available in commercial processors (e.g., Intel [26], ARM [4]) and have also been proposed in academic works [32].
- *bandwidth* refers to monitoring the data throughput, i.e., the total number of bytes read or written by a p_i, on an m_q, or across a ϕ_j, along with the associated elapsed time for the transfer. Bandwidth monitoring can be implemented in SW or HW. SW-based oCMSs typically use routines to track transmitted/received data and elapsed time [7, 64]. HW-based oCMSs rely on performance counters to measure byte transfer counts and time intervals, exposing results to the processor or runtime manager [56].
- *timestamp* refers to assigning time references to events occurring on p_i, ϕ_j, or m_q. Timestamps can be obtained using SW-based oCMSs (e.g., by reading processor timers to tag events or execution regions [50]), or using dedicated HW timestamping units embedded in trace/debug components such as ARM ETM [2].
- *power* refers to estimating or measuring the power dissipation of a p_i. Power monitoring can be implemented either through analog mechanisms (e.g., on-chip voltage/current sensors [31]), or all-digital methods, where power is inferred through a model that takes as input the number of transitions or access events happening across the HW architecture and estimates power [65]. Such all-digital monitoring may be performed by SW routines or through dedicated HW counters.

In this work, we selected these four metrics based on their ability to cover distinct and relevant functional objectives of monitoring, as discussed in existing surveys of oCMSs [5, 15, 32, 40, 54], and formalized in the functional taxonomy proposed in [31]. Specifically, [31] identifies seven functional objectives for oCMSs: performance, quality-of-service, energy/power/thermal management, debugging, fault tolerance, security, and application-specific monitoring. The selected metrics have been chosen to represent four of these key categories: *cache* is a canonical representative of performance monitoring [5, 31, 32, 54]; *bandwidth* corresponds to quality-of-service monitoring [5, 31]; *power* directly supports energy and thermal management strategies [31]; and *timestamp* enables system-level observability and debugging [32, 40], allowing temporal correlation of events and runtime traceability. Notably, reliability and security are also among the identified functional objectives; however, at the monitoring level, they typically rely on similar information as the aforementioned categories. Thus, the selected metrics contribute indirectly to these objectives as well. This selection ensures broad coverage across the monitoring design space while maintaining compatibility with existing HW/SW implementation approaches.

It is worth noting that more metrics can be added to Ω (e.g., number of translation look-aside buffer misses), so as to not limit the scope of the monitoring model and to involve all the existing oCMSs [31]. However, even if the monitored metrics differ, they typically map to one or more of the functional objectives defined in [31], preserving the generality and extensibility of the proposed monitoring model.

4.2.2 PORTABILITY. The proposed model allows expressing the portability of an oCMS (i.e., on which architectures it is able to perform a monitoring action) through processing units p_i or physical links ϕ_j (p_i and ϕ_j are

Table 3. oCMS and expressed using the monitoring model parameters.

	Behavioral MREQs		Structural MREQs	oCMS
	ibmreq	**ebmreq**	**smreq**	
Metrics	$\mu\tau_f, \mu\lambda_s$	ω	ω	ω
Portability	τ_r, λ_t	τ_r, λ_t	p_i, ϕ_j	p_i, ϕ_j
Intrusiveness	$[pd, area, res, pwd, energy]$	$[pd, area, res, pwd, energy]$	$[pd, area, res, pwd, energy]$	$[pd, area, res, pwd, energy]$

HW components typically targeted by a monitoring action [31]). Also, the model allows expressing the type of physical target where the oCMS can be implemented:

$$Portability = p_i, \phi_j$$
$$p_i = [pt, isa, ic, dc, tpi] \quad \phi_j = [ct, prot, tpi]$$

where:

- a p_i is described with a processor type $pt \in \{ gpp, spp\}$, where $pt = gpp$ if the p_i is a general-purpose processor; $pt = spp$ if the p_i is a single-purpose processor [55]. When $pt = gpp$, the general-purpose processor has an associated instruction-set $isa \in \{armv7, x86, ...\}$. When $pt = gpp$, the general-purpose processor can have an instruction cache $ic \in \{true, false\}$ and a data cache $dc \in \{true, false\}$;
- ϕ_j is described with a communication type $ct \in \{bus, p2p\}$, where $ct = bus$ means that the ϕ_j is a bus able to connect multiple masters and multiple slaves, while $ct = p2p$ means that the ϕ_j is a point-to-point connection between one master and one slave. The ϕ_j has an associated protocol $prot \in \{uart, i2c, AXI4, ...\}$;
- $tpi \in \{rcf, fx\}$ refers to the type of physical target, where $tpi = fx$ means that the target technology is completely fixed (i.e., Application-Specific Integrated Circuits - ASIC), while $tpi = rcf$ means that the target technology is reconfigurable (e.g., FPGA).

4.2.3 INTRUSIVENESS. The intrusiveness of an oCMS (i.e., the impact of the oCMS on system resources) is expressed as:

$$Intrusiveness = [pd, area, res, pwd, energy]$$

where: $pd \in \mathbb{N}$ describes the maximum impact in timing overhead when using the oCMS, in number of clock cycles; $area \in \mathbb{R}$ is the area occupied by the oCMS, expressed in squared millimeters and valid when $tpi = fx$; $res = \{lut \in \mathbb{N}, ff \in \mathbb{N}, dsp \in \mathbb{N}, bram \in \mathbb{N}\}$ are the resources occupied by the oCMS, valid when $tpi = rcf$; $pwd \in \mathbb{R}$ is the power dissipated by the oCMS; $energy \in \mathbb{R}$ is the energy consumed by the oCMS.

4.3 Modeling the On-Chip Monitoring Systems and the Monitoring Requirements

On-Chip Monitoring Systems. To describe an oCMS through the monitoring model, we use the parameters as indicated in Table 3. *Metrics* can contain multiple values, as the same oCMS can collect more than one metric.

When using multiple instances of the same oCMS, its intrusiveness typically grows. Eventually, by increasing the number of instances of an oCMS, it can happen that not all the associated parameters grow. For this reason, the *Intrusiveness* parameters of an oCMS can be indicated with an asterisk, meaning that the parameter will not increase when growing the number of instances. For instance, this applies to oCMSs implemented as HW components (e.g., [61]), whose intrusiveness in terms of *pd* remains constant regardless of the increasing number.

Monitoring Requirements. To express MREQs through the proposed monitoring model, we consider that MREQs are specified as ⟨*Metrics, Intrusiveness, Portability*⟩. Furthermore, as emerged from the literature, the portability

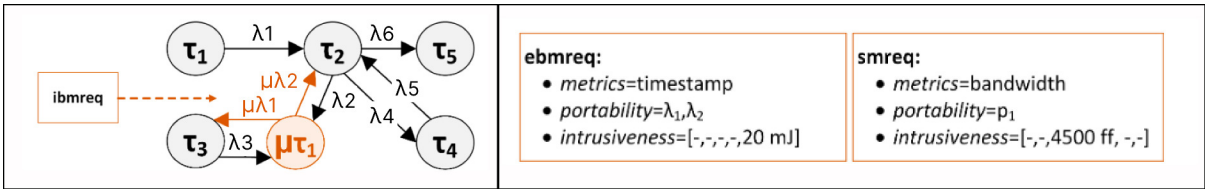


Fig. 5. Example of *ibmreq*, *ebmreq*, and *smreq*.

dimension can refer to either a behavioral view or a structural view of the system. Based on these premises, the proposed monitoring model supports the following classification of MREQs:

- **Behavioral MREQs**
  - **Explicit behavioral monitoring requirements (ebmreqs)** — When a designer requires a monitoring action using predefined metrics $\omega \in \Omega$ that target system behavior, the MREQ is expressed as an *ebmreq*. An *ebmreq* enables the specification of a monitoring action that observes system behavior, collects known metrics, and adheres to a given impact constraint.
    The elements ⟨*Metrics*, *Portability*, *Intrusiveness*⟩ are expressed as listed in Table 3.
    For example, an *ebmreq* can be: "Get the timestamp related to the communication occurring on specific logical links, with an energy budget not exceeding 20 mJ" (see Fig. 5).
  - **Implicit behavioral monitoring requirements (ibmreqs)** — Not all metrics required by designers can be predefined . When a custom metric is needed, the MREQ is captured as an *ibmreq*. In this case, the *Metrics* element is specified using the same formalism adopted for *FR*:

$$Metrics = (\Gamma_M, \Lambda_M) \tag{3}$$

    where $\Gamma_M = \{\mu\tau_1, \ldots, \mu\tau_f, \ldots, \mu\tau_{n_{\mu\tau}}\}$ is the set of $n_{\mu\tau}$ monitoring tasks, and $\Lambda_M = \{\mu\lambda_1, \ldots, \mu\lambda_s, \ldots, \mu\lambda_{n_{\mu\lambda}}\}$ is the set of $n_{\mu\lambda}$ monitoring links.
    The *Portability* and *Intrusiveness* elements are expressed using the parameters defined in Table 3.
    Fig. 5 shows an example of an *ibmreq* requesting a custom monitoring action to count the number of messages exchanged between τ_2 and τ_3. The custom metric is implemented by introducing one $\mu\tau_f$ and two $\mu\lambda_s$. An *ibmreq* is particularly useful when the designer needs to specify a domain-specific metric that cannot be derived from the predefined ones, such as detecting a specific communication pattern between behavioral elements.
- **Structural MREQs (smreqs)** — An *smreq* captures a monitoring requirement that observes a structural component of the system (e.g., processors, physical links), collecting predefined metrics with bounded impact.
  The elements ⟨*Metrics*, *Portability*, *Intrusiveness*⟩ are expressed as listed in Table 3.
  For example, an *smreq* could be: "Measure the bandwidth usage of a specific processor, using at most 4500 flip-flops on the target architecture" (see Fig. 5).

In conclusion, MREQs are expressed through three sets: *IBMREQ* (set of *ibmreqs*), *EBMREQ* (set of *embreqs*), and *SMREQ* (set of *smreqs*).

5 The Proposed Design Approach

In this section, we first introduce our proposed design approach (Subsection 5.1), which begins by capturing all requirements at the system level (i.e., *FR*, *NFR*, *IBMREQ*, *EBMREQ*, and *SMREQ*) and, by leveraging the monitoring

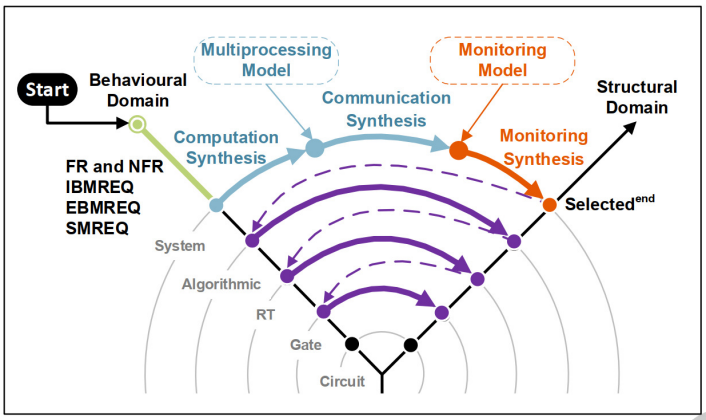


Fig. 6. The proposed approach represented on the Gajski-Kuhn chart.

model, enables a comprehensive HW/SW co-design of monitored SoCs. Next, we describe the various monitoring configurations that the proposed approach is capable of modeling and supporting (Subsection 5.2).

5.1 Description of the Approach

The proposed approach is depicted in Fig. 6 through the Gajski-Kuhn chart. Initially, all requirements are captured. Subsequently, a system-level synthesis is conducted to produce a structural model incorporating oCMSs (over processors p_i, physical links ϕ_j, and memories m_q). This synthesis is divided into three moves from the behavioral to the structural domain, represented as *computation synthesis* (blue arrow), *communication synthesis* (blue arrow), and *monitoring synthesis* (orange arrow) in Fig. 6. As detailed in Section 3, these moves on the Gajski-Kuhn chart can be elaborated into steps on a flowchart.

Fig. 7a reports a flowchart of the proposed approach. After the requirements specification activity, system-level synthesis includes cost estimation and design-space exploration steps. Consequently, feedback loop ① remains unchanged between Fig. 2 and Fig. 7a. After the design space exploration step, a new step, referred to as *design space exploration for reuse*, is performed. This step aims to satisfy the MREQs with oCMSs already during system-level synthesis (rather than at the end of the low-level implementation activity, as happens in Fig. 2). Both design-space exploration and design-space exploration for reuse work to satisfy all input requirements before low-level implementation activity, ensuring alignment with aspect (ii) outlined in the Introduction. Additionally, the design space exploration for reuse step relies entirely on existing oCMSs, addressing aspect (iii). Following system-level synthesis activity, a structural model of a monitored SoC, compliant with all initial requirements, is generated.

In case there is no solution, two new feedback loops emerge: the feedback loop ④ allows to perform a new design space exploration, while the feedback loop ⑤ allows adjusting the requirements. These new feedback loops are less time-consuming compared to the old feedback loop ③ in Fig. 2, since they do not involve Low-level implementation at all. Overall, the proposed approach early prevents requirement violations due to oCMSs integration at the end of Low-level implementation.

The next sections provide more details about the proposed approach.

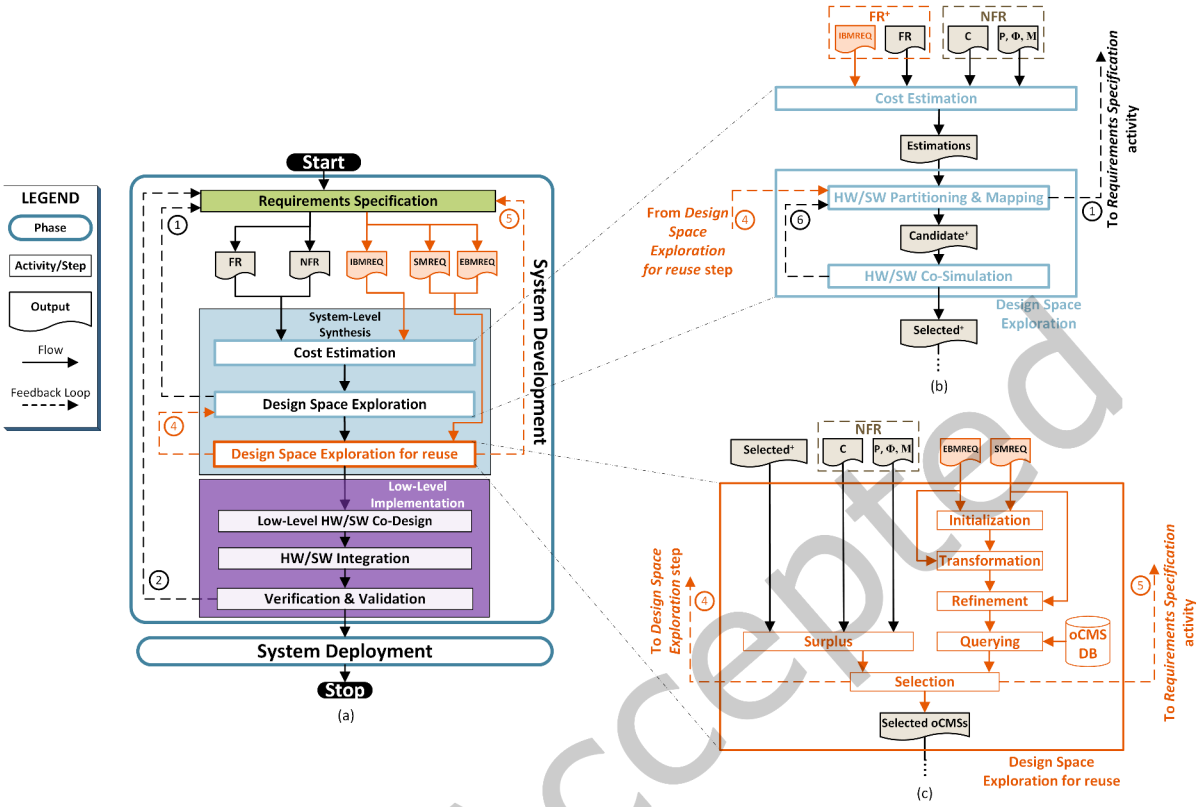


Fig. 7. The proposed approach represented as a flowchart (a), with the design space exploration (b) and design space exploration for reuse (c) steps detailed.

5.1.1 ***Requirements specification.*** In the requirements specification activity, designers provide *FR*, *NFR*, *IBMREQ*, *EBMREQ*, and *SMREQ* (see Fig. 7a). *IBMREQ* is provided using $\mu\tau_f \in \Gamma_M$ and $\mu\lambda_s \in \Lambda_M$, while *EBMREQ* and *SMREQ* are provided as annotations (e.g., through a text file).

Being *IBMREQ* expressed with the same formalism of *FR*, they can be merged to obtain FR^+:

$$FR^+ = (\Gamma^+, \Lambda^+) \quad \Gamma^+ = \Gamma \cup \Gamma_M \quad \Lambda^+ = \Lambda \cup \Lambda_M$$

The adopted formalism to express the requirements, based on tasks τ_r and logical links λ_t (see Section 3), as well as monitoring tasks $\mu\tau_f$ and monitoring links $\mu\lambda_s$, has been chosen for its generality and modularity, allowing system-level specification of functional and monitoring behavior in a platform-independent manner. The adopted formalism does not preclude the use of alternative or complementary specification techniques. To this end, it remains compatible with established formal methods such as temporal logic [44] and timed automata [1]. Indeed, our model can be regarded as complementary to these methods, and appropriate transformations can be defined to focus on specific properties (e.g., [12]).

5.1.2 ***System-level Synthesis.*** In the system-level synthesis activity, three steps take place: cost estimation, design space exploration, and design space exploration for reuse (see Fig. 7a).

Cost Estimation and Design Space Exploration. - The cost estimation and design space exploration steps are detailed in Fig. 7b. The cost estimation step is carried out using FR^+ and *NFR* as inputs. The specific method used to estimate design metrics does not affect the proposed approach and depends on the HW/SW co-design tool adopted.

The design space exploration step takes as input FR^+, *NFR*, and the output of the cost estimation step, and produces as output the selected solution, referred to as Selected$^+$ (see Fig. 7b). Selected$^+$ is a structural model that satisfies *FR*, *NFR*, and *IBMREQ*. The design space exploration step consists of two substeps: HW/SW partitioning & mapping and HW/SW co-simulation. HW/SW partitioning & mapping [52] is typically based on heuristic approaches (e.g., genetic algorithms [28]), and is used to propose a candidate structural model. The comparison between the proposed approach depicted in the flowchart and its representation on the Gajski-Kuhn chart allows for a clearer understanding of the intermediate model representations and abstraction levels involved in the process. Within the HW/SW partitioning & mapping substep, two functions are applied:

(1) $f^{comp} : \Gamma^+ \rightarrow P$ - This function assigns each $\tau_r \in \Gamma^+$ and $\mu\tau_f \in \Gamma^+$ to one of the processing units $p_i \in P$ (i.e., $f^{comp}(\tau_r) = p_i$). The f^{comp} function corresponds to the computation synthesis arrow in Fig. 6. Its application yields a representation in terms of a multiprocessing model, specifically a mixed behavioral/structural description that outlines the computation structure while retaining communication at a behavioral level.
(2) $f^{comm} : \Lambda^+ \rightarrow \Phi$ - This function assigns each $\lambda_i \in \Lambda^+$ and $\mu\lambda_s \in \Lambda^+$ to one of the $\phi_j \in \Phi$ (i.e., $f^{comm}(\lambda_i) = \phi_j$). The f^{comm} function corresponds to the communication synthesis arrow in Fig. 6. Its application yields a representation in terms of the monitoring model (outlined in Section 4).

At the end of HW/SW partitioning & mapping, there will be a candidate solution (Candidate$^+$, see Fig. 7b). Then, the HW/SW co-simulation allows to obtain the associated metrics $metrics^+ = \{rt^+, area^+, res^+, pwd^+, energy^+\}$ and to verify the satisfaction of FR^+ and *NFR*: if they are satisfied, the Selected$^+$ is obtained as a result. HW/SW co-simulation is typically performed using event-driven simulation engines (e.g., SystemC). If FR^+ and *NFR* are not satisfied, designers can go back to the requirements specification activity (feedback loop ① in Fig. 7b) or to HW/SW partitioning & mapping substep (feedback loop ⑥ in Fig. 7b). The Selected$^+$ is a structural model that satisfies *FR*, *NFR*, and *IBMREQ* and is composed of interconnected components part of set $P^* \subseteq P$, $\Phi^* \subseteq \Phi$, and $M^* \subseteq M$, with cardinality n_{P^*}, n_{Φ^*}, and n_{M^*}, respectively.

It is worth noting that the proposed approach does not impose any specific method for performing cost estimation and design space exploration, nor does it depend on the choice of underlying algorithms. What is essential is that the design space exploration process operates on inputs expressed using the formalism of FR^+ and *NFR*. In Section 6, a HW/SW co-design tool is selected to instantiate the proposed approach, and the implementation details of the cost estimation and design space exploration steps are provided in that context.

Design space exploration for reuse. - Fig. 7c shows the flowchart of the design space exploration for reuse. This step takes as input *EBMREQ*, *SMREQ*, the Selected$^+$, and the *NFR*, and produces as output a set of oCMSs that satisfy *EBMREQ*, *SMREQ*, and *NFR* (hence able to collect the required $\omega \in \Omega$ metrics on the Selected$^+$).

The design space exploration for reuse corresponds to the monitoring synthesis arrow in Fig. 6 and comprises several substeps (see Fig. 7c). Specifically, the Initialization, Transformation, and Refinement substeps convert the *EBMREQ* and *SMREQ* (provided by designers) into a specific intermediate representation that enables:

- identifying which components of Selected$^+$ are required to be monitored;
- counting how many times *EBMREQ* and *SMREQ* collectively request the monitoring of a given metric on a specific component of Selected$^+$. This count is referred to as the *frequency of involvement*.

Based on this intermediate representation, the Querying, Surplus, and Selection substeps:

- perform multiple queries on a database[1] to identify oCMSs capable of monitoring the required components of Selected$^+$, referred to as candidate oCMSs. The database (represented as oCMS DB in Fig. 7c) is created with MongoDB[2] and contains N^{DB} existing oCMSs. It serves as a repository of oCMSs sourced from literature, with each oCMS described using monitoring model parameters and indexed with an incremental number. At the time of writing, the oCMSs database contains $N^{DB} = 30$ oCMSs. Notably, the database continues to grow as more oCMSs from literature are added.
- Scale the intrusiveness of each candidate oCMS using the associated frequency of involvement.
- Select combinations of oCMSs that satisfy the *NFR* constraints.

The next paragraphs detail each substep.

Initialization - The initialization substep takes as input *EBMREQ* and *SMREQ* and, for each ω expressed in *EBMREQ* and *SMREQ*, produces a pair of matrices Q^ω and $oCMS^\omega$ with all elements initialized to 0:

$$Q^\omega = \left[f_{p1}, f_{p2}, \ldots, f_{\phi 1}, f_{\phi 2}, \ldots\right]_{1\times M}$$
$$oCMS^\omega = \left[\frac{p-block}{\phi-block}\right]_{M\times N^{DB}}$$

where $M = n_{P^*} + n_{\Phi^*}$.

Q^ω is used to initialize the frequency of involvement, introduced earlier, which is computed by the Transformation and Refinement substeps. The frequency of involvement is a scalar value associated with a metric ω and a platform component in Selected$^+$. It is denoted as f_{p_i} (or f_{ϕ_j}) and indicates how many times a component of the Selected$^+$ ($p_i \in P^*$ or $\phi_j \in \Phi^*$) is required to be monitored to extract the metric ω. Q^ω has one row and a number of columns equal to the sum of n_{P^*} and n_{Φ^*}. The columns correspond to the components included in Selected$^+$, starting with $p_i \in P^*$ from the 1$^{\text{st}}$ to the $n_{P^*}{}^{\text{th}}$ column, followed by $\phi_j \in \Phi^*$ from the $(n_{P^*}+1)^{\text{th}}$ to the M^{th} column. The single row of Q^ω contains the frequency of involvement of each component with respect to the metric ω.

$oCMS^\omega$ has the goal of collecting which oCMSs, among the N^{DB} in the database, can monitor the platform components part of Selected$^+$. $oCMS^\omega$ is a block matrix, composed of two blocks: the p-block and the ϕ-block. The p-block is the top part of $oCMS^\omega$, with n_{P^*} rows and N^{DB} columns, while the ϕ-block is the bottom part of $oCMS^\omega$, with n_{Φ^*} rows and N^{DB} columns. $oCMS^\omega$ contains Boolean values. When one of the N^{DB} oCMSs parts of the database satisfies a monitoring requirement referred to $p_i \in P^*$ or $\phi_j \in \Phi^*$, the corresponding element inside $oCMS^\omega$ is set to 1. Filling $oCMS^\omega$ is the role of Querying substep (see next paragraphs).

Transformation and **Refinement** - The transformation and refinement substeps take as input *EBMREQ* and *SMREQ* and update the values f_{p_i} and f_{ϕ_j} inside Q^ω. During the transformation substep, the *Portability* of *EBMREQ* is transformed from the behavioral domain to the structural domain by transforming requirements on τ_r and λ_t on requirements on p_i and ϕ_j, also incrementing f_p and f_ϕ. During the refinement substep, the *Portability* of *SMREQ* is analyzed and used to increment f_p and f_ϕ. Both substeps perform different actions depending on the metric ω. The pseudo-code of the transformation and refinement substeps are reported in the external Appendix[3] for reader convenience.

Querying - The querying substep takes as input the Q^ω, the oCMSs database, and the $oCMS^\omega$, and provides as output the $oCMS^\omega$ matrices opportunely updated. For each Q^ω, the querying substep performs a query on the database for each of the columns containing $\mathrm{f}_p \neq 0$ or $\mathrm{f}_\phi \neq 0$: the oCMSs resulting from each query are then set to 1 inside the corresponding $oCMS^\omega$. Once the $oCMS^\omega$ is available, for all the oCMSs for which $oCMS^\omega(x,y) \neq 0$, with x=1,..., M and y=1,...,N^{DB}, the associated monitoring model parameters are multiplied by the value of the

[1] https://monicatool.cloud
[2] MongoDB: https://www.mongodb.com
[3] https://tinyurl.com/externalAppendix

corresponding f_p or f_ϕ (which are stored in Q^ω) to properly scale the intrusiveness. The pseudo-code of the querying substep is reported in the external Appendix[3] for reader convenience.

Surplus - The surplus substep takes as input the Selected$^+$ and the *NFR*, and produces as output a *budget* by subtracting the impact of Selected$^+$ from the constraint set *C* within the *NFR*. This budget represents the available margin for selecting additional oCMSs to be included in the final solution, without violating the *NFR*.
The Selected$^+$ has associated design metrics $metrics^+$. The difference between the input *NFR* and $metrics^+$ is the available budget:

$$budget = \{\Delta rt, \Delta area, \Delta res, \Delta pwd, \Delta energy\} \tag{4}$$

where: Δrt = *C(rt)* - rt^+, $\Delta area$ = *C(area)* - $area^+$, Δres = *C(res)* - res^+, Δpwd = *C(pwd)* - pwd^+, and $\Delta energy$ = *C(energy)* - $energy^+$.

Selection - The selection substep takes $oCMS^\omega$ and the budget as input and selects one or more solutions able to satisfy *EBMREQ* and *SMREQ* while not overcoming the budget (hence respecting FR^+ and *NFR*). In the proposed approach, the selection substep performs an *exhaustive search* among all the combinations of oCMSs, and it picks the ones respecting the budget. The pseudo-code of the selection substep is reported in the external Appendix[3] for reader convenience. The selection substep produces solutions composed of set of oCMSs that satisfy *EBMREQ* and *SMREQ* without invalidating FR^+ and *NFR*.

Final Steps - At the end of the proposed approach, designers can decide which solution to pick and include in the Selected$^+$, to finally obtain the Selected^{end}. The Selected^{end} is composed of interconnected components part of set $P^* \subseteq P$, $\Phi^* \subseteq \Phi$, and $M^* \subseteq M$, plus oCMSs.

From Selected^{end}, it is possible to proceed with the low-level implementation activity (see Fig. 7a). If the design space exploration for reuse fails to yield a solution, or if the obtained solutions are unsatisfactory for designers, there are three possible actions: (i) relaxing requirements (feedback loop ⑤ in Fig. 7c), (ii) reinitiating the design space exploration (feedback loop ④ in Fig. 7c), or (iii) enriching the database with additional oCMSs. The feedback loops ④ and ⑤ are less time-consuming compared to the feedback loop ③ in Fig. 2.

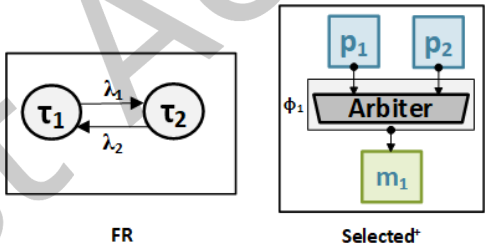


Fig. 8. The proposed approach represented on the Gajski-Kuhn chart.

Design Space Exploration for Reuse: A Conceptual Example - We provide a conceptual example to show how the design space exploration for reuse step operates. While Section 6 presents the application of the entire proposed approach, this example is intended to clarify the specific connection between *EBMREQ*, *SMREQ*, and the frequency of involvment parameters f_{p_i} and f_{ϕ_j}.

Suppose that the designers elicit *FR* as shown in Fig. 8, comprising two tasks, τ_1 and τ_2, which communicate via two logical links, λ_1 and λ_2. Additionally, assume that three MREQs are defined as inputs to the flow shown in Fig. 7a:

- $ebmreq_1$ targets the monitoring metric ω = *cache*, with a portability requirement referring to τ_1.
- $ebmreq_2$ also targets the same monitoring metric ω = *cache*, but its portability requirement refers to τ_2.

- $ebmreq_3$ targets the monitoring metric $\omega = bandwidth$, with a portability requirement referring to λ_2.

Assume that the design space exploration step produces a Selected$^+$ platform composed of two processors p_1 and p_2, a bus ϕ_1, and a memory m_1, as shown in Fig. 8. In the resulting mapping, τ_1 is allocated to p_1, τ_2 to p_2, and both λ_1 and λ_2 are mapped onto ϕ_1, with shared access to m_1. Suppose also that an additional MREQ is provided:

- $smreq_1$ targets the monitoring metric $\omega = power$, with a portability requirement referring to p_1.

In the design space exploration for reuse step, the matrices Q^{cache}, $Q^{bandwidth}$, and Q^{power} are created. When the Transformation and Refinement substeps are applied, $ebmreq_1$ and $ebmreq_2$ are translated into monitoring actions targeting the cache of p_1 and p_2, respectively; $ebmreq_3$ is translated into a monitoring action targeting the bandwidth of ϕ_1; and $smreq_1$ results in a monitoring action targeting the power of p_1. As a result, the matrix Q^{cache} is updated with $f_{p_1} = 1$ and $f_{p_2} = 1$, indicating that the initial MREQs ultimately translate into one monitoring action for *cache* on both p_1 and p_2. Similarly, $Q^{bandwidth}$ is updated with $f_{\phi_1} = 1$, and Q^{power} is updated with $f_{p_1} = 1$. If an additional requirement $smreq_2$ were defined with $\omega = cache$ and targeted p_2, then f_{p_2} in Q^{cache} would be incremented to 2.

Subsequently, the Querying substep searches the database for oCMSs capable of monitoring the *cache*, *bandwidth*, and *power* metrics on the respective components. This results in a set of candidate oCMSs, each characterized by a certain level of intrusiveness, as described by the monitoring model. For each candidate oCMS, its intrusiveness is scaled according to the corresponding frequency of involvement.

In the final Selection substep, combinations of oCMSs are evaluated and selected based on their ability to satisfy the defined MREQs while complying with the constraints specified in the *NFR*.

5.2 Review of Monitoring Configurations Supported in the Proposed Approach

This subsection summarizes the types of monitoring configurations that can be handled by the proposed approach:

- **Which type of monitoring implementation**. The proposed approach supports monitoring configurations in which oCMSs are implemented either in HW or in SW. Specifically, when the input includes *IBMREQ*, the approach determines whether the oCMS should be implemented in HW or SW based on the constraint set C defined within the NFR. In contrast, when the input includes *EBMREQ* or *SMREQ*, the proposed approach selects from a set of existing oCMSs, whose implementation (HW or SW) is determined by their original developers. The choice of the appropriate oCMS is guided by the portability and intrusiveness requirements, and it is performed during the design space exploration for reuse step, as described in Section 5.1 and in the external Appendix[3].
- **Which type of metric extraction**. The proposed approach does not explicitly address the internal implementation details of how metrics are extracted. In the case of *IBMREQ*, the monitoring functionality is modeled through $\mu\tau$ and $\mu\lambda$, which are treated as tasks and logical links mapped onto processors p_i or interconnects ϕ_j. Metric extraction, in this case, is implicitly determined by how the underlying processor executes the corresponding task and captures the data. For *EBMREQ* and *SMREQ*, where the approach reuses existing oCMSs, the handling of metric extraction depends on the specific oCMS. If the existing oCMS includes this analysis, as in the case of the solution proposed by Lee et al [32], then metric extraction is inherently addressed. Otherwise, the extraction process remains unspecified and outside the scope of the proposed approach.
- **Intrusiveness of the monitoring**. The level of monitoring intrusiveness is explicitly considered in the proposed approach and is captured within the MREQs. The intrusiveness level influences both the implementation strategy (for *IBMREQ*) and the selection of existing oCMSs (for *EBMREQ*/*SMREQ*), ensuring that the final monitored system adheres to the specified constraints.
- **Which components are subject to monitoring**. Regardless of the type of MREQ provided as input, the proposed approach ultimately introduces oCMSs at the granularity of $p_i \in P$, $\phi_j \in \Phi$, and $m_q \in M$. The

way in which oCMSs are associated with these components depends on the MREQ type. For *IBMREQs*, the monitor is implemented as a task executed on a processor p_i. In contrast, for *EBMREQs* and *SMREQs*, the connection of the oCMS to system components is determined by the design of the reused oCMS itself.

6 Experimental Activities

The proposed experimental activities aim to evaluate the effectiveness of the proposed approach in addressing the three key aspects (i), (ii), and (iii) outlined in the Introduction. This section presents two experimental activities:

- We report on the design of a monitored SoC executing a synthetic application (Subsection 6.2), referred to as monitored FIR-FIR-GCD (FFG). The objective is to demonstrate the capability of the proposed approach to capture and satisfy, at system level, both behavioral MREQs and structural MREQs – addressing aspect (i). In addition, this experiment shows how the approach supports the reuse of existing oCMSs – addressing aspect (iii). For comparison, we also apply the customary approach to satisfy MREQs, in order to compare development time issues with respect to the proposed approach.
- We then consider a real-world application (Subsection 6.3), specifically the design of a monitored SoC executing a pacemaker control application. The pacemaker is widely recognized in the literature for its relevance to system reliability, which elicits diverse MREQs (see, e.g., [46] [47] [27]). This experiment serves to demonstrate how the proposed approach enables a design-space exploration to trade off between response time and resource utilization – addressing aspect (ii).

After detailing the experimental settings (Section 6.1), Subsections 6.2 and 6.3 describe the two experimental activities. Finally, Section 6.4 provides a discussion about the scalability of the proposed approach with respect to the size of the systems under design. All results are produced using the algorithms described in Section 5, with pseudocode provided in the external Appendix[3].

6.1 Experimental Settings

To implement the proposed approach, we extended an existing HW/SW co-design tool. Among the ones compliant with the flow reported in Fig. 2 (e.g., CoFluent [4], SpaceStudio [5], HEPSYCODE [6], ForSyDe [7], SystemCoDesigner [52], Metropolis), we selected HEPSYCODE [10, 36, 45] for its capabilities in HW/SW partitioning & mapping. HEPSYCODE autonomously constructs HW architectures using P, Φ, and M, and suggests mappings automatically. The fact to not be limited to a predetermined target architecture input by the designer enables the full exploitation of the potential of the proposed approach.

In the requirements specification activity in HEPSYCODE, *FR* are modeled through a communicating sequential processes model of computation [25], coded in SystemC, and constraints are provided through metrics as in Eq. (1). The technology library file specifies the parameters for Φ, M, and P. During system-level synthesis, in the cost estimation step, HEPSYCODE estimates the affinity of each task $\tau_r \in \Gamma$ with each processing unit $p_i \in P$, the level of concurrency achievable in executing τ_r, and the computational load incurred when executing a specific τ_r on p_i.

In the design space exploration step, HEPSYCODE uses these metrics to perform multi-objective optimization, finding sub-optimal solutions to satisfy requirements thanks to a genetic algorithm. HW/SW co-simulation of candidate solutions provides response time and energy metrics.

[4]Intel CoFluent Technology: https://www.intel.com
[5]SpaceStudio by SpaceCoDesign: https://www.spacecodesign.com
[6]HEPSYCODE Tool: https://hepsycode.github.io/
[7]ForSyDe: https://forsyde.github.io/

We modified HEPSYCODE (i) to provide *IBMREQ* in SystemC to obtain the FR^+, plus *EBMREQ* and *SMREQ* in a dedicated file, and (ii) to manage *IBMREQ* in design space exploration and implement all the substeps of design space exploration for reuse.

In both experimental activities, we targeted an FPGA, specifically the Xilinx Artix7 *xc7a35t* (with total *res = [20800 lut, 41600 ff, 90 dsp, 50 bram]*), using the Vivado 2017.4 development environment [62]. The choice of an FPGA facilitates rapid prototyping and fully exploits the potential of the proposed approach. Although HEPSYCODE can target ASICs and complex SoCs like AMD/Xilinx Zynq Ultrascale+ [63], these were not used to maintain maximum flexibility for experimental evaluation. The host computer for the tests was equipped with an Intel Xeon CPU E3-1225 v5 at 3.3 GHz and 32 GB of memory.

In both experimental activities, we used the same sets of P, Φ, and M:

- **Processors (P)**: Table 4 reports the types of p_i. They include the Intel 8051 microcontroller [41], the Gaisler Leon3 microprocessor [21], and several single-purpose processors generated via the Bambu high-level synthesis tool [20]. It is worth noting that these choices were influenced by the availability of timing and energy metrics (CC4CS [37] and J4CS [35]) for these processors in HEPSYCODE. Up to two instances of Intel 8051 and Leon3 were used.
- **Physical Links (Φ)**: They include Intel 8051 custom bus [41], AMBA AXI4 [3], and GPIO links. There was no limit on the number of ϕ_j instances used.
- **Memories (M)**: They include on-chip RAM with no limit on the number of m_q instances used.

Table 4. p_i made available to HW/SW co-design to implement the two experimental activities. Timing performance is related to CC4CS [37] (clock cycles/C statement). Energy consumption is related to J4CS [35] (micro Joule/C statement).

p_i type	Reference	Timing Performance (CC4CS)	Energy Performance (J4CS)	*res* [lut, ff, dsp, bram]
[GPP,i8051,f,f,rcf]	Intel 8051 [41]	126-294	0.3-14.7	2321, 1347, 0, 0.5
[GPP,sparcv8,t,t,rcf]	Gaisler Leon3 [21]	91-585	0.606-3.9	9042, 4973, 4, 16.5
[SPP,0,f,f,rcf]	Bambu [20]	1-3	0.015-0.03	-

6.2 Design of a Monitored System-on-Chip for a Synthetic Application

This use case presents the design of a monitored SoC executing a synthetic application, referred to as monitored FFG. It shows how the proposed approach enables the specification and fulfillment of both behavioral MREQs and structural MREQs – addressing aspect (i) from the Introduction. Moreover, it shows the capability of the approach to support the *reuse* of existing oCMSs – addressing aspect (iii). In addition, this use case highlights the reduction in development time achieved when compared to the customary approach for addressing the same MREQs. To support reproducibility, the complete code and configuration related to this use case are made available in an open-source GitHub repository[8]. This section first describes the requirements specification activity, then presents the solution to the proposed problem using the proposed approach, and finally compares, in terms of development time, the solutions obtained with the customary and proposed approach.

6.2.1 ***Requirements Specification.*** The FFG processes 10 pairs of integer values to conduct two filtering operations, calculates the greatest common divisor of the filter outputs, and displays the results. The *FR* is shown in Fig. 9 (left side, grey part) and includes 8 τ_r and 15 λ_t, with additional τ_r for stimulus and display. It is worth noting that, in this use case, any input sequence triggers execution across all $\tau_r \in \Gamma$ (i.e., 100% model coverage). Therefore, the specific details of the input data are not particularly relevant to the discussion.

[8]https://github.com/alkalir/DesignForMonitorability

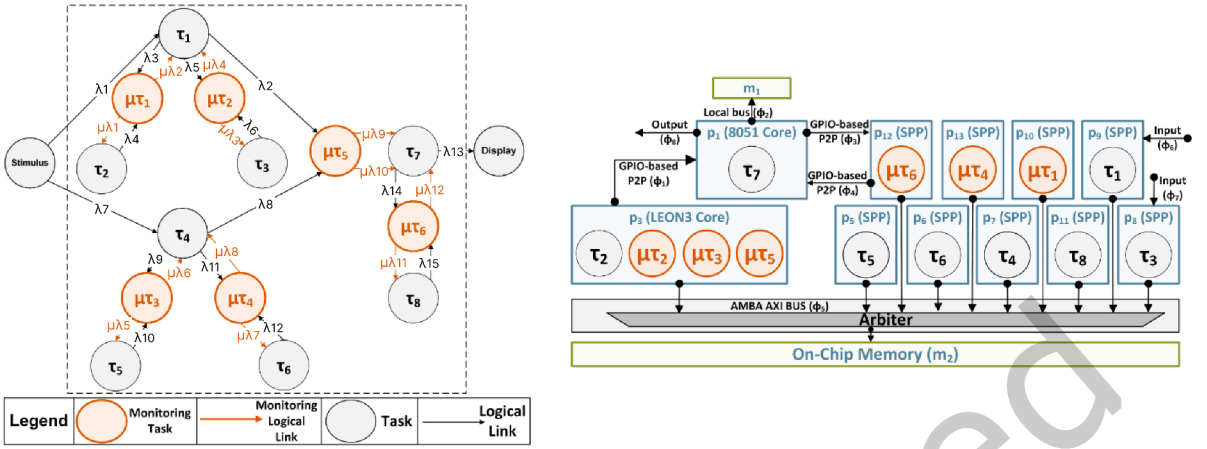


Fig. 9. FR^+ (left) and Selected$^+$ (right) of the monitored FFG use case

The considered *NFR* are $C = \{rt, area, res, pwd, energy\} = \{40ms, -, -, -, 50mJ\}$ and P, Φ, and M as reported in Section 6.1. Two further *NFR* are defined: minimizing system resource (*res*) and maximizing system flexibility, particularly by employing general-purpose processors whenever possible so that they are reprogrammable.

Table 5. *IBMREQ*, *EBMREQ*, and *SMREQ* for the monitored FFG use case.

mreq	*Metrics*	*Portability*	*Intrusiveness*
ibmreq$_1$	$\Gamma_M^1 = \{\mu\tau_1, \cdots, \mu\tau_6\}, \Lambda_M^1 = \{\mu\lambda_1, \cdots, \mu\lambda_{12}\}$	$(\tau_1, \tau_2), (\tau_1, \tau_3), (\tau_1, \tau_7), (\tau_4, \tau_7), (\tau_4, \tau_5), (\tau_4, \tau_6), (\tau_7, \tau_8)$	-
ibmreq$_2$	$\Gamma_M^2 = \{\mu\tau_1, \cdots, \mu\tau_6\}, \Lambda_M^2 = \{\mu\lambda_1, \cdots, \mu\lambda_{18}\}$	Λ	-
ebmreq$_3$	cache	Γ	-
smreq$_4$	bandwidth	p_i connected to shared m_q	-

The MREQs considered in this use case are reported in Table 5 and specified below:

- ***mreq*$_1$ → *ibmreq*$_1$**: To count the total number of messages exchanged between each directly connected τ_r pair part of the *FR*. For the *Metrics*, we placed a $\mu\tau_f$ (making use of $\mu\lambda_s$) between each of the τ_r pairs (see left side of Fig. 9, orange part), and we described, in SystemC, the actions of $\mu\tau_f$ as counting the number of messages that it receives and re-transmits. There are seven directly connected τ_r pairs; for the pairs: τ_1 and τ_7, τ_4 and τ_7, we used a single $\mu\tau_6$; $\mu\tau_6$ counts the total messages exchanged from the two pairs as a single metric. The *Portability* is represented by all the pairs of τ_r targeted by the monitoring activity.
- ***mreq*$_2$ → *ibmreq*$_2$**: To count the number of messages that flow through each λ_t of the *FR*. For the *Metrics*, we placed a $\mu\tau_f$ (making use of $\mu\lambda_s$) between each of the τ_r pair. Inside each $\mu\tau_f$ is added the code to perform the actions of counting the number of messages flowing on $\mu\lambda_s$. In this case, the $\mu\tau_f$ of ibmreq$_1$ are reused and the SystemC code is integrated. The *Portability* is represented by all the $\lambda_t \in \Lambda$;

To provide ibmreq$_1$ and ibmreq$_2$, *FR* of the FFG is extended with Γ_M and Λ_M to get FR$^+$, as shown by the orange elements in Fig. 9.

- ***mreq*$_3$ → *ebmreq*$_3$**: To count the number of data cache misses caused by each τ_r during the execution of the FFG on the final platform. It is not possible to describe the *Metrics* through the formalism of *FR*, as the

Table 6. Design performance related to monitored FFG.

	rt [ms]	*area*	*res [lut, ff, dsp, bram]*	*pwd [mW]*	*energy [mJ]*
Constraints C	40	-	[20800, 41600, 90, 50]	-	50
metrics^{+}	14.1	-	[11397, 6325, 4, 18]	-	30
budget	25.9	-	[9403, 35275, 86, 32]	-	20
Total oCMSs Intrusiveness	0	-	[3336, 7656, 0, 0]	-	-
metrics^{end}	14.1	-	[14733, 13981, 4, 18]	-	30

metric of number of data cache misses targets structural elements. Hence, we set ω = cache. The *Portability* is represented by all the $\tau_r \in \Gamma$.

- $\boldsymbol{mreq_4 \rightarrow smreq_4}$: To measure the bandwidth utilization of p_i accessing shared m_q during the execution of the FFG on the final platform. It is not possible to describe the *Metrics* through the formalism of *FR*; hence ω = bandwidth. The *Portability* is represented by all the $p_i \in P$, eventually connected to shared m_q.

It is worth noting that we did not provide any *Intrusiveness* requirement (see the last column of Table 5). Therefore, the objective is to satisfy the *FR* and *MREQs* while respecting the *NFR*.

6.2.2 ***System-Level Synthesis.*** The system-level synthesis is composed of the steps of Cost Estimation, design space exploration, and design space exploration for reuse.

Cost Estimation and Design Space Exploration. The cost estimation and design space exploration allow performing exploration to map 8 τ_r and 6 $\mu\tau_f$ on 5 different p_i instances (considering that each τ_r or $\mu\tau_f$ can be mapped on one of the two instances of Leon3, one of the two instances of Intel 8051, or on the single-purpose processor instance). Hence, the exploration is performed among $5^{(8+6)}$ alternatives. f^{comp} and f^{comm} are both managed through a genetic algorithm (as reported in Section 6.1). We obtained the Candidate^{+} reported in Fig. 9 (right side), showing p_i, ϕ_j, and m_q chosen during the exploration, together with the τ_r and $\mu\tau_f$ mapped on p_i (λ_t and $\mu\lambda_s$ are not reported for space reasons). Each p_i, ϕ_j, and m_q in Fig. 9 (right side) also reports the corresponding index inside P, Φ, and M. For the Candidate^{+}, n_{P^*} = 13 and n_{Φ^*} = 8.

The HW/SW co-simulation provides results for rt^{+} and $energy^{+}$, while the data from P, Φ, and M allows to estimate res^{+}, obtaining $metrics^{+}$ as reported in Table 6. The Candidate^{+} consumes the 60% of C(*energy*) and the 35% of C(*rt*), hence satisfies FR^{+} and *NFR* and it becomes the Selected^{+}.

Design space exploration for reuse. During the initialization, four matrices are created:

$$Q^{cache} = Q^{bandwidth} = \left[f_{p1}, f_{p3}, \ldots, f_{p13}, f_{\phi 1}, \ldots, f_{\phi 8}\right]_{1\times M}$$

$$oCMS^{cache} = oCMS^{bandwidth} = \left[\frac{p-block}{\phi-block}\right]_{M\times N^{DB}}$$

where p-block has n_{P^*} = 13 rows and N^{DB} = 30 columns, while ϕ-block has n_{Φ^*} = 8 rows and N^{DB} = 30 columns. On the other hand, Q^{ω} has $M = n_{P^*} + n_{\Phi^*}$ = 21 columns. All the matrix components of p-block and ϕ-block, as well as all the f_p and f_ϕ, are initialized to 0. Then, the substeps of transformation and refinement fill the matrices Q^{cache} and $Q^{bandwidth}$. The products of the substeps are reported in Fig. 10.

The querying substep provides the oCMSs resulting from the query on the database and the multiplication of oCMSs parameters with the frequency of involvement. Table 7 reports the parameters of oCMSs coming out from the querying substep. It is worth noting that data about *energy* and *pwd* are available only for a limited number of oCMSs in literature; in particular, no data were available for the oCMSs reported in Table 7. More details on how oCMSs *pd* and *res* have been extracted from the corresponding research papers and added to the database are reported in the repository[8].

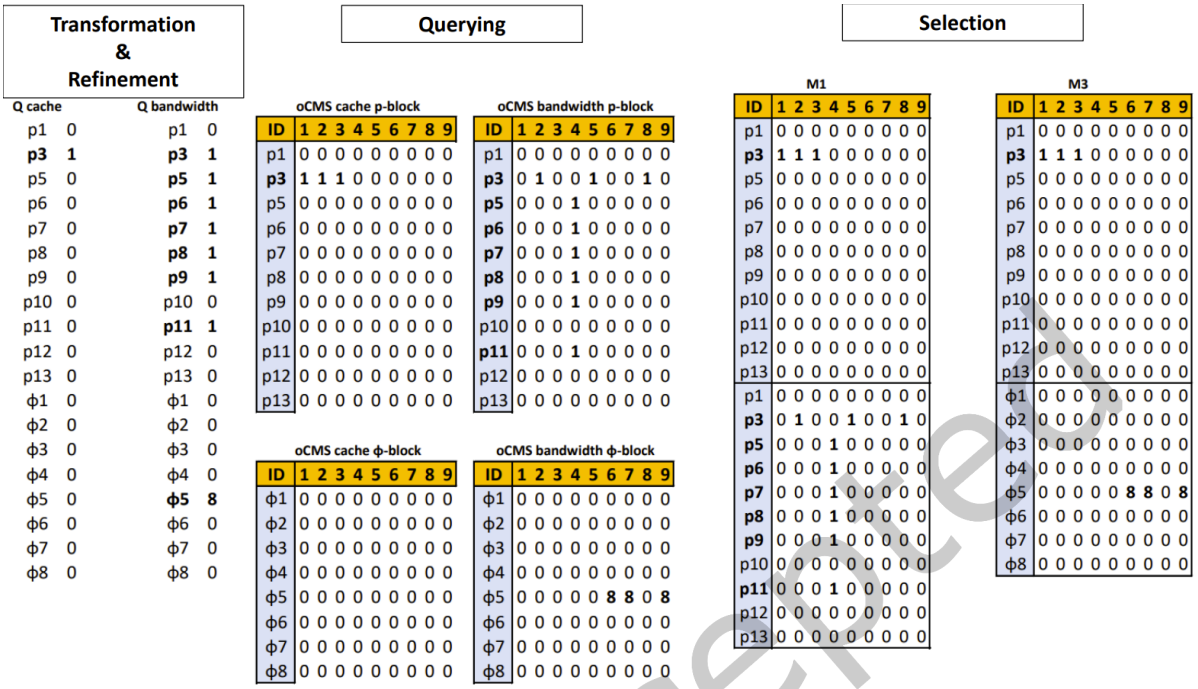


Fig. 10. Products of Transformation, Refinement, Querying, and Selection substeps in the monitored FFG use case. Note that Q^{cache} and $Q^{bandwidth}$ are in transposed form. M1 and M3 matrices are products of the Selection substep[3].

At this point, the surplus substep computes the budget by applying Eq. (4), obtaining the values reported in Table 6.

Finally, by applying the Selection substep, we obtained the combinations reported in Fig. 11. Both figures also show the budget Δrt and Δres as geometric planes. The total number of combinations on left side of Fig. 11 is 9, and the ones that respect the budget (referred to as solutions) is 8. On the other hand, the total number of combinations on right side of Fig. 11 is 9, with still 8 solutions. Among the solutions, we selected the one with minimum resource impact, as required by *NFR*. It is composed of:

- p_3 is monitored using one instance of the oCMS identified as ID3, which measures the number of cache misses. ID3 refers to the solution proposed in [34], which provides a HW oCMS capable of monitoring the cache of Leon3 processors in terms of various metrics.
- ϕ_5 monitored with eight instances of oCMS ID9, measuring the bandwidth. ID9 refers to the solution proposed in [8], which provides a HW oCMS capable of monitoring the bandwidth of different types of buses.

The total impact, in terms of *Intrusiveness* parameters of the monitoring model, is reported in Table 6. By putting the selected combination inside Selected^{+}, we obtained the Selected^{end} with the metrics^{end} reported in Table 6.

In summary, in this use case, starting from *FR*, *NFR*, and MREQs, we highlighted the intermediate artifacts produced by the proposed approach (see Fig. 10), showing the addressing of both behavioral MREQs and structural MREQs, as well as the reuse of existing oCMSs from literature.

Table 7. oCMSs resulting from the querying substep in the monitored FFG use case, described with monitoring model parameters. Parameters with * mean that that parameter is not multiplied by f_p or f_ϕ during the Querying substep.

ID	ω	*Portability*	oCMS	***pd*** **[cc]**	***res*** **[lut,ff,dsp,bram]**
1	cache	p_3	(2014) Ho et al.	0	[886, 303, 0, 0]
2	cache,bandwidth	p_3	(2023) L3SU	0	[78, 47, 0, 0]
3	cache	p_3	(2015) Moro et al.	0	[160, 320, 0, 0]
4	bandwidth	*spp*	(2021) Valente et al.	0	[887, 1014, 0, 2]
5	bandwidth	p_3	(2016) Valente et al.	0	[260, 205, 0, 0]
6	bandwidth	ϕ_5	(2023) AXI PMU	2848 *	[1127, 1681, 0, 1]
7	bandwidth	ϕ_5	(2010) Kyung et al.	0	[4815, 3168, 0, 0] *
8	bandwidth	p_3	(2022) Brilli et al.	300	0
9	bandwidth	p_3, ϕ_5	(2023) Brilli et al.	0	[397, 917, 0, 0]

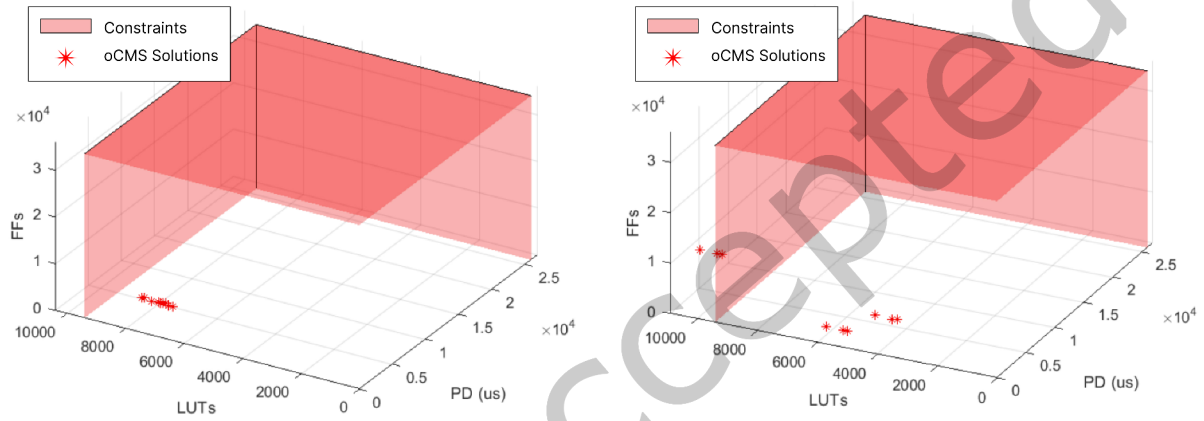


Fig. 11. Combinations of oCMSs produced by the selection substep in the monitored FFG use case, considering the M1 matrix (left) and M3 matrix (right). M1 and M3 matrices are products of Selection substep[3],

6.2.3 ***Comparison between customary approach and proposed approach***. In order to highlight the risks associated with the customary approach to satisfying MREQs (the customary approach is reported in Fig. 2), this section presents the consequences of addressing $mreq_1$ and $mreq_2$ (described in Section 6.2.1), using the customary flow. In the customary approach, MREQs are fulfilled during the oCMSs Integration step, which occurs after the verification & validation step, here referred to as *late oCMSs integration*.

Specifically, we first evaluate the impact of late oCMSs integration on development time by comparing the customary and proposed approaches. Then, we show how late oCMSs integration in the customary approach can result in late violations of *FR* and *NFR* across different implementation scenarios. It is worth noting that the development time was obtained by directly measuring the duration of each development activity as executed by an experienced embedded systems designer, following each respective approach. The selected designer was already proficient in using the HEPSYCODE HW/SW co-design tool, ensuring that the reported development time excludes any learning overhead. Furthermore, no previous designs or reference implementations were reused; as such, the reported figures reflect a ground-up development process. In scenarios where prior designs are available, development time may be significantly reduced. Finally, the designs are unoptimized; by this, we refer to the fastest correct implementation that satisfies *FR*, *NFR*, and MREQs, without investing additional effort in fine-tuning or optimizing for performance or resource usage.

Impact of late oCMSs Integration on Development Time. The goal of this paragraph is to show that integrating oCMSs only after verification & validation step may lead to a system redesign and an increase in development time, potentially resulting in a missed time-to-market deadline. To this end, we analyze the development time of a case in which the *FR*, *NFR*, and MREQs are satisfied using both the customary and proposed approaches. The corresponding development times, expressed in minutes, are reported in the third and fourth columns of Table 8.

Focusing on the customary approach (see third column of Table 8), during the requirements specification activity, the *FR* was defined using HEPSYCODE, and the *NFR* was provided through annotations. MREQs, however, were left in natural language, as is typically the case in customary flows [31]. This activity required approximately 120 minutes. The system-level synthesis activity took only 1 minute, given the simplicity of the application, and provided that all the $\tau_r \in \Gamma^+$ and $\lambda_t \in \Lambda^+$ were implemented on a SoC composed of one instance of Leon3. The resulting implementation, which satisfies *FR* and *NFR* without any monitoring logic, was developed through low-level implementation activity. Since all τ_r are mapped to the Leon3, they are implemented as SW-tasks, resulting in a complete C-based implementation within 120 minutes.

To transition from the unmonitored SoC to the monitored version, oCMSs were integrated via manual source code instrumentation. To implement $mreq_1$ and $mreq_2$, six instrumentation steps were necessary. Each instrumentation step took approximately 15 minutes. Unfortunately, after three such instrumentations (3×15 minutes), it was discovered that the response-time constraint $C(rt)$ was no longer satisfied (a more detailed discussion of the rt behavior depicted in Fig. 12 is provided in the next paragraph). This issue emerged after approximately 286 minutes of cumulative effort, forcing the design team to follow the feedback loop ③ to: (i) relax the *NFR*, (ii) drop the MREQs, or (iii) redesign the HW platform.

In this instance, we chose to pursue option (iii), preserving the original requirements and leveraging designer expertise to guide system-level synthesis toward an over-dimensioned HW solution. To do that, we manually introduced a 10% increase in the load estimates derived during the cost estimation step, attempting to account for the additional computational effort associated with monitoring operations. Focusing on Table 8 (the 2^{nd} iteration), after another 1-minute synthesis, we spent approximately 60 minutes redesigning and reimplementing a solution incorporating one additional Leon3 and some single-purpose processors to meet all the requirements, including *FR*, *NFR*, and (hopefully) the MREQs. We emphasize "hopefully" because, in the customary approach, the impact of monitoring is only verified post-integration. Fortunately, in this iteration, the oCMS integration proceeded successfully, requiring six instrumentations (6×15 minutes), and the final implementation satisfied all specified requirements.

In total, by using the customary approach, the second loop accounted for 151 additional minutes, resulting in a cumulative development time of 437 minutes, an increase of 53% over the initial loop. This outcome is consistent with the general trends shown in Fig. 1, even in the context of a relatively simple design.

Solving the same problem using the proposed approach required a requirements specification step that included the definition of the *FR*, the addition of $\mu\tau$ and $\mu\lambda$ elements to model $ibmreq_1$ and $ibmreq_2$ (as per Fig. 9), and the annotation of the *NFR*. This activity took approximately 150 minutes in total, representing an overhead of only 30 minutes compared to the non-monitored version. Subsequently, the system-level synthesis was executed in less than 2 minutes. The resulting implementation, referred to as $Solution^{end}$, satisfies both FR^+ and *NFR*, and was finalized through the low-level implementation activity. This final step required approximately 180 minutes to generate a complete HW/SW implementation from the SystemC-based HEPSYCODE model.

In summary, the total development time using the proposed approach was 332 minutes, representing a reduction of approximately 24% compared to the 437 minutes required by the customary approach. Furthermore, even in the optimistic scenario where the customary approach achieves a valid implementation on the first attempt, the resulting development time would be 331 minutes (resulting from the addition of 3 more instrumentation points to the 286 minutes of the first loop iteration in the customary approach), effectively identical to that of

Table 8. Estimated development time for a monitored FFG using the customary and proposed approaches.

Loop Iteration	Activity	Customary Approach (min)	Proposed Approach (min)
1^{st}	Requirements Specification	120	150
	System-Level Synthesis	1	2
	Low-Level Implementation	120	180
	oCMS Integration	3×15	–
2^{nd}(loop ③)	Requirements Specification	0	–
	System-Level Synthesis	1	–
	Low-Level Implementation	60	–
	oCMS Integration	6×15	–
	Total (1^{st} loop)	**286**	**332**
	Total (2^{nd} loop)	**151**	**0**
	Total	**437 (+53% w.r.t. first loop)**	**332**

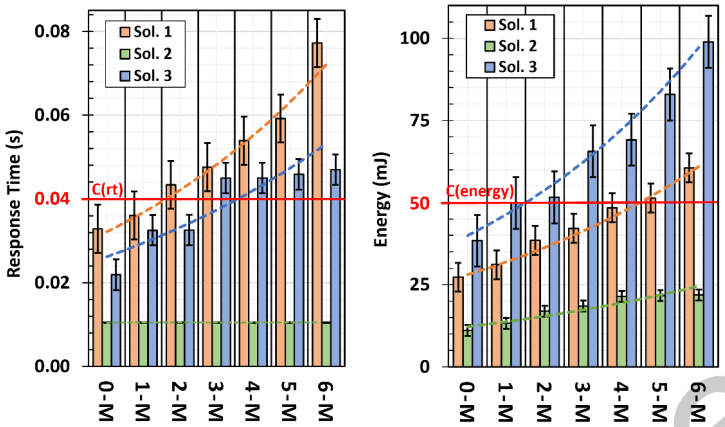
the proposed approach. This confirms that the proposed approach introduces no meaningful overhead, while providing more predictable and robust design outcomes by avoiding the need for costly feedback iterations.

It is worth noting that, due to the relative simplicity of the system under development, we were able to manually fix the design obtained through the customary approach within a single feedback loop. However, this number can easily increase in the case of more complex designs. Moreover, if the prototyping were ASIC-oriented instead of FPGA-based, the involvement of a foundry in the process (typically adding at least an extra month) would significantly increase the cost of a single feedback loop, potentially resulting in a missed time-to-market deadline.

Impact of late oCMSs Integration on FR and NFR. In this paragraph, we show how late oCMSs integration can result in violations of *FR* and *NFR* across different implementation scenarios. We assume that three different types of SoC platforms have successfully reached the verification & validation step of Fig. 2. We then analyze how the subsequent integration of oCMSs affects each platform configuration.

The three cases are described as follows:

- **Case 1)** All the $\tau_r \in \Gamma$ and $\lambda_t \in \Lambda$ implemented on a SoC constituted of one instance of Leon3. To satisfy mreq_1 and mreq_2, expressed in natural language, we adopt the oCMS technique of source code instrumentation, through which the metric and portability requirements can be satisfied; in particular, the metric is implemented by adding source code to the Leon3 able to increment a SW counter, while the portability is satisfied through the cross-compilation for the Leon3 target. It is worth noting that this is exactly the case analyzed in the previous paragraph.
- **Case 2)** All the $\tau_r \in \Gamma$ implemented on a SoC constituted of dedicated single-purpose processors for each τ_r. To satisfy mreq_1 and mreq_2, we connect a custom counter to each single-purpose processor, able to count the number of messages produced by single-purpose processors and flowing on each ϕ_j;



(a) *rt* (b) *energy*

Fig. 12. *rt* (a) and *energy* (b) when satisfying mreq_1 and mreq_2 on the monitored FFG with the customary approach.

- **Case 3)** $\tau_r \in \Gamma$ implemented on a SoC with two Intel 8051 and one Leon3, that communicate through m_q and ϕ_j. To satisfy mreq_1 and mreq_2, we adopt the oCMS technique of source code instrumentation.

In all three cases, the substep of oCMSs Integration is performed, and *rt* and *energy* are measured. Fig. 12 shows the trend of *rt* and *energy*. x-axes refer to the implementation without oCMSs (referred to as *0-M*) and, moving to the right, *1-M* to *6-M* refer to the addition of oCMSs to satisfy $mreq_1$ and $mreq_2$. The orange, green, and blue bars refer to the first, second, and third cases, respectively. In both graphs, there are horizontal lines indicating C(*rt*) and C(*energy*). For cases (1) and (3), it is visible how the final implemented system violates both C(*rt*) and C(*energy*).

The failure to meet the input *NFR* for the case (1) and (3) leads designers to follow the feedback loop ③ of Fig. 2, in order to (i) relax *NFR*, (ii) remove MREQs, or (iii) redesign the HW platform. Consequently, the development time increases, as shown in the previous paragraph.

6.3 Design of a Monitored System-on-Chip for a Pacemaker

This use case considers the design of a monitored SoC executing a pacemaker control application, referred to as monitored pacemaker. The objective is to demonstrate how the proposed approach enables a *design-space exploration* to trade off between response time and occupied resources, by reducing the level of monitoring – addressing aspect (ii) from the Introduction. The HEPSYCODE model and the corresponding *NFR* have been primarily derived from [47] and [27]. Code and resources related to this use case are available in the GitHub repository[8].

6.3.1 ***Requirements specification.*** The monitored pacemaker receives inputs from the two heart cavities, referred to as natural ventricular (NV) and natural atrial (NA); based on the heart rate frequency (indicated as f^{HR}), it responds by providing stimulus when necessary (stimulated ventricular (SV) and stimulated atrial (SA)). The pacemaker is required to be able to work with a frequency up to 120 bpm (i.e., $f^{HR} = 120$ bpm), with an accepted deviation of 1 bpm. This requirement sets the constraints on design metrics at $C(rt) = 504$ ms. The *FR* contains *22* τ_r and *57* λ_t. Two more τ_r (stimulus and display) are used to model the input NA, NV, and f^{HR}. The

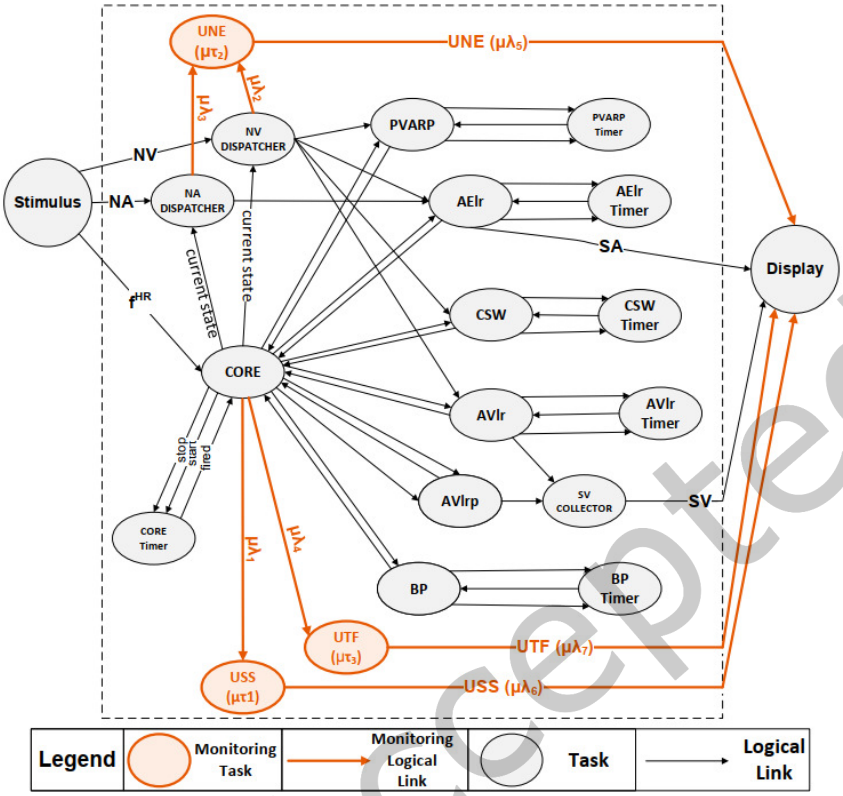


Fig. 13. FR^+ of the monitored pacemaker. UTF (unexpected timer fired), UNE (unexpected natural event), and USS (unexpected state sequence) refer to ibmreq$_1$, ibmreq$_2$, and ibmreq$_3$, respectively.

FR is reported in Fig. 13 (gray circles and black arrows) in a reduced form for reader convenience (the whole *FR* is available in the repository).

It is worth noting that when an SA-SV sequence is generated by the pacemaker, it represents a situation in which all tasks are involved in the processing (i.e., 100% model coverage). For this reason, it will later be used to evaluate worst-case scenarios (cf. Fig. 15).

The *NFR* are $C = [504\text{ ms}, –, –, –, –]$ and P, Φ, and M as described in Section 6.1. A further *NFR* is defined: minimizing system resources (*res*).

The MREQs considered in this use case have been mainly inspired by [46] and are specified as follows:

- $\boldsymbol{mreq_1 \rightarrow ibmreq_1}$: To detect whether the *CORE_Timer* τ_r is already fired upon entering the CSW state. Under normal conditions, it should not be fired; if it is, this may indicate undetected physical defects or aging;
- $\boldsymbol{mreq_2 \rightarrow ibmreq_2}$: To detect when the pacemaker receives natural events as input when it is not in a state able to manage them. This problem can happen due to a not accurate f^{HR} provided as input;
- $\boldsymbol{mreq_3 \rightarrow ibmreq_3}$: To detect when the pacemaker is moving along a state sequence that is not correct. This problem can happen due to undetected manufacturing physical defects or aging;

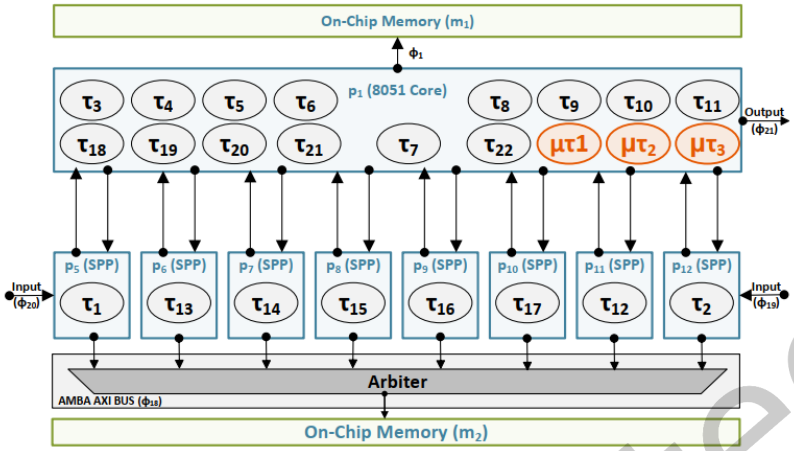


Fig. 14. Candidate$^+$ of the monitored pacemaker, which becomes the Selected$^+$.

Table 9. Design performance for the monitored pacemaker. The leftward arrow (←) refers to the initial metrics related to the oCMSs solution for $f^{HR} = 120$ bpm, while both the upward (↑) and downward (↓) arrows indicate variations in the *pd* and *res* metrics after the monitored SoC design space exploration.

f^{HR}		*rt [ms]*	*area*	*res [lut, ff, dsp, bram]*	pd_{var}% - oCMSs	res_{var}% - oCMSs
120	Constraints C	504	-	[20800, 41600, 90, 50]		
	metrics$^+$	504	-	[2338, 1445, 0, 3]		
	budget_{120}	0	-	[18462, 40155, 90, 47]		
	$\text{metrics}_{120}^{end}$	504	-	[2506, 1613, 0, 3]	←	←
80	budget_{80}	4	-	[18462, 40155, 90, 47]		
	$metrics_{80}^{end}$	501	-	[2000, 1500, 0, 3]	↑ 0.2%	↓ 2.5%
60	budget_{60}	11	-	[18462, 40155, 90, 47]		
	$metrics_{60}^{end}$	494	-	[2000, 1500, 0, 3]	↑ 0.2%	↓ 2.5%

- $\boldsymbol{mreq_4 \rightarrow smreq_4}$: To measure the power consumption over time of the general-purpose processors and single-purpose processors composing the system.

IBMREQ are added to *FR* (see Fig. 13, with *IBMREQ* in orange circles and arrows) to obtain the FR^+.

6.3.2 ***System-Level Synthesis***. The cost estimation and design space exploration steps allow performing exploration to map 22 τ_r and 3 $\mu\tau_f$ on 5 different p_i instances. The Candidate$^+$ is reported in Fig. 14. The HW/SW co-simulation, performed with $f^{HR} = 120$ bpm, provides the $metrics^+$ reported in Table 9. The rt^+ reaches the value of C(*rt*), hence the Candidate$^+$ becomes the Selected$^+$. Selected$^+$ satisfies *FR*, *NFR*, and *IBMREQ*.

In the design space exploration for reuse step, the goal is to satisfy the *SMREQ*. The querying provides 7 oCMSs to monitor p_1 and 2 oCMSs to monitor the single-purpose processors of the Selected$^+$. The available budget is reported on left side of Fig. 15 and Table 9 as budget_{120}. By using the querying results, the Selection substep provides 896 combinations, reported both in Table 10 and plotted in the scatter plot on right side of Fig. 15 (red points). Considering the available budget, indicated in red plane in Fig. 15, we obtained different solutions in satisfying mreq_4. In this case, we obtained as solutions only the combinations involving the oCMSs not impacting on *pd*, namely 256 solutions out of 896 combinations.

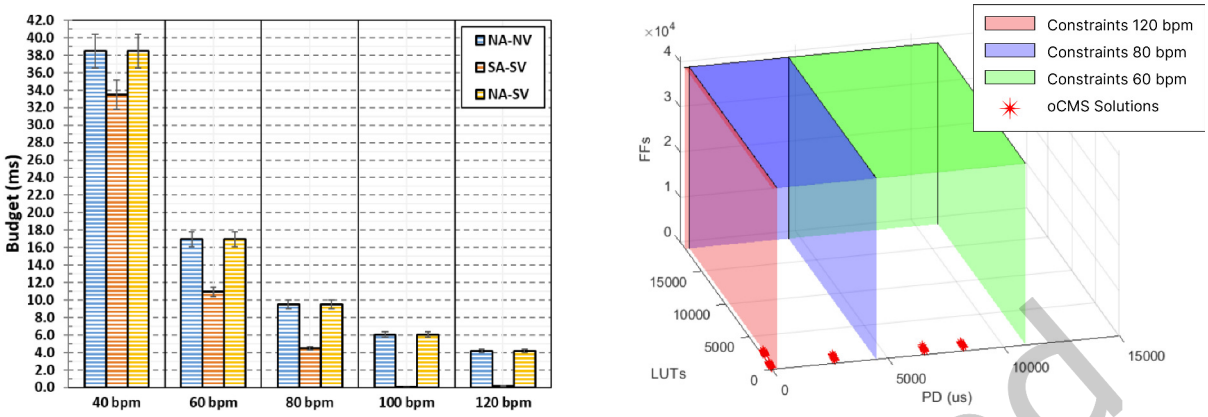


Fig. 15. Results for the pacemaker. On the left side, available budget, in terms of *rt* (i.e., Δrt) for the monitored pacemaker for different f^{HR} and different sequences of stimulus. The sequence SA-SV represents the worst-case scenario for a pacemaker since it has to generate both atrial and ventricular stimuli. On the right side, combinations given by the Selection substep in the monitored pacemaker use case for different budget values.

Among the 256 total combinations, we selected as the best solution the one with minimum *res*, which is composed of one type of oCMS to monitor the power, the one from Najem et al. [38]. The oCMS presented in [38] is a HW oCMS designed to be connected to specific components and to estimate power consumption using an all-digital approach. It operates by building a model to derive power measurements based on monitored signals. In the selected solution, it is connected in the following way: 14 instances connected to p_1, and 1 instance connected to single-purpose processors from p_6 to p_{12}. Once included the oCMSs in the Selected$^+$, the final monitored SoC, namely Selected^{end}, has the metrics$^{end}_{120}$ reported in Table 9.

In order to highlight how the proposed approach allows quick exploration of other monitoring solutions, we perform the design space exploration for reuse by starting from the same Selected$^+$, but changing the budget. We considered the budget obtained by making the system work at f^{HR} = 60 bpm and f^{HR} = 80 bpm; the new budgets are referred to as $budget_{80}$ and $budget_{60}$, and are reported both in Fig. 15 and Table 9. For these budgets, we obtain more solutions, as reported in Table 10 and visible in Fig. 15. This is due to the fact that in this case, the budget in terms of *rt* is larger. The best solution in both cases is:

- 1 instance of the oCMS from Wang et al. [60] connected to p_1. This type of oCMS is a piece of SW to monitor the component;
- 1 instance of oCMS from Najem et al. [38] connected to single-purpose processors from p_6 to p_{12}.

For f^{HR} = 80 bpm and f^{HR} = 60 bpm, the Selected^{end} has $metrics^{end}_{80}$ and $metrics^{end}_{60}$ reported in Table 9.

By lowering the f^{HR}, we are effectively accepting a system working at a slower heart rate. In turn, this provides more budget. In introducing oCMSs during the design space exploration for reuse, we can lower the impact in *res* of oCMSs (*res* ↓ in Table 9) by considering SW elements to monitor the system, in turn impacting with a *pd* on the budget (*pd* ↑ in Table 9). The action of *res* ↓ and *pd* ↑ is effectively a trade-off between response time and occupied resources (as mentioned in the Introduction).

In summary, this use case concerned the design of a monitored SoC executing a pacemaker control application, making use of both SW and HW elements to observe the system, depending on the type of *NFR* as input. This is highlighted by the oCMSs contained in Selected^{end} for different types of f^{HR}, for which the results reported in Table 10 and Fig. 15 show how the approach enables the exploration among different monitoring solutions.

Table 10. Resulting combinations of oCMSs and solutions respecting the three budgets.

Total number of combinations	896
Solutions for 120 bpm	256
Solutions for 80 bpm	384
Solutions for 60 bpm	640

The total time required to perform the system-level synthesis step in the proposed approach for this use case was consistently 2 minutes for each evaluated value of f^{HR}. This time is reported because it highlights the primary efficiency gain introduced by the proposed approach, particularly when combined with the elimination of the costly feedback loop ③ shown in Fig. 2. It can be noticed that this time is equal to the time spent in the system-level synthesis activity for the monitored FFG use case (see Table 8). This is because the system-level synthesis time in the proposed approach is dominated by the design space exploration step (as will be discussed in the next section), which is implemented as a multi-objective optimization process. Since the design space exploration step required the same execution time in both cases, the overall synthesis time remains unchanged.

6.4 Discussion about the Scalability

The scalability of the proposed approach in terms of development time and with respect to the size of the systems under design can be analyzed by considering its two main components: the adopted HW/SW co-design tool and the algorithms involved in the design space exploration for reuse. Regarding the first component, the analysis focuses on the cost estimation and design space exploration steps. In this work, we rely on HEPSYCODE and thus refer to its scalability characteristics as follows:

- Cost Estimation. HEPSYCODE is primarily based on a technology library, which is fully evaluated offline. As a result, the cost of adding new $p_i \in P$, $\phi_j \in \Phi$, and $m_q \in M$ is incurred only once. Metrics estimated at runtime include:
  - *Affinity*, which is re-evaluated each time the *FR* or *NFR* are modified. The associated algorithm has a complexity of $O(\#\text{tasks} \times \#\text{processors}) = O(n^2)$ [6].
  - *Load* and *Concurrency*, evaluated via a built-in HW/SW co-simulator. These estimations have complexities of $O(\#\text{tasks} \times \#\text{processors}) = O(n^2)$ and $O(\#\text{tasks} \times \#\text{logical links}) = O(n^2)$, respectively [6, 36, 45]
- Design Space Exploration. The design space exploration step in HEPSYCODE is based on:
  - A genetic algorithm whose theoretical complexity depends on the size of the search space, i.e., $O(\#\text{tasks} \times \#\text{logical links} \times \#\text{processors} \times \#\text{physical links}) = O(n^4)$. However, the algorithm aims to find suboptimal solutions, and its execution time can be controlled by appropriately tuning the genetic parameters, such as the population size and number of generations.
  - HW/SW co-simulation based on an event-driven simulation engine. Its scalability with respect to the number of processes and logical links has been validated on real-world applications [6, 36, 45].

Regarding the design space exploration for reuse (as described in Section 5), it consists of six substeps, each one with an associated algorithm: Initialization, Transformation, Refinement, Querying, Surplus, and Selection. Their execution times primarily depend on the cardinality of EBMREQ, SMREQ, the number of $p_i \in P$, the number of $\phi_j \in \Phi$, and available oCMSs in the database. Among them, Selection is the most computationally demanding, with a complexity of $O(m \cdot k^m)$, where m is the number of monitored components (e.g., $p_i \in P$, $\phi_j \in \Phi$) and k is the maximum number of candidate oCMSs per component. To mitigate the exponential worst-case complexity, future versions of the Selection substep will incorporate Pareto-optimal exploration strategies, aiming to reduce the size of the search space while preserving solution quality.

Based on this analysis, the impact of adding *ibmreq*s to the baseline HEPSYCODE model is negligible. This is supported by the experimental results reported in Section 6, where the complete design space exploration time was consistently less than 2 minutes, even with models containing 11 τ_r (monitored FFG) and up to 22 τ_r (monitored pacemaker). In conclusion, the total development time using the proposed approach is entirely dominated by the design space exploration step.

7 Conclusions and Future Works

Existing HW/SW co-design methodologies for monitored SoCs exhibit some shortcomings, such as not being able to capture all the types of MREQs, limiting the design-space exploration to some specific oCMSs solutions, and not allowing the reuse of existing oCMSs. These shortcomings hinder the efficiency of monitored SoCs development, limiting the designers ability to optimize their design choices and exposing the risk to miss the Time-to-Market deadlines.

The work presented in this paper addresses these shortcomings by proposing a new approach for the HW/SW co-design of monitored SoCs. In particular, as indicated in seminal works on design methodologies, the proposed approach (i) guarantees to capture the requirements at system-level, (ii) allows a design space exploration of monitored SoCs, and (iii) favors the reuse of existing components.

To do this, we first reviewed how MREQs and oCMSs parameters are commonly expressed in the literature, aiming to identify any gaps in their relationship. Through this analysis, we identified a missing link between them. Subsequently, we developed a new monitoring model to establish this relationship and facilitate the fulfillment of MREQs with oCMSs at the system-level. Then, we added the steps to satisfy MREQs with oCMSs at system-level inside a reference HW/SW co-design flow, obtaining a new HW/SW co-design flow reported in Fig. 7. The proposed approach introduces a way to express and capture MREQs together with *FR* and *NFR*, as well as a new step referred to as Design Space Exploration for reuse. The new step is composed of different substeps, for which the respective pseudo-code is provided in the external Appendix[3].

The proposed approach provides a design flow that builds a monitored SoC using a database encompassing P, Φ, M, and oCMSs. This facilitates keeping up with the advancements in components necessary for building monitored SoCs. For instance, by augmenting the oCMSs database with additional entries measuring bandwidth, the richness of combinations depicted in Fig. 11 can be expanded, offering designers a broader array of options. Furthermore, as the complexity and the design effort to develop monitored SoCs grows, an approach like the presented one that favors reuse allows to keep the design time under safe bounds [30]. In essence, by adopting the proposed approach, designers would be empowered to input all requirements and rapidly assess various monitored SoC solutions within a matter of minutes. This would ultimately result in a significant enhancement of the efficiency of the monitored SoCs development lifecycle.

The experimental activities highlighted the effectiveness of the proposed approach in designing a monitored SoC for a synthetic application and a real-world use case. The first experimental activity captured mixed MREQs (i.e., targeting both behavioral and structural domain elements) at system-level, together with *FR* and *NFR*, and highlighted all the intermediate artifacts of the proposed approach to reach the final monitored SoC. The final monitored SoC was composed by reusing existing oCMSs, with a total time to apply the proposed approach in the step of system-level synthesis equal to 2 minutes. The second experimental activity involved capturing MREQs, *FR*, and *NFR* at the system-level, showing the exploration of the monitored SoC. This exploration involved trading off resource utilization and performance degradation, in turn increasing the flexibility in design choices.

Future work will focus on enabling the automatic selection of the best monitored SoC depending on input requirements, and on refining the *Selection* substep to generate Pareto-optimal solutions. Additionally, we aim to extend the proposed approach to support the HW/SW co-design of custom oCMSs for *EBMREQ* and *SMREQ*,

which are currently satisfied only through the reuse of existing oCMSs. This extension will build upon the framework for flexible oCMS construction presented in [58].

Further development will also include the refinement of the integration of the proposed approach into the HEPSYCODE tool, as well as the extension of its interoperability with other HW/SW co-design tools (e.g., S3D[9]). These enhancements will facilitate the application of the proposed approach to a broader range of use cases.

Acknowledgments

This work was funded by the Italian Ministry of Economic Development (MISE) under the project "SICURA – Casa intelligente delle tecnologie per la sicurezza" (CUP C19C200005200004), within the investment plan for the deployment of ultra-broadband infrastructure (FSC 2014–2020), and by the European Union – NextGenerationEU through the Italian Ministry of University and Research (MUR) under the National Innovation Ecosystem grant ECS00000041 – VITALITY (CUP E13C22001060006).

References

[1] Rajeev Alur and David L. Dill. 1994. A theory of timed automata. *Theoretical Computer Science* 126, 2 (1994), 183–235. doi:10.1016/0304-3975(94)90010-8

[2] ARM 2021. *Embedded Trace Macrocell Architecture Specification.* ARM. Retrieved September 15, 2025 from https://developer.arm.com/documentation/ihi0014/latest

[3] ARM 2023. *AMBA 4 Specifications.* ARM. Retrieved September 15, 2025 from https://www.arm.com/architecture/system-architectures/amba/amba-4

[4] ARM 2023. *CoreSight Performance Monitoring Unit Architecture.* ARM. Retrieved September 15, 2025 from https://developer.arm.com/documentation/ihi0091/latest/

[5] Imran Ashraf, Mottaqiallah Taouil, and Koen Bertels. 2015. Memory profiling for intra-application data-communication quantification: A survey. In *2015 10th International Design Test Symposium (IDT).* IEEE, Piscataway, NJ, USA, 32–37. doi:10.1109/IDT.2015.7396732

[6] Carlo Brandolese, William Fornaciari, Luigi Pomante, Fabio Salice, and Donatella Sciuto. 2006. Affinity-driven system design exploration for heterogeneous multiprocessor SoC. *IEEE Trans. Comput.* 55, 5 (May 2006), 508–519. doi:10.1109/TC.2006.66

[7] Gianluca Brilli, Alessandro Capotondi, Paolo Burgio, and Andrea Marongiu. 2022. Understanding and Mitigating Memory Interference in FPGA-based HeSoCs. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, Piscataway, NJ, USA, 1335–1340. doi:10.23919/DATE54114.2022.9774768

[8] Gianluca Brilli, Giacomo Valente, Alessandro Capotondi, Paolo Burgio, Tania Di Mascio, Paolo Valente, and Andrea Marongiu. 2023. Fine-Grained QoS Control via Tightly-Coupled Bandwidth Monitoring and Regulation for FPGA-based Heterogeneous SoCs. In *2023 60th ACM/IEEE Design Automation Conference (DAC).* IEEE, Piscataway, NJ, USA, 1–6. doi:10.1109/DAC56929.2023.10247840

[9] Jon Perez Cerrolaza, Roman Obermaisser, Jaume Abella, Francisco J. Cazorla, Kim Grüttner, Irune Agirre, Hamidreza Ahmadian, and Imanol Allende. 2020. Multi-core Devices for Safety-critical Systems: A Survey. *ACM Comput. Surv.* 53, 4, Article 79 (Aug. 2020), 38 pages. doi:10.1145/3398665

[10] Daniele Ciambrone, Vittoriano Muttillo, Luigi Pomante, and Giacomo Valente. 2018. HEPSIM: An ESL HW/SW co-simulator/analysis tool for heterogeneous parallel embedded systems. In *2018 7th Mediterranean Conference on Embedded Computing (MECO).* IEEE, Piscataway, NJ, USA, 1–6. doi:10.1109/MECO.2018.8406078

[11] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. 2006. The Detection and Classification of Non-Functional Requirements with Application to Early Aspects. In *14th IEEE International Requirements Engineering Conference (RE'06).* IEEE, Piscataway, NJ, USA, 39–48. doi:10.1109/RE.2006.65

[12] Vittorio Cortellessa, Luigi Pomante, and Vincenzo Stoico. 2023. From UML/MARTE Specifications to ESL HW/SW Co-Design: Early Functional Verification and Timing Validation. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering* (Coimbra, Portugal) *(ICPE '23 Companion).* Association for Computing Machinery, New York, NY, USA, 373–380. doi:10.1145/3578245.3584850

[13] Gabriella D'Andrea, Tania Di Mascio, and Giacomo Valente. 2019. Self-adaptive loop for CPSs: is the Dynamic Partial Reconfiguration profitable?. In *2019 8th Mediterranean Conference on Embedded Computing (MECO).* IEEE, Piscataway, NJ, USA, 1–5. doi:10.1109/MECO.2019.8760036

[9] S3D Tool: https://s3d.unican.es/

[14] André de Matos Pedro, Jorge Sousa Pinto, David Pereira, and Luís Miguel Pinho. 2018. Runtime verification of autopilot systems using a fragment of MTL. *International Journal on Software Tools for Technology Transfer* 20, 4 (01 Aug 2018), 379–395. doi:10.1007/s10009-017-0470-5

[15] Tania Di Mascio, Federica Caruso, Laura Tarantino, and Giacomo Valente. 2021. MONICA Vision: An Approach, a Model and the Interactive Tools for Cyber-Physical Systems Designers. In *CHItaly 2021: 14th Biannual Conference of the Italian SIGCHI Chapter*. Association for Computing Machinery, New York, NY, USA, Article 33, 5 pages. https://doi.org/10.1145/3464385.3464778

[16] Daniele Di Pompeo, Emilio Incerto, Vittoriano Muttillo, Luigi Pomante, and Giacomo Valenete. 2017. An Efficient Performance-Driven Approach for HW/SW Co-Design. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering* (L'Aquila, Italy) *(ICPE '17)*. Association for Computing Machinery, New York, NY, USA, 323–326. doi:10.1145/3030207.3030239

[17] ECS-SRIA Steering Committee. 2024. *ECS Strategic Research and Innovation Agenda (ECS-SRIA) 2025*. Strategic Research and Innovation Agenda. AENEAS, EPoSS, INSIDE Industry Association. https://ecssria.eu/2025

[18] Michael Factor, Assaf Schuster, and Konstantin Shagin. 2004. Instrumentation of standard libraries in object-oriented languages: the twin class hierarchy approach. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Vancouver, BC, Canada) *(OOPSLA '04)*. Association for Computing Machinery, New York, NY, USA, 288–300. doi:10.1145/1028976.1029000

[19] Tiziana Fanni, Daniel Madronal, Claudio Rubattu, Carlo Sau, Francesca Palumbo, Eduardo Juarez, Maxime Pelcat, Cesar Sanz, and Luigi Raffo. 2019. Run-time Performance Monitoring of Heterogenous Hw/Sw Platforms Using PAPI. In *Proceedings Sixth International Workshop on FPGAs for Software Programmers*. VDE VERLAG, Berlin, Germany, 1–10.

[20] Fabrizio Ferrandi, Vito Giovanni Castellana, Serena Curzel, Pietro Fezzardi, Michele Fiorito, Marco Lattuada, Marco Minutoli, Christian Pilato, and Antonino Tumeo. 2021. Invited: Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, Piscataway, NJ, USA, 1327–1330. doi:10.1109/DAC18074.2021.9586110

[21] Frontgrade Gaisler 2022. *LEON3 processor*. Frontgrade Gaisler. Retrieved September 15, 2025 from https://www.gaisler.com/products/leon3

[22] Andreas Gerstlauer and Daniel D. Gajski. 2002. System-Level Abstraction Semantics. In *Proceedings of the 15th International Symposium on System Synthesis* (Kyoto, Japan) *(ISSS '02)*. IEEE, Piscataway, NJ, USA, 231–236. doi:10.1145/581199.581251

[23] Mohamed Ben Hammouda, Philippe Coussy, and Loïc Lagadec. 2017. A Unified Design Flow to Automatically Generate On-Chip Monitors During High-Level Synthesis of Hardware Accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 3 (2017), 384–397. doi:10.1109/TCAD.2016.2587278

[24] Shaahin Hessabi, Amir Masoud Gharehbaghi, Benyamin Hamdin Yaran, and Maziar Goudarzi. 2004. Integrating assertion-based verification into system-level synthesis methodology. In *Proceedings 16th International Conference on Microelectronics*. IEEE, Piscataway, NJ, USA, 232–235. doi:10.1109/ICM.2004.1434254

[25] Charles Antony Richard Hoare. 1978. Communicating Sequential Processes. *Commun. ACM* 21, 8 (aug 1978), 666–677. doi:10.1145/359576.359585

[26] Intel. 2023. *Performance Monitoring Units*. Intel. Retrieved September 15, 2025 from https://perfmon-events.intel.com/

[27] Zhihao Jiang, Miroslav Pajic, and Rahul Mangharam. 2012. Cyber–Physical Modeling of Implantable Cardiac Medical Devices. *Proc. IEEE* 100, 1 (2012), 122–137. doi:10.1109/JPROC.2011.2161241

[28] Sourabh Katoch, Sumit Singh, and Vijay Kumar. 2021. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications* 80, 5 (2021), 8091–8126. doi:10.1007/s11042-020-10139-6

[29] Joachim Keinert, Martin Streubühr, Thomas Schlichter, Joachim Falk, Jens Gladigau, Christian Haubelt, Jürgen Teich, and Michael Meredith. 2009. SystemCoDesigner—an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. *ACM Trans. Des. Autom. Electron. Syst.* 14, 1, Article 1 (jan 2009), 23 pages. doi:10.1145/1455229.1455230

[30] Kurt Keutzer, Arthur Richard Newton, Jan M. Rabaey, and Alberto Sangiovanni-Vincentelli. 2000. System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19, 12 (2000), 1523–1543. doi:10.1109/43.898830

[31] Georgios Kornaros and Dionisios Pnevmatikatos. 2013. A Survey and Taxonomy of On-Chip Monitoring of Multicore Systems-on-Chip. *ACM Trans. Des. Autom. Electron. Syst.* 18, 2, Article 17 (April 2013), 38 pages. doi:10.1145/2442087.2442088

[32] Jong Chul Lee and Roman Lysecky. 2015. System-Level Observation Framework for Non-Intrusive Runtime Monitoring of Embedded Systems. *ACM Trans. Des. Autom. Electron. Syst.* 20, 3, Article 42 (June 2015), 27 pages. doi:10.1145/2717310

[33] Marcel Mettler, Daniel Mueller-Gritschneder, and Ulf Schlichtmann. 2020. Runtime Monitoring of Inter- and Intra-Thread Requirements on Embedded MPSoCs. In *Proceedings 33rd International Conference on VLSI Design and 19th International Conference on Embedded Systems (VLSID)*. IEEE, Piscataway, NJ, USA, 49–54. doi:10.1109/VLSID49098.2020.00026

[34] Andrea Moro, Fabio Federici, Giacomo Valente, Luigi Pomante, Marco Faccio, and Vittoriano Muttillo. 2015. Hardware performance sniffers for embedded systems profiling. In *2015 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES)*. IEEE, Piscataway, NJ, USA, 29–34.

[35] Vittoriano Muttillo, Paolo Giammatteo, Vincenzo Stoico, and Luigi Pomante. 2020. An early-stage statement-level metric for energy characterization of embedded processors. *Microprocessors and Microsystems* 77 (2020), 103200. doi:10.1016/j.micpro.2020.103200

[36] Vittoriano Muttillo, Luigi Pomante, Marco Santic, and Giacomo Valente. 2023. SystemC-based Co-Simulation/Analysis for System-Level Hardware/Software Co-Design. *Computers and Electrical Engineering* 110 (2023), 108803. doi:10.1016/j.compeleceng.2023.108803

[37] Vittoriano Muttillo, Giacomo Valente, Luigi Pomante, Vincenzo Stoico, Fausto D'Antonio, and Fabio Salice. 2018. 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* (Berlin, Germany) *(ICPE '18)*. Association for Computing Machinery, New York, NY, USA, 119–122. doi:10.1145/3185768.3186291

[38] Mohamad Najem, Pascal Benoit, Mohamad El Ahmad, Gilles Sassatelli, and Lionel Torres. 2017. A Design-Time Method for Building Cost-Effective Run-Time Power Monitoring. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 7 (2017), 1153–1166. doi:10.1109/TCAD.2016.2614347

[39] Ahmed Nassar, Fadi J. Kurdahi, and Wael Elsharkasy. 2015. NUVA: Architectural support for runtime verification of parametric specifications over multicores. In *2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. IEEE, Piscataway, NJ, USA, 137–146. doi:10.1109/CASES.2015.7324554

[40] Geoffrey Nelissen, David Pereira, and Luís Miguel Pinho. 2015. A Novel Run-Time Monitoring Architecture for Safe and Efficient Inline Monitoring. In *Reliable Software Technologies – Ada-Europe 2015*, Juan Antonio de la Puente and Tullio Vardanega (Eds.). Springer, Cham, Switzerland, 66–82.

[41] University of California. 2022. *Synthesizable VHDL Model of 8051*. University of California. Retrieved September 15, 2025 from http://newit.gsu.by/resources/CPUs/i8051/VHDL/Synthesizeable%20VHDL%20Model%20of%208051.htm

[42] Marco Pagani, Alessandro Biondi, Mauro Marinoni, Lorenzo Molinari, Giuseppe Lipari, and Giorgio Buttazzo. 2022. A Linux-based support for developing real-time applications on heterogeneous platforms with dynamic FPGA reconfiguration. *Future Generation Computer Systems* 129 (2022), 125–140. doi:10.1016/j.future.2021.11.007

[43] Linux perf community. 2023. *Linux profiling with performance counters*. Linux perf community. Retrieved September 15, 2025 from https://perfwiki.github.io/main/

[44] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE, Piscataway, NJ, USA, 46–57. doi:10.1109/SFCS.1977.32

[45] Luigi Pomante, Vittoriano Muttillo, Marco Santic, and Paolo Serri. 2020. SystemC-based electronic system-level design space exploration environment for dedicated heterogeneous multi-processor systems. *Microprocessors and Microsystems* 72 (2020), 102898. doi:10.1016/j.micpro.2019.102898

[46] Eberle A. Rambo, Bryan Donyanavard, Minjun Seo, Florian Maurer, Thawra Kadeed, Caio B. de Melo, Biswadip Maity, Anmol Surhonne, Andreas Herkersdorf, Fadi Kurdahi, Nikil Dutt, and Rolf Ernst. 2022. The Self-Aware Information Processing Factory Paradigm for Mixed-Critical Multiprocessing. *IEEE Transactions on Emerging Topics in Computing* 10, 1 (2022), 250–266. doi:10.1109/TETC.2020.3011663

[47] Minjun Seo and Fadi Kurdahi. 2019. Efficient Tracing Methodology Using Automata Processor. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 80 (Oct. 2019), 18 pages. doi:10.1145/3358200

[48] Minjun Seo and Roman Lysecky. 2018. Non-Intrusive In-Situ Requirements Monitoring of Embedded System. *ACM Trans. Des. Autom. Electron. Syst.* 23, 5, Article 58 (Aug. 2018), 27 pages. doi:10.1145/3206213

[49] Minjun Seo and Roman Lysecky. 2018. Work-in-Progress: Runtime Requirements Monitoring for State-based Hardware. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. IEEE, Piscataway, NJ, USA, 1–3. doi:10.1109/CODESISSS.2018.8525882

[50] Rapita Systems. 2023. *RapiTime tool*. Rapita Systems. Retrieved September 15, 2025 from https://www.rapitasystems.com/products/rapitime

[51] Salman Taherizadeh, Andrew C. Jones, Ian Taylor, Zhiming Zhao, and Vlado Stankovski. 2018. Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review. *Journal of Systems and Software* 136 (2018), 19–38. doi:10.1016/j.jss.2017.10.033

[52] Jürgen Teich. 2012. Hardware/Software Codesign: The Past, the Present, and Predicting the Future. *Proc. IEEE* 100, Special Centennial Issue (2012), 1411–1430. doi:10.1109/JPROC.2011.2182009

[53] Mark Thompson, Hristo Nikolov, Todor Stefanov, Andy D. Pimentel, Cagkan Erbas, Simon Polstra, and Ed F. Deprettere. 2007. A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs. In *2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. IEEE, Piscataway, NJ, USA, 9–14. doi:10.1145/1289816.1289823

[54] Jason G. Tong and Mohammed A. S. Khalid. 2008. Profiling tools for FPGA-based embedded systems: survey and quantitative comparison. *Journal of Computers* 3, 6 (2008), 1–14. doi:10.4304/jcp.3.6.1-14

[55] Frank Vahid and Tony Givargis. 1999. *Embedded system design: A unified hardware/software approach*. Wiley, Hoboken, NJ, USA.

[56] Giacomo Valente, Gianluca Brilli, Tania Di Mascio, Alessandro Capotondi, Paolo Burgio, Paolo Valente, and Andrea Marongiu. 2025. Fine-Grained QoS Control via Tightly-Coupled Bandwidth Monitoring and Regulation for FPGA-Based Heterogeneous SoCs. *IEEE*

Transactions on Parallel and Distributed Systems 36, 2 (2025), 326–340. doi:10.1109/TPDS.2024.3513416

[57] Giacomo Valente, Tania Di Mascio, Luigi Pomante, and Vincenzo Stoico. 2020. An ESL Methodology for HW/SW Co-Design of Monitorable Embedded Systems: the "Design for Monitorability" Project - Work-in-Progress. In *2020 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. IEEE, Piscataway, NJ, USA, 40–42. doi:10.1109/CODESISSS51650.2020.9244025

[58] Giacomo Valente, Tiziana Fanni, Carlo Sau, Tania Di Mascio, Luigi Pomante, and Francesca Palumbo. 2021. A Composable Monitoring System for Heterogeneous Embedded Platforms. *ACM Trans. Embed. Comput. Syst.* 20, 5, Article 42 (July 2021), 34 pages. doi:10.1145/3461647

[59] Simon Volpert, Philipp Eichhammer, Florian Held, Thomas Huffert, Hans P. Reiser, and Jörg Domaschka. 2023. The view on systems monitoring and its requirements from future Cloud-to-Thing applications and infrastructures. *Future Generation Computer Systems* 141 (2023), 243–257. doi:10.1016/j.future.2022.11.024

[60] Yefu Wang, Kai Ma, and Xiaorui Wang. 2009. Temperature-constrained power control for chip multiprocessors with online model estimation. In *Proceedings of the 36th Annual International Symposium on Computer Architecture* (Austin, TX, USA) *(ISCA '09)*. Association for Computing Machinery, New York, NY, USA, 314–324. doi:10.1145/1555754.1555794

[61] Xilinx. 2017. *AXI Performance Monitor Product Guide*. Xilinx. Retrieved September 15, 2025 from https://docs.xilinx.com/v/u/en-US/pg037_axi_perf_mon

[62] Xilinx. 2017. *Vivado 2017.4*. Xilinx. Retrieved September 15, 2025 from https://www.xilinx.com/support/download.html

[63] Xilinx. 2021. *Zynq Ultrascale+ MPSoC*. Xilinx. Retrieved September 15, 2025 from https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[64] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. 2013. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. Association for Computing Machinery, New York, NY, USA, 55–64. doi:10.1109/RTAS.2013.6531079

[65] Davide Zoni, Luca Cremona, Alessandro Cilardo, Mirko Gagliardi, and William Fornaciari. 2018. PowerTap: All-digital power meter modeling for run-time power monitoring. *Microprocessors and Microsystems* 63 (2018), 128–139. doi:10.1016/j.micpro.2018.07.007

[66] Davide Zoni, Andrea Galimberti, and William Fornaciari. 2023. A Survey on Run-time Power Monitors at the Edge. *ACM Comput. Surv.* 55, 14s, Article 325 (July 2023), 33 pages. doi:10.1145/3593044

Received 13 June 2024; revised 7 September 2025; accepted 15 September 2025