ASPO: Constraint-Aware Bayesian Optimization for FPGA-based Soft Processors

Haoran Wu*, Ce Guo†, Wayne Luk†, Robert Mullins*
*Department of Computer Science and Technology, University of Cambridge, Cambridge, UK
†Department of Computing, Imperial College London, London, UK
Emails: hw691@cam.ac.uk, {c.guo, w.luk}@imperial.ac.uk, robert.mullins@cl.cam.ac.uk

Abstract—**Bayesian Optimization (BO) has shown promise in tuning processor design parameters. However, standard BO does not support constraints involving categorical parameters such as types of branch predictors and division circuits. In addition, optimization time of BO grows with processor complexity, which becomes increasingly significant especially for FPGA-based soft processors. This paper introduces ASPO, an approach that leverages disjunctive form to enable BO to handle constraints involving categorical parameters. Unlike existing methods that directly apply standard BO, the proposed ASPO method, for the first time, customizes the mathematical mechanism of BO to address challenges faced by soft-processor designs on FPGAs. Specifically, ASPO supports categorical parameters using a novel customized BO covariance kernel. It also accelerates the design evaluation procedure by penalizing the BO acquisition function with potential evaluation time and by reusing FPGA synthesis checkpoints from previously evaluated configurations. ASPO targets three soft processors: RocketChip, BOOM, and EL2 VeeR. The approach is evaluated based on seven RISC-V benchmarks. Results show that ASPO can reduce execution time for the "multiply" benchmark on the BOOM processor by up to 35% compared to the default configuration. Furthermore, it reduces design time for the BOOM processor by up to 74% compared to Boomerang, a state-of-the-art hardware-oriented BO approach.**

Index Terms—**FPGA design parameter optimization; Bayesian optimization; Processor customization**

I. Introduction

Soft processors are instruction processors whose architecture and behavior are captured in software. They can be deployed on reconfigurable fabrics such as FPGAs [1] and support different application programs without lengthy place-and-route. A soft processor is a parametric hardware design where the characteristics of microarchitectural components, such as the number of cache lines and the choice of the division circuit, are determined by a set of tunable configuration parameters defining the design space for the soft processor. The design process involves the selection of appropriate values for each parameter, enabling it to be optimized for specific tasks.

Manually optimizing soft processors for enhanced performance in specific tasks is impractical due to their vast and complex design space. This complexity arises from the high dimensionality of the parameter space and the non-linear relationships between parameters and performance. Additionally, many parameters are interdependent, requiring careful coordination to achieve optimal configurations. The challenge is further compounded by the lengthy evaluation process, which involves FPGA synthesis and significantly limits the number of configurations that can be tested during the tuning process.

To address these issues, we introduce ASPO, **A**utomated **S**oft **P**rocessors **O**ptimization, a soft processor parameter optimization platform based on Bayesian Optimization (BO). A critical insight of this study is that Bayesian optimization cannot directly deal with categorical parameters, processor-specific parameter constraints, and the slow design evaluation procedure with FPGA synthesis. The key novelty is that we modify the mathematical form of BO to cope with these shortcomings rather than using BO directly. Compared with existing BO approaches for soft processors, ASOP involves the following new components: (i) customized BO covariance kernel to consider categorical parameters, (ii) smooth functions to encode parameter constraints, (iii) speed-aware BO acquisition function to speed up evaluation.

The development of ASPO involves addressing several key challenges.

- **Challenge C1:** The design space of FPGA-based soft processors includes numerous categorical parameters, including types of branch predictors and division circuits. These categorical parameters are particularly hard to optimize because they lack a natural order or continuity, making them incompatible with standard BO methods [2] and conventional constraint-handling techniques.
- **Challenge C2:** Various types of constraints need to be imposed among the design parameters [3]. Allowing designers to specify these constraints is particularly valuable in FPGA designs, where strict resource limitations and architectural requirements must be adhered to. While these constraints are typically encoded as functions with Boolean outputs, standard BO methods require constraint functions with continuous and smooth ranges, making direct integration of Boolean constraints difficult.
- **Challenge C3:** The optimization process requires evaluating numerous design parameter configurations, which is particularly time-consuming due to the need for accurate evaluation through FPGA synthesis. This exhaustive process significantly increases computational overhead, especially when configurations with minimal variations are repeatedly generated and evaluated.

The key contributions of this work are outlined as follows:

- A BO workflow with a customized kernel for categorical parameters, enhancing the efficiency of the optimization process, addressing Challenge C1. This technique is discussed in Section III-A.
- An innovative method involving the disjunctive form to enable BO to identify all types of constraints during its generation phase, ensuring that the resulting soft processor configuration complies with its specifications, addressing Challenge C2. This method is elaborated in Section III-B.
- A novel checkpoint reuse method that incorporates a similarity-based matching function into the BO acquisition process, enabling dynamic selection of prior configurations to accelerate design evaluation. This approach addresses Challenge C3 and is detailed in Section IV, as shown in Figure 2.

The ASPO optimization platform, comprising the Bayesian optimizer, evaluation framework, and scripts for replicating the experiments presented in this paper, is made available as open-source software[1].

II. Background

Optimizing soft processors increasingly relies on black-box techniques such as Bayesian Optimization (BO) [3], [4]. This section outlines the core principles behind these strategies.

Soft processor optimization is inherently a black-box problem, involving complex and opaque relationships between design parameters, performance, power, and resource usage. The lack of explicit models and gradient information makes gradient-based methods impractical, requiring designers to rely on and time-consuming simulation or expensive empirical testing.

BO [5] addresses these issues using a probabilistic surrogate model to approximate the objective function. Let $\mathbf{x}$ denote a parameter configuration and $f(\mathbf{x})$ a performance metric such as execution time or clock frequency. A Gaussian Process (GP) [6] is commonly used to provide probabilistic predictions of $f(\mathbf{x})$. An acquisition function then guides the selection of the next configuration by balancing exploration and exploitation. This process repeats until convergence or a stopping criterion is met.

BO has demonstrated significant potential in optimizing processor design parameters. Table I summarizes and compares the key features of Bayesian optimization methods for tuning processor designs over the past five years, including FIST [7], BOOM-Explorer (BE) [3], Boomerang [4], Orbit-ML [8], RCBO [10], and our proposed method. Furthermore, the workflows of four representative methods are illustrated in Figure 1.

Besides processor optimization, BO has been applied beyond architectural parameter tuning to various aspects of hardware and software optimization on FPGA platforms. For instance, BO has been employed to refine computational modules, such as in [11], where it is used to generate approximate multipliers that balance accuracy and hardware efficiency. Similarly, [12] demonstrates the application of BO to determine optimal layer sparsity in FPGA-based object detectors, showcasing its adaptability across domains. Additionally, BO has been leveraged for automation at multiple levels of FPGA design, including routing [13] and High-Level Synthesis (HLS) code optimization [14], [15].

III. Integrating Categorical Parameters and Parameter Constraints into Optimization

Bayesian Optimization (BO) methods used in prior work [3], [4], [10] do not support the parameter constraints defined in the soft processor's design specifications. Consequently, the optimizer may produce invalid designs that fail during configuration, simulation, or FPGA synthesis, or exceed resource limitations. Evaluating these invalid designs reduces productivity and wastes computational resources. Examples of such parameter constraints are presented in Table III.

To address this, we propose three types of parameter constraint-checking functions, which are used to examine the validity of configuration parameter sets against three classes of constraints. These functions are integrated into the parameter selection process of the BO method. This integration allows the BO method to handle constrained optimization problems by maximizing the acquisition function while satisfying feasibility requirements [16].

Under this framework, gradient-based constrained optimization solvers, such as Sequential Least Squares Programming (SLSQP) [17] and trust-region algorithms [18], can be employed alongside the optimizer. However, the application of these methods has two key requirements. First, the domain of the constraint functions must be numeric. Second, the range of the functions must be smooth to effectively guide the search for feasible and optimal solutions.

In typical FPGA-based soft processor designs, parameter sets often include categorical design parameters, which make the domain non-numeric. Moreover, traditional constraint functions output boolean values (true or false), indicating whether a constraint is met, but these outputs lack the smoothness required for gradient-based optimization. The following two subsections address these challenges by adapting the BO covariance kernel to better handle categorical parameters and formulating constraint functions with smooth numerical ranges, respectively.

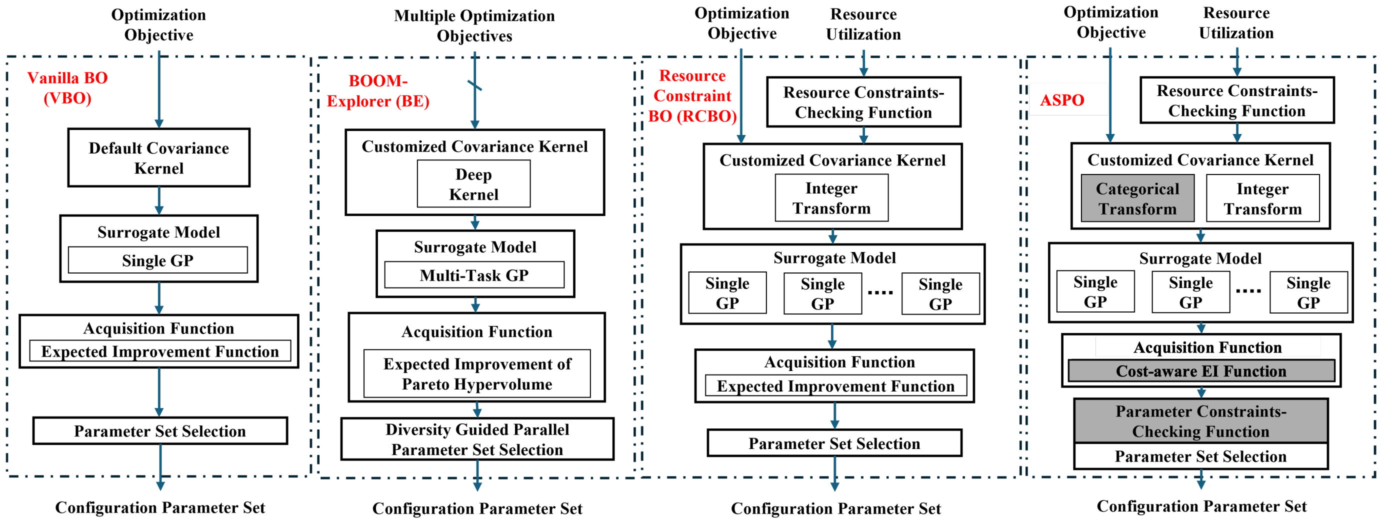
A. Adaptation for Categorical Parameters

In ASPO, we introduce a domain-specific one-hot encoding strategy tailored to soft processor design. This approach helps the BO kernel avoid redundant sampling of previously evaluated categorical configurations. Following the method in [2], each categorical parameter is expanded into a one-hot encoded

[1]https://github.com/GeorgeWu1204/Configurable-Processor-Design-Platform-and-Dataset

TABLE I
BAYESIAN OPTIMIZATION TECHNIQUES FOR TUNING PROCESSOR DESIGN PARAMETERS

Method	FIST	BE	Boomerang	Orbit-ML	VBO	RCBO	ASPO
Reference	[7]	[3]	[4]	[8]	[9]	[10]	This paper
Year	2020	2021	2023	2023	2024	2024	2025
Target platform	ASIC	ASIC	ASIC	Generic	ASIC	FPGA	FPGA
Surrogate model	XGBoost	GP	GP	GP	GP	GP	**Enhanced GP for categorical parameters**
Acquisition function	Standard	Standard	Standard	Standard	Standard	Standard	**Penalized with FPGA synthesis speed**
Number of out-of-box processors	1	1	1	1	2	1	**3**
Application-specific optimization	No	No	No	**Yes**	No	**Yes**	**Yes**
Parameter constraint awareness	No	No	No	No	No	No	**Yes** (Section III)
Interface for arbitrary FPGA-based processor	**Yes**	No	No	**Yes**	No	No	**Yes** (Section IV, illustrated in Fig. 2)
Accelerated design evaluation for FPGA	No	No	No	No	No	No	**Yes** (Section IV, illustrated in Fig. 3)
FPGA resource awareness	No	No	No	No	No	**Yes**	**Yes**
Performance evaluation with physical layout	No	No	**Yes**	No	No	No	**Yes** (Section V-C)
Open-source software	No	**Yes**	No	**Yes**	No	**Yes**	**Yes**



Optimization Objective
Vanilla BO (VBO)
Default Covariance Kernel
Surrogate Model
Single GP
Acquisition Function
Expected Improvement Function
Parameter Set Selection
Configuration Parameter Set
Multiple Optimization Objectives
BOOM-Explorer (BE)
Customized Covariance Kernel
Deep Kernel
Surrogate Model
Multi-Task GP
Acquisition Function
Expected Improvement of Pareto Hypervolume
Diversity Guided Parallel Parameter Set Selection
Configuration Parameter Set
Optimization Objective
Resource Utilization
Resource Constraint BO (RCBO)
Resource Constraints-Checking Function
Customized Covariance Kernel
Integer Transform
Surrogate Model
Single GP
Single GP
Single GP
Acquisition Function
Expected Improvement Function
Parameter Set Selection
Configuration Parameter Set
Optimization Objective
Resource Utilization
ASPO
Resource Constraints-Checking Function
Customized Covariance Kernel
Categorical Transform
Integer Transform
Surrogate Model
Single GP
Single GP
Single GP
Acquisition Function
Cost-aware EI Function
Parameter Constraints-Checking Function
Parameter Set Selection
Configuration Parameter Set

Fig. 1. This comparison evaluates the workflows of ASPO, RCBO [10], BOOM-Explorer [3], and vanilla BO [9]. Compared to RCBO, ASPO incorporates an additional categorical transformation within the customized covariance kernel, optimizing its handling of categorical parameters. Additionally, ASPO introduces a parameter constraints-checking function to address inter-parameter constraints. In contrast, BOOM-Explorer is designed for multi-objective scenarios but does not account for resource constraints or parameter dependencies and is not optimized for effectively handling categorical and integer parameters. Lastly, ASPO also introduces a cost-aware Expected Improvement (EI) acquisition function, enhancing the efficiency of configuration selection during the BO process.

vector, with each element treated as an independent real-valued variable bounded within $[0, 1]$.

Unlike [2], ASPO further customizes the covariance kernel $K(\mathbf{x}, \mathbf{x}')$ to handle categorical variables in a way that reduces redundant design evaluations. In particular, we apply a masking transformation that restores the one-hot structure before computing the kernel, ensuring that all variants of a previously evaluated category are treated as equivalent. This customized kernel plays a critical role in computing the covariance function $\sigma^2(\mathbf{x}, \mathbf{x}')$, which measures similarity between parameter configurations and guides the acquisition function in selecting the next candidate [19].

The kernel customization is to introduce a categorical transformation for the one-hot encoded vectors derived from categorical parameters before the covariance kernel computation. This transformation sets the maximum value within each one-hot encoded vector to one and all remaining elements to zero. As a result, samples that map to the same one-hot vectors share the same customized kernel function value $K'(\mathbf{x}, \mathbf{x}')$. Consequently, when a sample is evaluated, the covariance function $\sigma^2(\mathbf{x}, \mathbf{x}')$ for any sample that can be transformed to the same one-hot vector becomes zero [10]. This causes the acquisition function to exclude these regions from further sampling, thereby effectively preventing the repetitive selection of identical soft processor designs and enhancing the efficiency of the BO process.

B. Constraint Functions with Smooth Numeric Ranges

There are three main types of parameter constraint, either from the specification [3] or from the empirical experiment. Some example constraints for the BOOM processor are listed in Table III.

To formulate these three constraint-checking functions, we define that a negative output from either function signifies unmet constraints, while a non-negative output indicates that the constraints are satisfied. For the inequality constraint-

TABLE II
PREDEFINED DESIGN SPACE FOR ROCKETCHIP AND BOOM PROCESSORS IN THE EXPERIMENT

Processor	Parameter	Description	Values	Default Config
EL2 VeeR	icache size	Instruction Cache Size in KB	{16, 32, 64, 128, 256}	16
	lsu_stbuf_depth	Number of store operations the LSU buffer can hold simultaneously	{2, 4, 8}	4
	btb_enable	Boolean parameter deciding whether to include BTB.	{True, False}	True
	iccm_size	Instruction closely coupled memory size	{4, 8, 16, …, 512}	64
	dccm_size	Data closely coupled memory size	{4, 8, 16, …, 512}	64
RocketChip & BOOM	core_num	The number of cores.	{1, 2, 3, 4}	1
	icache_nSets	Number of sets in the set-associative icache.	{2, 4, 8, 16, 32, 64}	64
	icache_nWays	Number of ways in each set of the icache.	{2, 4, 8, 16, 32, 64}	4
	dcache_nSets	Number of sets in the set-associative dcache.	{2, 4, 8, 16, 32, 64}	64
	dcache_nWays	Number of ways in each set of the dcache.	{2, 4, 8, 16, 32, 64}	4
RocketChip	mul_div_config	Configuration of the multiplication and division unit.	{`Fast`, `Default`, `Simple`}	`Default`
	btb_config	Configuration of the branch target buffer.	{`Default`, `WithoutBTB`}	`Default`
BOOM	bpd_config	Configuration of the branch predictor.	{`TAGEL`, `Boom2`, `Alpha21264`}	`TAGEL`
	FetchWidth	The number of instructions the fetch unit can retrieve per cycle.	{1, 4, 8}	4
	DecodeWidth	The number of instructions the decode unit can process simultaneously.	{1, 2, 3, 4, 5, 6}	1
	RobEntry	The number of reorder buffer entries.	{32, 64, 96, 128, 120}	32
	FetchBufferEntry	The number of entries in the instruction fetch buffer.	{8, 16, 24, 32, 35, 40}	16

Note. Selected Design space sizes: 1,920 (VeeR), 31,104 (RocketChip), 8,398,080 (BOOM).

TABLE III
EXAMPLE CONSTRAINTS OF BOOM DESIGN SPECIFICATION

Classification	Descriptions
Inequalities	FetchWidth $\geq$ DecodeWidth FetchBufferEntry $>$ FetchWidth
Conditional	if icache_nWays $\in [64, 128]$, then nSets $\in [2, 4]$ $\{$ if dcache_nWays $\in [16, 32]$, then nSets $\in [2, 4]$; or if dcache_nWays $\in [128, 256]$, then nSets $\in [4, 8]$ $\}$
Divisibility	RobEntry $\mid$ DecodeWidth FetchBufferEntry $\mid$ DecodeWidth

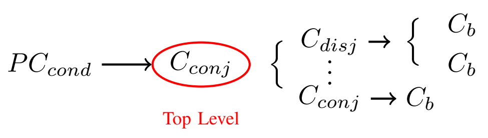
checking function, denoted as $PC_{inequality}(x)$, all inequality constraints are generalized into the following format:

$$PC_{inequality}(x) = k_a x_a - k_b x_b + t \tag{1}$$

where k_a, k_b, and t are real-valued parameters that are automatically assigned based on inequality constraints derived from the processor's specifications.

In addition to inequality constraints, ASPO also supports non-linear conditional constraints. Their relationships can be categorized as either conjunctive or disjunctive. A conjunctive structure consists of multiple conditional constraints and is satisfied only when all constraints are met. In contrast, a disjunctive structure is satisfied if at least one of its conditional constraints is true.

With minimal human effort, all the conditional constraints in the processor's specification can be systematically organized into a hierarchical logical structure comprising conjunctive and disjunctive components. At the top level, a conjunctive constraint combines all the underlying conditional constraint structures. This hierarchical organization serves as the target for our proposed constraint-checking function, $PC_{cond}(x)$, which evaluates the constraints as follows:



The single basic conditional constraint example format and the corresponding constraint-checking function, C_b are illustrated below:

- If x_1 lies within the interval $[a_1, b_1]$, then x_2 must lie within the interval $[a_2, b_2]$.

$$C_b(x) = \min_m c_m(x_m), \quad \forall m \in \{1, \ldots, d\} \tag{2}$$

$$c_m(x_m) = -(x_m - a_m)(x_m - b_m) \tag{3}$$

Here, each $c_m(x_m)$ is differentiable, ensuring sufficient smoothness to support the Bayesian optimizer.

For a disjunctive structure of constraints, the constraint-checking function, C_{disj}, is defined as:

$$C_{disj} = \max_i C_i(x), \quad \forall i \in \{1, \ldots, k\}, \tag{4}$$

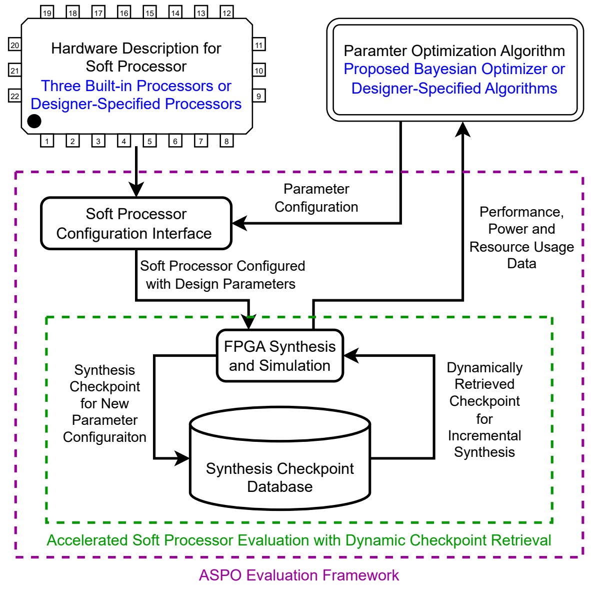
where each $C_i(x)$ represents the i-th conditional constraint of the k conditional constraints of the disjunctive structure. It can be C_b, C_{conj}, or C_{disj}. Given that these functions are defined such that a nonnegative output indicates that the input x satisfies the evaluated constraint, taking the maximum value in C_{disj} ensures that if at least one $C_i(x)$ returns a nonnegative value, the disjunctive constraint structure is satisfied.

Conversely, for a conjunctive structure of constraints, the constraint-checking function, C_{conj}, is determined by taking the minimum value among all conditional constraints within the structure:

$$C_{conj} = \min_i C_i(x), \quad \forall i \in \{1, \ldots, k\} \tag{5}$$

This approach ensures that if the smallest output is negative, at least one of the conditional constraints is violated, indicating that the conjunctive constraint is not satisfied.

For the third type of constraint, which involves checking the divisibility between parameters, we propose the following checking function:



Hardware Description for Soft Processor
Three Built-in Processors or Designer-Specified Processors
Paramter Optimization Algorithm
Proposed Bayesian Optimizer or Designer-Specified Algorithms
Parameter Configuration
Soft Processor Configuration Interface
Performance, Power and Resource Usage Data
Soft Processor Configured with Design Parameters
FPGA Synthesis and Simulation
Synthesis Checkpoint for New Parameter Configuraiton
Dynamically Retrieved Checkpoint for Incremental Synthesis
Synthesis Checkpoint Database
Accelerated Soft Processor Evaluation with Dynamic Checkpoint Retrieval
ASPO Evaluation Framework

Fig. 2. Overview of the proposed framework for accelerated soft processor design and evaluation. The framework supports three built-in processors or designer-specified processors, which can be configured using a parameter optimization algorithm, such as the proposed Bayesian optimizer or alternative designer-specified methods. Design parameters are iteratively optimized to achieve desired performance objectives. The evaluation framework integrates FPGA synthesis and simulation tools, using a dynamic checkpoint retrieval system to reuse synthesis data from previously evaluated configurations, significantly accelerating the evaluation process.

$$PC_{\text{divisibility}}(x) = -\sin^2\left(\frac{\pi x_a}{x_b}\right) \quad (6)$$

The function $PC_{\text{divisibility}}(x)$ returns zero only when x_b is divisible by x_a; otherwise, it returns a negative value. The use of a trigonometric function ensures differentiability for the BO optimizer, while its periodic nature guarantees that the function's value depends solely on whether $\frac{x_a}{x_b}$ is an integer and not on its magnitude.

IV. Accelerated Soft Processor Evaluation

The primary goal of our soft processor evaluation framework is to provide a rapid and automated assessment of the performance of specified soft processor configurations. Since the FPGA synthesis process is often a significant bottleneck in the optimization workflow, this framework adopts a data-driven approach to accelerate the evaluation process. The detailed workflow is illustrated in Figure 2.

A. Checkpoint Retrieval for Fast Design Evaluation

During the processor design process, design methods often produce configurations that share several components with previously evaluated processors. This overlap allows us to store key information from the evaluation of earlier configurations and reuse it to expedite the evaluation of new designs.

While tools like Vivado and Quartus Prime support incremental synthesis at a low level, they do not provide a mechanism for checkpoint selection or reuse across diverse design '2. Our proposed method includes a novel way that dynamically constructs a checkpoint database and selects the closest prior configuration using a weighted similarity metric. This enables us to automatically identify optimal reuse opportunities across the design space.

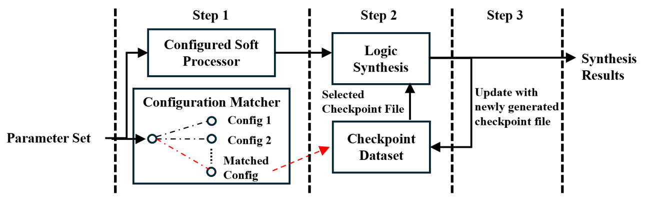


Fig. 3. Accelerated evaluation flow.

To take advantage of the incremental synthesis feature for faster evaluation, two key components are required. The first is a database of checkpoints to store designs that can serve as starting points for incrementally synthesizing new configurations. The second is a matching function to identify an appropriate starting point from the database, minimizing potential synthesis time by reusing previously synthesized configurations.

We propose dynamically building the checkpoint database during the optimization process. This database stores checkpoint files generated from previously synthesized processor configurations, allowing efficient reuse in subsequent synthesis tasks. For reference checkpoint selection, we introduce a configuration matcher that identifies the checkpoint file with the highest overlap with the processor currently under evaluation. By minimizing the number of modified components requiring re-synthesis, this approach significantly reduces synthesis time. The flow of the accelerated evaluation process is illustrated in Figure 3.

To implement the reference checkpoint selection, we develop a configuration matching function that identifies the most similar configuration in the dataset based on a weighted Euclidean distance metric. The mathematical formulation of the matching function is as follows:

$$\text{MatchConfig}(\mathbf{x}, \mathbf{w}, Q) = \underset{\mathbf{q}\in Q}{\text{argmin}} \sum_{i=0}^{d-1} w_i \cdot (x_i - q_{ni})^2 \quad (7)$$

In this equation, x_i represents the i-th parameter of the configuration $\mathbf{x}$ being evaluated, and q_{ni} denotes the i-th parameter of the n-th stored configuration $\mathbf{q}$ within the checkpoint dataset Q. The weight vector $\mathbf{w} \in \mathbb{R}^d$ quantifies the relative importance of changes in each parameter for configuration matching. We aim to compute the optimal weights $\mathbf{w}^*$ by solving the following optimization problem:

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathbb{R}^d}{\text{argmin}} \sum_{\mathbf{q}\in Q} \mathcal{T}_{\text{syn}}(\mathbf{q}, \text{MatchConfig}(\mathbf{q}, \mathbf{w}, Q \setminus \mathbf{q})) \quad (8)$$

Here, $\mathcal{T}_{\text{syn}}(\mathbf{x}, \mathbf{y})$ represents the synthesis time required to process configuration $\mathbf{x}$ using the checkpoint from configuration $\mathbf{y}$. This optimization determines the optimal weight vector $\mathbf{w}^*$ by minimizing the total synthesis time across all configurations in the dataset Q. Given the substantial size of the dataset and the lengthy synthesis time for each configuration, an approximate solution is obtained by randomly sampling a small subset of design configurations and applying heuristic search techniques.

B. Warm Start

The evaluation process in our proposed framework becomes increasingly efficient as more soft processor configurations are evaluated. With each evaluated configuration, the checkpoint database grows with an additional checkpoint file. This expansion enables the configuration matcher to identify checkpoint files from previously evaluated configurations with greater overlap to the current configuration being tested, thereby accelerating the evaluation process. Additionally, a larger database increases the likelihood that the performance of a new configuration has already been recorded, eliminating the need for re-evaluation.

To achieve high evaluation speeds, implementing a warm-start feature that strategically guides the sampling of processor configurations is essential. This feature ensures that sampled configurations are uniformly distributed across the entire design space, maximizing the chances that any new configuration to be evaluated will have a closely matched configuration in the database, thus improving evaluation efficiency.

Traditional sampling strategies, such as random sampling, are avoided due to their inefficiency in achieving uniform distribution across the design space. This inefficiency arises because the design space is vast and consists of multiple parameters, each with its own distinct range of possible values. To address this challenge, the orthogonal array (OA) strategy is adopted. The orthogonality of OA ensures that sampled configurations are evenly distributed throughout the design space [20]. Since the design space for the soft processor is fixed within our framework, the OA can be pre-generated using online methods, further simplifying implementation.

C. Incorporating Evaluation Time Cost Into Optimization

Due to the proposed accelerated soft processor evaluation method, the evaluation time of a configuration depends on how similar it is to previously evaluated configurations stored in the checkpoint database. Therefore, we incorporate the consideration of evaluation time in the BO to optimize configuration selection and reduce the total evaluation time.

To achieve this, we incorporate a cost-aware cooling mechanism into the BO acquisition function, steering configuration selection in the BO process for optimized overall evaluation time. The cost function, $\hat{c}(x)$, is defined as the minimum weighted Euclidean distance between the candidate configuration and all previously evaluated configurations stored in the checkpoint database, same as Equation 7. Inspired by [21], the modified cost-aware acquisition function is defined as:

$$\alpha_{\text{cool}}(x, t) = \frac{\alpha(x)}{\lambda(t) \cdot \hat{c}(x)} \tag{9}$$

where $\alpha(x)$ is the original acquisition function (Expected Improvement in this project), and $\lambda(t)$ denotes a cooling schedule that evolves over the current BO iteration t. In this work, we adopt a simple decaying schedule designed to impose a high penalty on distant configurations in the early optimization stages. This encourages the algorithm to remain in regions that are quick to evaluate. As the optimization progresses, the penalty is gradually reduced, allowing the algorithm to explore more distant, potentially expensive but promising configurations. The cooling schedule is defined as:

$$\lambda(t) = \lambda_0 \cdot \exp(-kt) \tag{10}$$

where λ_0 is the initial cost sensitivity, and k is the decay rate; both are determined empirically through experiments.

TABLE IV
RISC-V BENCHMARKS USED IN THIS PROJECT.

Benchmark	Focus	minstret
coremark [22]	Basic operations	282995
dhrystone [23]	Integer arithmetic and string handling	186031
rsort [24]	Memory access	171154
qsort [24]	Branching, recursion and memory access	123506
multiply [24]	Multiplication	42503
spmv [24]	Floating point and memory access	34466
mm [24]	Cache and memory access	24744

V. EVALUATION

We conducted two experiments to assess the performance of ASPO. The first experiment aims to evaluate the acceleration improvements offered by the automated soft processor evaluation framework in Section IV. The second experiment focused on evaluating the overall performance of our proposed soft processor design platform that combines all features in Sections III and IV.

A. Experiment Setup

Our experimental implementation of the proposed framework supports three soft processors out-of-box: RocketChip [25], BOOM [26], and EL2 VeeR [27]. These processors were selected for their high performance, extensive design flexibility, and support for FPGA deployment. With minor modifications, the implementation can be extended to support other configurable RISC-V soft processors, such as Ibex [28] and CVA6 [29].

All experiments were conducted on a desktop workstation equipped with an Intel Xeon E5-1650 v3 CPU and 32GB of DDR4 RAM, without utilizing a GPU. GPU acceleration was deemed unnecessary, as the Bayesian optimizer contributes only a small portion of the total runtime and would not benefit significantly from GPU usage. The evaluation phase, primarily involving processor simulation using Verilator and FPGA hardware synthesis using Vivado, is the most time-consuming step and cannot be accelerated by a GPU.

The evaluation includes simulation results for multiple RISC-V benchmarks supported by soft processors, such as dhrystone and coremark, as listed in Table IV. These benchmarks are designed to evaluate the processor's performance across various processing aspects. The complexity of each benchmark is measured by the machine instructions retired (minstret) [30], representing the total number of instructions executed by the processor.

The results for resource utilization, power, and timing metrics are derived from the report generated from the synthesis process. The timing metric is assessed by manually configuring the processor's clock input to a default reference frequency of 50 MHz and extracting the worst setup slack and worst hold slack values from the Vivado timing report. These two values offer insights into the timing performance of the soft processor at this default frequency and can be used to estimate its maximum operating frequency.

B. Experiment for Soft Processor's Evaluation Framework

This experiment focuses on evaluating the time required to assess a set of soft processor configurations using three different frameworks. The evaluation time is defined as the duration from receiving the configuration parameters to the completion of all performance metrics, including execution time, resource utilization, power consumption, and timing performance.

We compare three evaluation approaches. The first, direct evaluation, reflects the standard flow used in methods such as RCBO [10], where each configuration is synthesized and evaluated independently from scratch. The second, evaluation with a fixed checkpoint, uses a single checkpoint derived from the default soft processor configuration for all evaluations, disabling both the configuration matcher and checkpoint dataset; the default parameters are listed in Table II. The third is our proposed framework, which dynamically builds a checkpoint database and uses a similarity-based configuration matcher with a sufficiently diverse pool, enabling efficient reuse of prior synthesis results to accelerate evaluation.

To ensure a fair comparison, all three evaluation frameworks are provided with an identical set of soft processor configurations. In this experiment, ten randomly generated configurations were created for each supported soft processor, serving as the candidate experiment set. Experimental results are presented in Figure 4, and the quantitative comparison among the three frameworks is shown in Table V.

C. Experiment for Overall Design Platform

To rigorously evaluate the design platform's performance, this experiment outlines a series of application benchmarks. These tasks involve configuring the three supported soft processors within the predefined design space outlined in Table II, optimizing their performance using selected RISC-V benchmarks listed in Table IV, and ensuring deployability on the target FPGA boards. The *Zynq UltraScale+ XCZU3CG* FPGA, offering 70,560 available LUTs, is selected as the target platform for EL2 VeeR, while the *Zynq UltraScale+ XCZU6CG*, with 214,604 available LUTs, is chosen for BOOM.

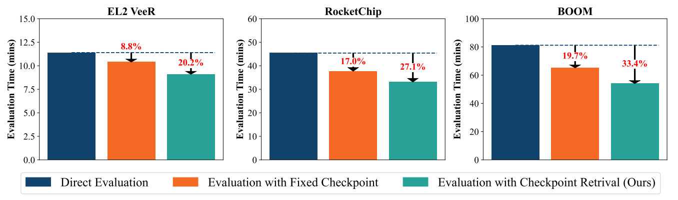


Fig. 4. The two frameworks employing acceleration during the logic synthesis stage can significantly enhance evaluation speed by reusing existing evaluation data. Additionally, our proposed framework achieves faster evaluations through a strategic selection of checkpoint files, in contrast to frameworks that consistently utilize the same checkpoint file.

TABLE V
EVALUATION TIME FOR DESIGN EVALUATION (IN MINUTES)

Processor	Evaluation Method	Avg	Min	Max
EL2 VeeR	Direct evaluation	11.4	11.2	11.5
	Evaluation with fixed checkpoint	10.4	8.5	11.1
	Our framework	9.1	8.4	10.1
RocketChip	Direct evaluation	45.4	44.7	46.2
	Evaluation with fixed checkpoint	37.7	24.0	43.9
	Our framework	33.1	18.0	37.5
BOOM	Direct evaluation	81.2	80.1	81.9
	Evaluation with fixed checkpoint	65.2	57.5	71.2
	Our framework	54.1	42.5	63.4

The performance comparison of our design platform with other methods focuses on design productivity and design quality. The design productivity is evaluated using two metrics: **Invalid Design Rate (IDR)** and **Total Design Time (TDT)**. IDR quantifies the proportion of designs that fail to meet either the processor's parameter constraints or the FPGA's resource constraints during the design process. TDT measures the total time, in hours, required for the design method to converge to the final processor configuration. While valid processors typically undergo longer evaluations, invalid processors may trigger errors at various stages, leading to variability in evaluation times. To ensure practical feasibility, an upper limit of **35 hours** is set for TDT, and the experiment is terminated upon reaching this time bound. This limit is selected based on the observation that most experiments are completed within this limit, and it is used to ensure fair comparison across different design configurations or strategies.

Design quality is assessed using **Estimated Execution Time (EET)** [4], which serves as the optimization objective in this experiment. EET estimates the processor's execution time for the designated tasks on FPGA fabric, providing an accurate reflection of the processor's physical layout performance [4]. It is computed by dividing the simulated number of execution cycles by the maximum operating frequency, as reported in the timing performance section of the synthesis report.

During the experiment, if a processor design method does not support parameter constraint awareness or FPGA resource limitations and generates invalid designs, such designs are discarded without being used to update the surrogate model, preventing the optimizer from being misled. Additionally, to ensure fair comparison, the initial points are fixed and selected from the sampled configurations obtained using the orthogonal array, as described in Section IV-B. Furthermore, due to the significant evaluation time and the large design space, methods requiring extensive evaluations, such as brute-force exhaustive search algorithms, acquisition-free surrogate optimization [31], genetic algorithms [32], and reinforcement learning [33], are excluded from consideration. The design methods selected for comparison include **Hill Climbing (HC) [34]**, **Vanilla Bayesian Optimization (VBO) [9]**, **BOOM-Explorer (BE) [3]**, and **Resource-Constraint Bayesian Optimization (RCBO) [10]**, as discussed in Section II.

TABLE VI
DESIGN PRODUCTIVITY

TDT: Total Design Time in Hours IDR: Invalid Design Rate

Processor	Task	Metric	HC	VBO	BE	RCBO	ASPO
EL2 VeeR	coremark	TDT	0.56†	2.61	2.18	1.93	**1.04**
		IDR	**0%**	14%	**11%**	**0%**	**0%**
	dhrystone	TDT	0.61†	3.98	2.12	1.88	**1.63**
		IDR	17%	**0%**	14%	**0%**	7%
	rsort	TDT	0.81†	1.14	3.46	**0.98**	1.01
		IDR	13%	**0%**	22%	**0%**	**0%**
	qsort	TDT	0.53†	3.43	1.33	1.02	**0.92**
		IDR	**0%**	**0%**	12%	**0%**	**0%**
	multiply	TDT	1.01	1.41	2.13	1.15	**0.72**
		IDR	20%	**0%**	13%	11%	14%
	spmv	TDT	0.44†	2.65	1.50	**1.11**	1.13
		IDR	**0%**	18%	11%	**0%**	**0%**
	mm	TDT	0.64†	1.43	1.98	1.15	**1.08**
		IDR	40%	**0%**	18%	**0%**	**0%**
RocketChip	coremark	TDT	3.56†	27.11	22.08	18.13	**11.24**
		IDR	75%	52%	38%	33%	**21%**
	dhrystone	TDT	3.00†	20.12	23.41	14.12	**11.25**
		IDR	88%	42%	48%	37%	**31%**
	rsort	TDT	4.00†	**8.43**	16.33	15.12	10.12
		IDR	71%	40%	52%	36%	**21%**
	qsort	TDT	5.30†	27.64	23.32	16.12	**11.30**
		IDR	67%	31%	39%	48%	**25%**
	multiply	TDT	3.57†	17.41	12.30	21.05	**9.71**
		IDR	71%	**30%**	33%	61%	34%
	spmv	TDT	6.02†	22.01	15.20	9.34	**8.11**
		IDR	67%	52%	31%	38%	**28%**
	mm	TDT	9.31†	21.73	24.98	16.75	**9.18**
		IDR	60%	51%	48%	54%	**23%**
BOOM	coremark	TDT	9.56†	32.11	27.48	35.00	**24.14**
		IDR	54%	63%	42%	47%	**32%**
	dhrystone	TDT	11.90†	35.00	24.92	22.18	**19.63**
		IDR	64%	57%	38%	43%	**21%**
	rsort	TDT	7.56†	35.00	29.90	26.10	**14.71**
		IDR	71%	63%	45%	50%	**25%**
	qsort	TDT	6.17†	32.13	22.13	28.12	**18.71**
		IDR	71%	72%	52%	36%	**15%**
	multiply	TDT	7.21†	31.61	35.00	**16.43**	18.23
		IDR	83%	52%	33%	**10%**	17%
	spmv	TDT	4.42†	28.15	35.00	27.96	**9.12**
		IDR	83%	58%	39%	**36%**	**36%**
	mm	TDT	9.31†	21.73	24.98	16.75	**10.18**
		IDR	60%	51%	48%	54%	**23.5%**

†TDT for HC is not compared with BO methods due to poor design quality

TABLE VII
DESIGN QUALITY MEASURED BY ESTIMATED EXECUTION TIME (EET) IN MILLISECONDS

Processor	Task	HC	VBO	BE	RCBO	ASPO	Default
EL2 VeeR	coremark	9.17	9.09	9.62	9.39	**9.18**	10.17
	dhrystone	8.76	8.68	8.48	8.35	**7.91**	9.06
	rsort	6.48	5.98	6.35	5.96	**5.82**	6.42
	qsort	5.34	5.14	5.23	**5.05**	5.16	5.35
	multiply	1.80	1.68	1.56	1.52	**1.35**	1.83
	spmv	1.55	1.52	1.47	1.42	**1.35**	1.57
	mm	1.31	1.23	1.29	**1.15**	1.16	1.31
RocketChip	coremark	8.09	8.01	7.61	7.61	**7.02**	8.09
	dhrystone	7.96	7.68	7.48	7.45	**7.41**	7.96
	rsort	5.60	5.34	5.29	5.50	**5.09**	5.61
	qsort	4.35	4.43	4.33	4.12	**4.05**	4.35
	multiply	1.33	1.30	1.26	**1.05**	1.08	1.33
	spmv	1.17	1.02	0.97	1.02	**0.93**	1.17
	mm	0.91	0.88	0.84	0.88	**0.82**	0.91
BOOM	coremark	7.17	6.89	6.62	6.79	**6.28**	7.20
	dhrystone	7.06	6.68	6.48	6.45	**6.11**	7.06
	rsort	3.55	3.45	3.45	3.46	**3.42**	3.56
	qsort	3.35	3.30	**3.12**	3.27	**3.12**	3.35
	multiply	1.07	0.86	0.79	0.74	**0.70**	1.07
	spmv	0.87	0.78	0.80	**0.73**	0.74	0.89
	mm	0.73	0.75	0.66	0.72	**0.63**	0.75

Table VI highlights the productivity of the processor design methods. Except for the HC method, which converges quickly but does not optimize the processor, ASPO achieves the best performance with the smallest Total Design Time (TDT) and lowest Invalid Design Rate (IDR) in most tasks. For the EL2 VeeR processor with fewer parameter constraints, ASPO achieves a comparable IDR to other methods while still attaining optimal Total Design Time (TDT) thanks to its accelerated evaluation framework. For the more complex RocketChip and BOOM processors with multiple constraints, ASPO significantly lowers IDR, showing its strength in generating valid configurations. As evaluation time increases, ASPO's TDT advantage becomes more pronounced. For BOOM's optimization on certain benchmarks like "spmv", it saves 67.4% TDT compared with RCBO, 74% with BE and 67.6% with VBO.

Table VII focuses on design quality. Processors optimized using ASPO outperform those designed by other methods in 17 out of 21 tasks. In the remaining tasks, ASPO achieves performance comparable to other methods, demonstrating that it consistently identifies high-performance designs. Results show that with ASPO, the optimized BOOM design reduces execution time by 34.6% on the "multiply" benchmark.

VI. Conclusions and Future Work

This paper presents ASPO, an automated design platform for efficiently optimizing soft processors for specified workload while adhering to FPGA constraints. Future work includes extending ASPO to support additional architectures and optimizations, such as System-on-Chip designs, custom instruction processors, and application-specific multi-cores.

VII. Acknowledgment

The support of the United Kingdom EPSRC (grant number UKRI256, EP/V028251/1, EP/N031768/1, EP/S030069/1, and EP/X036006/1), Altera, Intel, and AMD is gratefully acknowledged.

References

[1] J. G. Tong, I. D. L. Anderson, and M. A. S. Khalid, "Soft-core processors for embedded systems," *2006 International Conference on Microelectronics*, pp. 170–173, 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:5876189

[2] E. C. Garrido-Merchán and D. Hernández-Lobato, "Dealing with categorical and integer-valued variables in Bayesian Optimization with Gaussian processes," *Neurocomputing*, vol. 380, p. 20–35, Mar. 2020. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2019.11.004

[3] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. F. Wong, "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 29, no. 1, dec 2023. [Online]. Available: https://doi.org/10.1145/3630013

[4] Y.-F. Liu, C.-Y. Hsieh, and S.-Y. Kuo, "Boomerang: Physical-Aware Design Space Exploration Framework on RISC-V SonicBOOM Microarchitecture," in *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2023, pp. 85–93.

[5] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[6] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006, vol. 2, no. 3.

[7] Z. Xie, G.-Q. Fang, Y.-H. Huang, H. Ren, Y. Zhang, B. Khailany, S.-Y. Fang, J. Hu, Y. Chen, and E. C. Barboza, "FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 19–25.

[8] J. G. Coutinho, C. Guo, T. Todman, and W. Luk, "Exploring machine learning adoption in customisable processor design," in *International Conference on ASIC (ASICON)*. IEEE, 2023, pp. 1–4.

[9] Y. Gao, D. Luo, C. Bai, B. Yu, H. Geng, Q. Sun, and C. Zhuo, "Is Vanilla Bayesian Optimization Enough for High-Dimensional Architecture Design Optimization?" in *ICCAD*, 2024.

[10] C. Guo, H. Wu, and W. Luk, "Resource-Constraint Bayesian Optimization for Soft Processors on FPGAs," in *14th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART '24)*. Porto, Portugal: ACM, June 19-21 2024, pp. 1–13.

[11] Z. Li, H. Zhou, L. Wang, and X. Zhou, "AMG: Automated Efficient Approximate Multiplier Generator for FPGAs via Bayesian Optimization," in *2023 International Conference on Field Programmable Technology (ICFPT)*, 2023, pp. 294–295.

[12] D. T. Nguyen, H. Kim, and H.-J. Lee, "Layer-Specific Optimization for Mixed Data Flow With Mixed Precision in FPGA Design for CNN-Based Object Detectors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 6, pp. 2450–2464, 2021.

[13] S. Zheng, J. Qian, H. Zhou, and L. Wang, "Graebo: Fpga general routing architecture exploration via Bayesian optimization," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2022, pp. 282–286.

[14] H. Kuang and L. Wang, "Multi-objective design space exploration for high-level synthesis via Bayesian optimization," in *2023 International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 2023, pp. 150–155.

[15] H. Kuang, X. Cao, J. Li, and L. Wang, "Hgbo-dse: Hierarchical gnn and Bayesian optimization based hls design space exploration," in *2023 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2023, pp. 106–114.

[16] E. Brochu, V. M. Cora, and N. de Freitas, "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," *CoRR*, vol. abs/1012.2599, 2010. [Online]. Available: http://arxiv.org/abs/1012.2599

[17] D. Kraft, "A software package for sequential quadratic programming," DLR German Aerospace Center – Institute for Flight Mechanics, Köln, Germany, Tech. Rep. DFVLR-FB 88-28, 1988.

[18] R. H. Byrd, M. E. Hribar, and J. Nocedal, "An Interior Point Algorithm for Large-Scale Nonlinear Programming," *SIAM J. Optim.*, vol. 9, pp. 877–900, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:16293345

[19] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the Human Out of the Loop: A Review of Bayesian Optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[20] D. Li, S. Yao, Y.-H. Liu, S. Wang, and X.-H. Sun, "Efficient design space exploration via statistical sampling and AdaBoost learning," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[21] E. H. Lee, V. Perrone, C. Archambeau, and M. Seeger, "Cost-aware Bayesian optimization," 2020. [Online]. Available: https://arxiv.org/abs/2003.10870

[22] S. Gal-On and M. Levy, "Exploring coremark a benchmark maximizing simplicity and efficacy," *The Embedded Microprocessor Benchmark Consortium*, 2012.

[23] R. P. Weicker, "Dhrystone: a synthetic systems programming benchmark," *Commun. ACM*, vol. 27, no. 10, p. 1013–1030, Oct. 1984. [Online]. Available: https://doi.org/10.1145/358274.358283

[24] T. Newsome, "riscv-software-src/riscv-tests risc-v," https://github.com/riscv-software-src/riscv-tests, May 2019, [Online; accessed January 17, 2025].

[25] K. Asanović, R. Avižienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, P. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The Rocket Chip Generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, April 2016, if used for research, please cite Rocket Chip by the technical report.

[26] C. Celio, D. A. Patterson, and K. Asanović, "The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167, Jun 2015. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html

[27] Chips Alliance, "Cores-VeeR-EL2: A RISC-V Core," https://github.com/chipsalliance/Cores-VeeR-EL2, 2024, available online: https://github.com/chipsalliance/Cores-VeeR-EL2.

[28] lowRISCC.I.C., "Ibex: An embedded 32-bit risc-v cpu core," https://ibex-core.readthedocs.io, 2021, documentation.

[29] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.

[30] V. M. Weaver and S. A. McKee, "Can hardware performance counters be trusted?" in *2008 IEEE International Symposium on Workload Characterization*, 2008, pp. 141–150.

[31] M. Kurek, T. Becker, T. C. P. Chau, and W. Luk, "Automating Optimization of Reconfigurable Designs," in *Proceedings of the 2014 IEEE 22nd International Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '14. USA: IEEE Computer Society, 2014, p. 210–213.

[32] S.-C. Kao and T. Krishna, "Gamma: Automating the HW mapping of DNN models on accelerators via genetic algorithm," in *International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[33] C. Bai, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "Towards automated risc-v microarchitecture design with reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 1, 2024, pp. 12–20.

[34] J. Alastruey, T. Monreal, F. Cazorla, V. Viñals, and M. Valero, "Selection of the Register File Size and the Resource Allocation Policy on SMT Processors," in *2008 20th International Symposium on Computer Architecture and High Performance Computing*. Washington, DC, United States: ACM, 2008, pp. 63–70.