Dovado: An Open-Source Design Space Exploration Framework

Daniele Paletti
Politecnico di Milano, Italy
daniele.paletti@mail.polimi.it

Davide Conficconi
Politecnico di Milano, Italy
davide.conficconi@polimi.it

Marco D. Santambrogio
Politecnico di Milano, Italy
marco.santambrogio@polimi.it

***Abstract*—Traditional hardware development exploits description languages such as VHDL and (System)Verilog to produce highly parametrizable RTL designs. Different parameter values yield different utilization-frequency trade-offs, and hand-tuning is not feasible with a non-trivial amount of parameters. Generally, the Computer-Aided Design (CAD) literature proposes approaches that mainly tackle automatic exploration without combining a design automation feature. Hence, this work proposes Dovado, an open-source CAD tool for design space exploration (DSE) tailored for FPGAs-based designs. Starting from VHDL/(System)Verilog, Dovado exploits Vivado and supports the hardware developer for an exact exploration of a given set of parameters or a DSE where it returns the non-dominated set of configuration points. In this work, we exploit a multi-objective integer formulation and Non-Dominated Sorting Genetic Algorithm (NSGA)-II for a fast DSE. Moreover, we propose an approximation model for the NSGA-II fitness function to decide whether Vivado or a Nadaraya-Watson model should estimate the optimization metrics.**

***Index Terms*—Design Space Exploration, Design Automation, FPGAs, Approximation Model, Mutli-Objective Optimization**

I. Introduction

The Register Transfer Level (RTL) defines input-output relationships in terms of dataflow operations on signals, and register values [1]. Often RTL-based designs encode in Hardware Description Languages (HDLs) such as VHDL and Verilog/SystemVerilog (V/SV). Such languages give the designer the possibility of specifying parameters for each RTL module (e.g., VHDL generics and SV parameters) to provide a reusable module for multiple use cases. However, the volume of the parameters space (i.e., their possible permutations) is factorial in the number of parameters. Hence hand-tuning them may be unfeasible even in low-dimensional spaces. A designer is often interested in tuning module parameters to achieve the optimal area-frequency trade-off, which is often known after evaluating specific design points through the hardware design flow steps. The typical hardware design flow of an Electronic Design Automation (EDA) tool, for both Field Programmable Gate Arrays (FPGAs) and Application Specific Circuits (ASICs), starts with logic synthesis (i.e., translating RTL into logic gates with the technology mapping) and follows with the place and route phase on the target device (also called implementation). Within this context, the designer must manually run synthesis or implementation routines and then hand-tune the parameters for a given metric to optimize. However, synthesis and implementation are time-consuming for non-trivial designs, for this reason, manual Design Space Exploration (DSE) is unfeasible. On a different wave, many emergent HDLs try to tackle the tricky generation of highly parametrizable RTL-based designs [2], [3]. Indeed, they tremendously ease the generation of highly customizable modules and enable the birth of System on Chip generators [4], [5]. However, architecture modeling is often case-specific [6], and the support for automatic DSE is still missing. Nowadays, Computer-Aided Design (CAD) tools are vital to increase productivity, ensure correctness and performance of intricate designs, and enable a higher level of complexity. Many tools such as Aladdin [7], Heracles [8] and SystemCoDesigner [9] have been developed to tackle DSE but none of those provide a comprehensive approach to be used both for design automation and exploration of RTL designs.

For these reasons, we propose Dovado [10], an open-source tool that leverages an already existing hardware suite to provide design automation and automatic DSE for RTL parameters, currently tailored for FPGAs-based designs. Starting from an RTL hierarchy, a hardware developer can specify a set of design points, i.e., a set of free parameters, and then Dovado evaluates them in terms of maximum achievable frequency and/or user-defined area usage metrics, e.g., LUTs, RAMs, Array Cells. Dovado provides a methodology to extract these results from one of the typical design steps, synthesis or implementation, and automatically, or in case manually, evaluating the metrics.

Moreover, Dovado builds on top of the automation feature to provide a design exploration functionality. Indeed, we iterate single-point evaluation with several optimization strategies to compute the non-dominated set of points in DSE mode. This set is composed of points such that no further point exists among the solutions, which is better on all objectives. Additionally, Dovado exploits a model that, thanks to a synthetic dataset, predicts synthesis/implementation outcomes, which can highly reduce the exploration time for a complex module with a huge search space.

In summary, the contributions of this work are:

- a design automation methodology for the single design

DOI 10.1109/IPDPSW52791.2021.00027

point evaluation on FPGA devices (Section III-A)
- a mathematical formulation of a design space exploration problem as multi-objective problem (Section III-B) that enables an approximation model for synthesis and implementation outputs and the associated control model (Section III-C)
- the whole framework is open-sourced for contributions and usage, and available as a python package [10]

The remainder of the paper is organized as follows: Section II provides an overview of how some of the literature works address the automation and exploration issues. Then, Section III presents Dovado with design automation (Section III-A) and exploration (Section III-B) methodologies. Section IV describes the experimental setup and discusses the results obtained with Dovado, while Section V draws the conclusions.

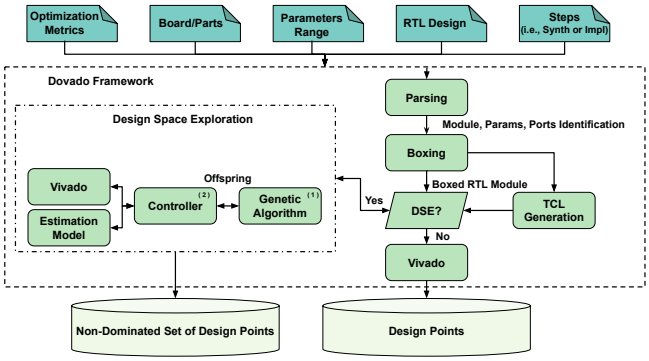

Figure 1: Dovado framework block diagram with design automation and design space exploration flows.

II. Related Work

Traditionally, the DSE term is used in HW/SW codesign methodologies as the configuration choice of a target system to find an optimal trade-off among figures of merit (e.g., area, delay) according to system architecture choices and HW/SW partitioning [11]. Indeed, optimizing one of the metrics usually implies the degradation of the other(s), and DSE problems find their solutions in an optimization problem, as shown in the literature [12]. Instead, in this work, we use a different DSE flavor at the RTL-module level, and we use it to find design parameters trade-offs according to figures of merit.

Starting with a more traditional approach, Hammerquist et al. [13] devised the concept of application-specific FPGAs (AFPGAs) to fill the gap between FPGAs and Application Specific Integrated Circuits (ASICs), which are much more expensive in terms of time and money to develop and implement. The exploration problem is defined as a search among architectural features to be customized. For instance, we might consider 3-input, 4-input, and 5-input LUTs, enabling different architectural level parametrizations.

Karakaya [14] tackles the problem of finding an efficient algorithm for DSE. The author treats the problem as an integer single-target optimization targeting power-delay-area product. The author adopts a heuristic to solve the problem, showing remarkable performance when parallelized; however, this hinders the optimization of metrics separately.

Kao et al. [15] used a commercial synthesis and implementation tool to tackle RTL DSE, formulating it as a module selection problem by determining all possible design implementations with different area-time curves. Modules contain either combinational or sequential logic, but they are not considered singularly, and the area-time curve is assessed on the whole system. The authors do not consider that more complex designs expose prohibitive time costs for a good DSE. Thus, others aim at finding alternative solutions to really executing all the synthesis or implementation steps. Indeed, So et al. [16] estimate the synthesis process through behavioral synthesis, highlighting the prohibitive cost of running a large amount of hardware synthesis. They further extend DSE using a compiler approach and consider parametrization and code transformation techniques usually employed by designers when exploring the broader design space.

On the other hand, Pilato et al. [17] examine the relevance of evolutionary algorithms, especially NSGA-II, exploiting High-Level Synthesis. Moreover, they address the computational cost of a full synthesis or implementation by using an inheritance model. In this way, they lower the number of actual fitness evaluations by inheriting the parents' fitness in a portion of the population. Besides, visual performance models, such as LogCA [18], and Roofline [19], offer a completely different approach that may be a valuable tool for designers together with DSE ones. However, they are still under investigation and offer limited support for RTL-based designs, showing exciting results in static imperative code analysis [20].

For all these reasons, we propose Dovado, an open-source CAD tool with a modular design that enables fast, automatic DSE for a module or a given RTL system with an online approximation model of a target EDA tool.

III. Dovado Framework

This Section presents Dovado, its components, and the three flows to support hardware developers at different levels. Dovado offers design automation functionalities for fast evaluation of a single design point or a set of given ones (i.e., one or multiple configurations) based on Vivado Design Suite (Section III-A). On top of that, Dovado provides an entry-level Design Space Exploration (DSE) feature, where it exploits a multi-objective solver based on a genetic algorithm to find a set of non-dominated design points (Section III-B). Based on this, Dovado can exploit an estimation model to decide whether or not to execute Vivado and pay the implementation time cost (Section III-B).

Figure 1 shows the high-level structure of Dovado, where the reader can notice the different flows of Dovado for design automation or DSE. Once the user feeds Dovado with inputs at the calling step, the tool completely automatize the flows.

```
library ieee;
____
use ieee.std_logic_1164.all;
entity box is
  port (
    clk: in std_logic
    );
end entity box;

architecture box_arch of box is
  attribute DONT_TOUCH : string;
  attribute DONT_TOUCH of BOXED :
    label is "TRUE";
begin
  BOXED:  entity ____
    ____
    port map(
    ____ => clk,
    ____
    );
end architecture box_arch;
```

Listing 1: Box example for VHDL modules, underscores are replaced with module specific parts

A. Single Design Point Evaluation Methodology

Dovado employs a modular methodology applied to RTL architectures, Vivado, and FPGAs, but it is generally valid for hardware development. The single point evaluation is a design automation step consisting of three preprocessing steps and then invoking the synthesis and implementation tool, i.e., Vivado. Dovado starts parsing the given RTL design and extracting the user's parameters along with other useful information. Then, it exploits a sandboxing procedure, which creates a secure environment to avoid unintended simplifications of input/output interfaces. It enables a secure clock constraint application without naming restrictions. Moreover, it enables proper support for safe parametrization applications. Starting from parsing and boxing information, Dovado generates module-specific scripts for the boxed module for a given target technology, i.e., boards or parts.

1) Parsing: The Parsing step is a significant part of the design flow automation. Indeed, we apply a first formal verification to the design and extract essential information for subsequent Dovado steps. We are interested in extracting the hardware module interfaces (i.e., module name, parameters declaration, ports/signal interface declaration) for the boxing step. VHDL and V/SV are context-free languages, hence requiring push-down automata for parsing, while they are regular in the declaration section. However, both languages' different standards present a wide variety of declaration styles of ports and parameters, hindering regular expressions usage and requiring to handle all the different cases in the parsing step. For these reasons, the Dovado parsing step is based on a parser generator called Antlr [21], which exposes different runtimes, easing Dovado deployment across different platforms, and provides VHDL2008 and V/SV grammars. Hence, we employ these grammars for a parsing step robust to declaration styles and reasonably performance on large RTL files.

2) Boxing - Dealing with Pin Overflow and Parsing Shortcomings: The Boxing step is essential to generalize the Dovado approach for every possible RTL module, even one not intended to be a top-level one. Indeed, to avoid tool simplifications at the input/output level, which may cut out part of the module interface, we introduce this sandboxing technique. Thanks to boxing, Dovado can deal with the FPGA implementation phase without incurring in any pin overflow issue or, in general, at the device input/output level. Moreover, the Dovado box is the entry point where we apply parameterization, and it enables a straightforward timing analysis of the module and the final achievable frequency. Listing 1 is an example box frame for VHDL modules, where the underscores represent the run-time information extracted from parsing, e.g., libraries, module name, clock port, and parameters.

3) Synthesis and Implementation: Dovado currently leverages Vivado as the hardware design suite for synthesis and implementation phases. However, another physical tool with a technology library can be used, e.g., Yosys [22], RapidWright [23], Symbiflow [24]. Dovado spawns Vivado as a subprocess and communicates with the physical tool through the TCL interface. As for the Boxing step, we also built general frames for TCL scripts that Dovado customizes at run-time for module specifications and user-selected directives. Indeed, Dovado exposes the possibility of tailoring this step for a given board or parts and customizing the toolchain directives for a given step, i.e., synthesis, place, and route. The user can specify the directives to guide the tool for a given optimization metric, e.g., run-time performance, area usage, and other additional features. For instance, Vivado offers the incremental design flow feature to reuse previous run checkpoints and information to speed up following computations for both synthesis and implementation steps. This feature turns to be particularly useful whenever the module parameters influence only a small subsection of a larger design.

To comply with Vivado's standard compilation for VHDL and V/SV, we apply some naming constraints for VHDL libraries (i.e., one subfolder per library with the same name) and parsing orders specifications (i.e., SV packages are read at the very beginning of the step).

4) Design Points Results: After the physical step, i.e., synthesis and/or implementation, Dovado retrieves metrics of interest as the final result. The main metrics we provide are area utilization and maximum achievable frequency for a given module, including all submodules. The latter comes from the worst negative slack (WNS) extraction and is computed as follows:

$$F_{max} = \frac{1000}{(\frac{1}{1000} * T) - WNS} \tag{1}$$

where T is a user-defined target period (in nano-seconds), usally bigger than the achievable one, *WNS* is the worst negative slack and it is the time interval calculated on the longest path (in nano-seconds). The slack is considered negative when the timing constraint is violated. Differently, the utilization metric divides into the different available resources for a given board/parts, e.g., BRAMs, CLBs, DSPs. However, some resources, such as URAMs, are not always available for

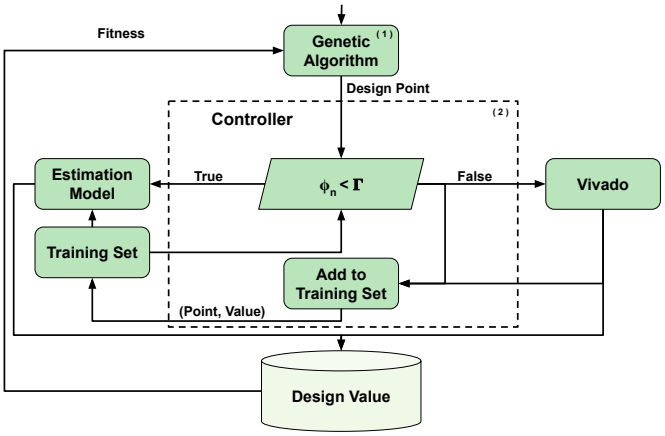

Figure 2: Dovado DSE Block Diagram zoom focused on Synthesis/Implementation Approximation

each possible FPGA, hence are device-dependent and reported only if present.

B. Design Space Exploration Methodology

Dovado builds the entry level of DSE on top of the single design point methodology. Here, given a module with parameters' possible ranges, the DSE module extracts the non-dominated set (or Pareto set) of design points according to some metrics, such as achievable frequency and LUTs usage. We adopt a genetic algorithm to find this set of design points according to our multi-objective optimization formulation. Moreover, we optimize to reduce DSE execution times coming from Vivado executions by exploiting special tool features.

1) Multi-Objective Optimization: We formulate the DSE problem as a multi-objective integer optimization problem since we optimize multiple metrics simultaneously. Considering a single metric is usually trivial, and the optimizer would yield only the degenerative case, i.e., the smallest design possible. Instead, considering multiple optimization metrics might lead to many non-dominated solutions with different area-frequency trade-offs. Moreover, we can consider the DSE as an integer optimization problem since only integer-valued parameters are synthesizable both in VHDL and V/SV. Besides, boolean parameters are treated as integer with 0 and 1 values. Designers may apply further restrictions to the design space; for instance, they can limit the range of a given parameter to only power of two values. In this way, reducing the volume space at exploration time, or even enforcing meaningful solutions only.

We solve this multi-objective optimization problem through Non-Dominated Sorting Genetic Algorithm (NSGA)-II [25], [26]. This procedure is a genetic algorithm that does not require specific domain knowledge on the search space or the optimization metrics, making it suitable for a general DSE solver with adequate performance. Moreover, NSGA-II is an elite-preserving algorithm that preserves non-dominated solutions in the population, while the sorting by non-domination reduces computational complexity [12].

2) Incremental Synthesis and Implementation: To further reduce the exploration time, Dovado exploits the incremental design flow feature of Vivado, which, for each run, writes some archives, called checkpoints. Thanks to these checkpoints, Dovado avoids repeating the exploration of design parts not affected by parametrization. The incremental flow may be independently activated for one, or both, design flow step(s), i.e., synthesis and implementation. For complex and large designs, this technique may be handy to save time for non-parametrized sections.

C. Fitness Function Approximation Model

A critical aspect of genetic algorithm design is the accurate composition of efficient yet representative fitness functions. Essentially, we would need to create a vector of functions where each function returns the value of a given metric for a given design point, e.g., `[utilization(`$point_n$`), frequency(`$point_n$`)]`. This naive approach implies calling Vivado for each exploration iteration and complete the synthesis/implementation, requiring prohibitive execution times for non-trivial modules.

For these reasons, we design a non-parametric statistical model to estimate the actual design points values close enough to already evaluated ones to reduce the time complexity of the exploration. We employ a method inspired by Shokri et al. [27] for reducing calls to an expensive computational simulator, thanks to a control model. Conversely from [27], we use a non-parametric regression model called Nadaraya-Watson Model (NWM) as an approximator. Thanks to the non-parametric method, we simplify the training of the model with a frequently updated dataset. Moreover, we simplify the validation step, according to Shapiai et al. [28] work, where they have shown how the NWM model performs better with a Gaussian Kernel, leaving the bandwidth as the only free parameter. We adopt Leave-One-Out cross-validation given the small size of the dataset and the NWM cheap computational cost. More complex models with higher variance, such as Neural Networks, showed overfitting on such small datasets [27]. In summary, Dovado model is loosely speaking a weighted average of the dataset points, where the weights are defined by a Gaussian Kernel function:

$$\hat{y} = f(x, D, h) = \frac{\sum_{i=1}^{n} K_h(x, x_i) \cdot y_i}{\sum_{i=1}^{n} K_h(x, x_i)} \tag{2}$$

where, D denotes the dataset, $\hat{y}$ is the estimated design value, x is a design point, y_i and x_i are the pair value-point in the dataset, $K_h(x, x_i)$ is the kernel function with bandwidth h, computed as :

$$K_h(x, x_i) = \frac{1}{\sqrt{2\pi}} exp(-\frac{(x - x_i)^2}{2h^2}) \tag{3}$$

Dovado's overall DSE algorithm is shown in Figure 2 without the model preparation step. More specifically, our algorithm starts to generate a synthetic dataset of size M by making M (100 default, can be user-defined) distinct calls to Vivado with randomly sampled design points from user-defined parameter

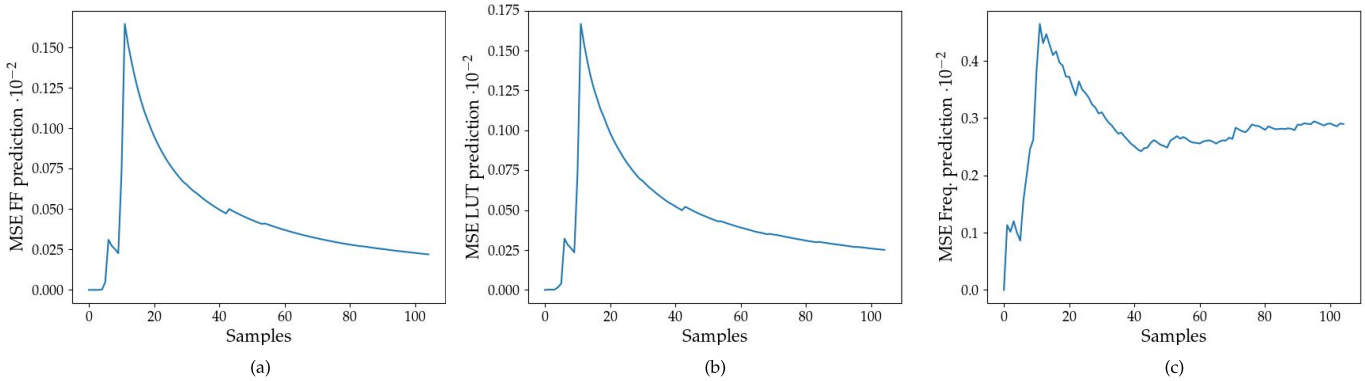

Figure 3: Mean Squared Error for metric estimation on the RISC-V CPU [29] targeting a Kintex-7: (a) Flip Flops predictions, (b) LUTs prediction, (c) Frequency prediction. The more the samples we collect, the more the accurate Dovado become

ranges. Then, we train and validate the statistical model on the synthetic dataset, and the exploration with the genetic algorithm starts. For each new design point explored, Dovado handles three different cases. First, if our design point is already in the dataset, Dovado calls Vivado, which employs cached results as the answer. Second, if the generated design point is similar enough to one of the dataset points, Dovado employs the statistical model for an estimate. Finally, if none of these applies, Dovado calls Vivado, adds the new design pair (tuple-value) to the dataset, and applies a new training/validation step.

As similarity measure to decide whether employing the statistical model or not, we exploit the one proposed by Shokri et al. [27]:

$$\Phi_n = \sqrt{\frac{\sum_{j=1}^{m}(x_j - z_j^n)^2}{m}} \tag{4}$$

where Φ_n is the similarity measure, *j* is the decision variables counter, *m* is the number of decision variables (or design point dimensionality), x_j is the j-th decision variable of the new design point, and z_j^n is the j-th decision variable of the n-th nearest solution from the training dataset. We employ a threshold-based approach to decide whether to call Vivado or the estimator by comparing the threshold with the similarity metric result. However, the threshold setting is a non-trivial problem that depends on run-time information, i.e., the parameters' range. For these reasons, we employ an adaptive threshold set Γ by averaging the distance between dataset points and updating it after an addition to the dataset, $\Gamma = \frac{\sum_{i=1}^{L}\Phi_n^i}{L}$, where *L* is the number of element in the dataset.

IV. Experimental Setup and Results

We employ Xilinx Vivado Design Suite 2019.2 and target different FPGA technologies. The parser exploits ANTLR python run-time [21], while we exploit a NSGA-II genetic algorithm implementation from a novel python library called pymoo [30]. The genetic algorithm hyperparameters are the following: integer random sampling, integer simulated binary crossover [31], with duplication elimination mutation occurs with an approximately Gaussian distribution with 0.5 as mean and variance controlled by a hand-tuned parameter.

We employ four different case studies to showcase Dovado compatibility across languages and projects. The user duty is to specify target board, top module, search space parameters (which one, desired range of exploration) and then Dovado runs automatically. We target for all of them a frequency of 1GHz to better verify the maximum theoretical frequency. The first is an SV-based RISC-V core called cv32e40p [29] where we show the model accuracy convergence. Then we exploit an open-source, high-performance FPGA-based NIC [32], called Corundum, where we explore the design space of a Verilog submodule and we showcase non-dominant solutions. We move then to Neorv32 [33], a VHDL-based RISC-V core, where we apply the power of two restrictions to explore core memories space. Finally, we explore a VHDL-based Architecture called TiReX [34] where we explore both the datapath and memories parameter spaces.

A. Testing Fitness Approximation Model: OpenHW group RISC-V IP

As a case study, we employ a small 32-bit RISC-V core by the open hardware group under the name of cv32e40p [29] to assess the Mean Squared Error (MSE) of the proposed approximation model. Moreover, we see the model variance according to the increasing number of Vivado calls targeting a XC7K70TFBV676-1. We test the DSE on a SystemVerilog FIFO submodule exploring the depth parameter. The interest for this case-study revolves not in finding the optimal trade-off but in exploiting a module that provides enough samples for accuracy assessment. We constrained on time the DSE with a four hour of a soft deadline to the genetic algorithm. The parameter range comprised 500 possible values, and the estimation model was pre-trained on 100 samples.

Figure 3 shows the MSE trends for Flip Flops, LUTs, and Frequency prediction. The reader can notice how the MSE is very low for all the metrics and how the proposed estimation model will converge to a real estimation of a Vivado call given a high enough number of samples in the dataset. For instance, the highest MSE, i.e., the one for frequency estimation, reaches

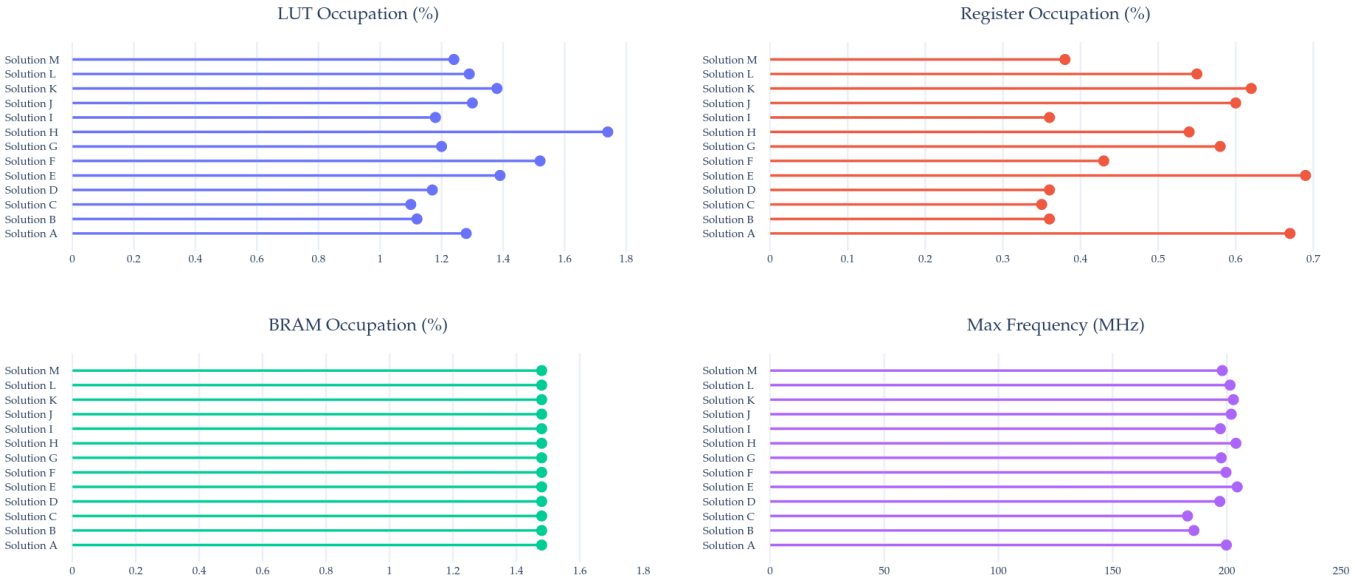

Figure 4: Solution trade-off for Corundum Space Exploration

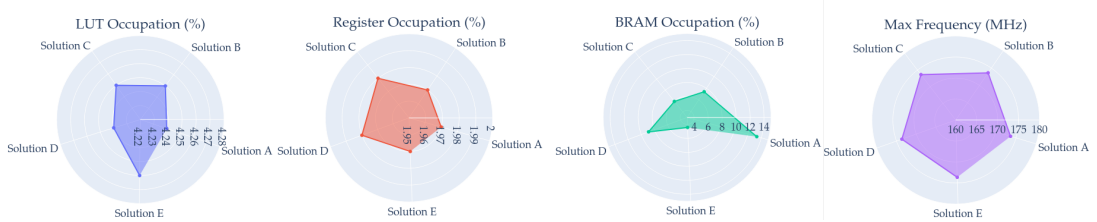

Figure 5: Non-Dominated Solutions for Neorv32 Space Exploration

Table I: Configurations for Corundum Queue Manager

Design Point	A	B	C	D	E	F	G	H	I	J	K	L	M
# operations outstanding	8	8	10	13	27	35	10	12	10	14	19	17	15
# of queues	5	4	4	4	4	4	4	4	7	4	4	4	4
Pipe. stages	2	2	2	3	3	2	3	2	3	3	5	3	4

a peak of $0.45{\cdot}10^{-2}$ and becomes stable around a value of $0.25{\cdot}10^{-2}$ after 40 samples.

B. Corundum: Verilog Network Interface Card

Corundum is an open-source, high-performance FPGA-based NIC [32] written in Verilog. We targeted the same Kintex-7 FPGA disabling the approximator model to employ direct Vivado evaluations. As figure of merits, we employ LUTs, Register, and BRAM occupations with maximum achievable frequency. We explored the parameter space of a non-top module implementing a completion queue manager and targeting design parameters such as the number of outstanding operations, the number of queues, and the pipeline stages. The non-dominated configurations parameters reported from our DSE are in Table I. Figure 4 reports all the quantitative results achieved with Dovado DSE of non-dominated points. Here the reader can appreciate the differences among possible trade-offs offered by different configurations. Indeed, the module is constant in the number of BRAMs needed, while the LUTs and Registers numbers vary according Table I configurations. On the other hand, this module achieves a running frequency near 200 MHz, showing a potential bottleneck in the overall design. We can see how our DSE approach provides a set of non-dominated solutions with useful insights to the hardware developer.

C. Neorv32: VHDL RISC-V core

Neorv32 [33] is an in-order 4-stage RISC-V core VHDL-based. We tested the top module and explore as module parameters the instruction and data memory sizes. We decided to constrain the exploration only to the power of twos to explore a larger parameter space without considering meaningless parameter assignments. We targeted the same Kintex-7 FPGA without the approximation model.

Figure 5 shows the fives non-dominated solutions identified by Dovado. The main difference that the reader can notice is in the high number of BRAMs compared to the others. The first solution exploits instruction and data memories size of 2^{15}, while the others 2^{14} and 2^{13}, respectively. We can see how increasing from 2^{14} to 2^{15} memory sizes yield a sensible

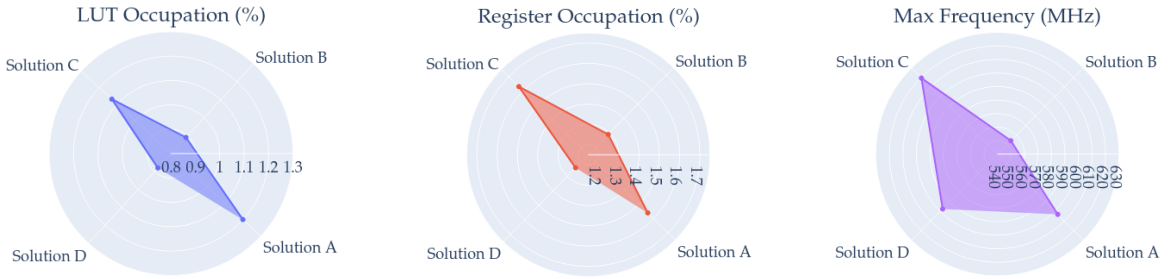

Figure 6: Non-Dominated Solutions for TiReX Space Exploration on ZU3EG

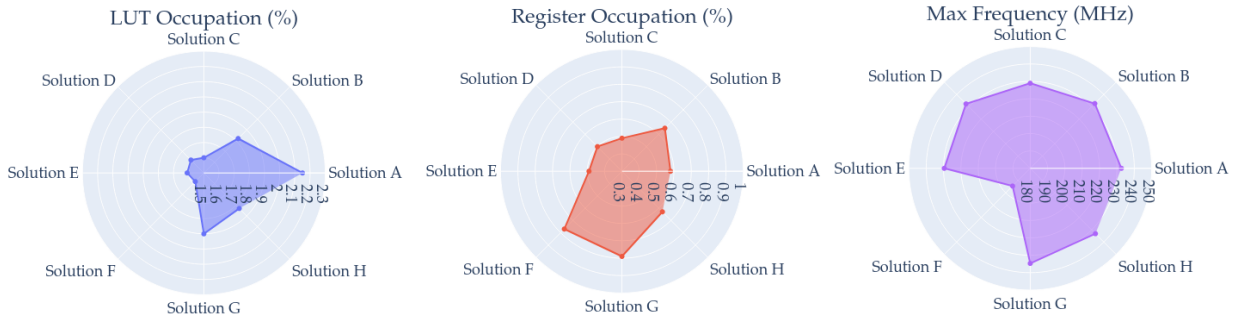

Figure 7: Non-Dominated Solutions for TiReX Space Exploration on XC7K70T

change in BRAM occupation while leaving almost unchanged the other metrics.

D. TiReX: VHDL DSA for Regular Expressions

TiReX [34] is a domain-specific architecture tailored for the Regular Expressions domain. We explore the architecture scaling according to datapath and memories parameters, targeting a Zynq Ultrascale+ XCZU3EG and the same Kintex-7 for the implementation phase. The architecture has two datapath parameters that determine the internal core parallelism and the width of the instruction that we constrain to be a unique parallelism parameter called *NCluster*. Moreover, we explore the size of the instruction and data memories and the stack's size the control unit exploits for context switching.

Figure 6 and Figure 7 shows the DSE results for the ZU3EG and the XC7K70T, respectively. We constrain the exploration with only the power of twos for both devices. We can also appreciate, also from Table II, the differences in the number of identified non-dominated solutions and achieved optimization metrics usage. In this way, we can analyze technology impacts from the different technologies in resource usage and achievable frequencies. For instance, the ZU3EG is produced at 16 nm process while the XC7K70T at 28 nm. Moreover, the ZU3EG has 70K LUTs and 141k Flip Flops, while the XC7K70T has 41k LUT and 82K FF. The reader can see how the achievable frequencies are so different, e.g., 550 against 190 MHz, even though configurations are quite similar.

Table II: Configuration Parameters for TiReX

ZU3EG	A	B	C	D	-	-	-	-
NCluster	1	1	1	1	-	-	-	-
Stack. Size	2^4	2^2	2^8	2^1	-	-	-	-
Instr. Mem. Size	2^3	2^3	2^3	2^3	-	-	-	-
Data Mem. Size	2^4	2^3	2^3	2^3	-	-	-	-
XC7K	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
NCluster	1	1	1	1	1	1	1	1
Stack. Size	2^0	2^1	2^3	2^1	2^0	2^4	2^5	2^3
Instr. Mem. Size	2^3	2^4	2^3	2^3	2^3	2^3	2^4	2^4
Data Mem. Size	2^4	2^3	2^3	2^3	2^3	2^3	2^3	2^3

V. Conclusions and Future Works

With this work, we present Dovado [10], an open-source support for highly parametrized RTL module design automation and exploration for FPGAs. Dovado's modular approach ensures flexibility in target devices, different HDLs (i.e., VHDL and (System)Verilog) for synthesis and implementation tasks. Moreover, Dovado exploits specific features of the given EDA tool, such as different Vivado directives and incremental flows. The experimental results show how VHDL and V/SV are handled seamlessly. Indeed, we explore design spaces of both VHDL (Neorv and TiReX), Verilog (Corundum), and SystemVerilog (cv32e40p) modules. Additionally, the user can tweak resources occupation percentage by targeting different utilization metrics (e.g., LUTs, BRAMs) of a given technol-

ogy, e.g., Xilinx UltraScale+. Dovado provides a model of approximation for synthesis/implementation outcomes based on a synthetic dataset for further exploration at a reduced cost showing promising results.

As future works, we will improve the parser to define design space more broadly in terms of code transformations [16] and parameter optimization for a further boost. We plan to support newer languages, such as Chisel [2] or SpinalHDL [3], that offer a convenient way for highly parametrizable hardware or to support high-level synthesis approaches for fast architecture choices explorations [35]. Currently, Dovado lacks in run-time performance modeling of RTL modules. Hence, we will add the chance of inserting a custom model for static performance that enables an improved DSE and adding a visual performance model (e.g., Roofline [19]).

The computation cost of running a DSE for Dovado can become prohibitive, and studying newer approaches for synthesis/implementation outcome approximations must be carried out. We plan to explore different statistical models, either parametric or non-parametric, to amortize the expensive synthetic dataset generation. Panerati et al. [12] provide an overview of exploration algorithms for multi-objective optimizations with their peculiarities and strengths. Starting from their work, we envision an investigation on a run-time choice among various algorithms based on information from synthetic dataset generation and an in-depth analysis of genetic algorithms operators and encoding strategies trade-offs.

Acknowledgements

The authors are grateful for precious feedbacks from anonymous reviewers and our colleagues M. Salaris, F. Peverelli, E. D'Arnese, E. Del Sozzo.

References

[1] P. A. Simpson, "Rtl design," in *FPGA Design*. Springer, 2015, pp. 91–139.

[2] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avizienis, J. Wawrzynek, and K. Asanovic, "Chisel: Constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*, June 2012, pp. 1212–1221.

[3] C. Papon, "Spinalhdl: An alternative hardware description language," *FOSDEM*, 2017.

[4] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[5] SpinalHDL, "Pinsec: a spinalhdl system on chip generator," https://github.com/SpinalHDL/VexRiscv.

[6] Y. Umuroglu, D. Conficconi, L. Rasnayake, T. B. Preusser, and M. Själander, "Optimizing bit-serial matrix multiplication for reconfigurable computing," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 3, pp. 1–24, 2019.

[7] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 2014, pp. 97–108.

[8] M. A. Kinsy, M. Pellauer, and S. Devadas, "Heracles: a tool for fast rtl-based design space exploration of multicore processors," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, 2013, pp. 125–134.

[9] C. Haubelt, T. Schlichter, J. Keinert, and M. Meredith, "Systemcodesigner: automatic design space exploration and rapid prototyping from behavioral models," in *Proceedings of the 45th annual Design Automation Conference*, 2008, pp. 580–585.

[10] D. Paletti, "Dovado source code," https://github.com/DPaletti/dovado, 2021.

[11] W. H. Wolf, "Hardware-software co-design of embedded systems," *Proceedings of the IEEE*, vol. 82, no. 7, pp. 967–989, 1994.

[12] J. Panerati, D. Sciuto, G. Beltrame *et al.*, "Optimization strategies in design space exploration," *Springer*, 2017.

[13] M. Hammerquist and R. Lysecky, "Design space exploration for application specific fpgas in system-on-a-chip designs," in *2008 IEEE International SOC Conference*. IEEE, 2008, pp. 279–282.

[14] F. Karakaya, "Automated exploration of the asic design space for minimum power-delay-area product at the register transfer level," *Doctor of Philosophy Dissertation*, 2004.

[15] P.-C. Kao, C.-K. Hsieh, C.-F. Su, and A. C.-H. Wu, "An rtl design-space exploration method for high-level applications," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 84, no. 11, pp. 2648–2654, 2001.

[16] B. So, M. W. Hall, and P. C. Diniz, "A compiler approach to fast hardware design space exploration in fpga-based systems," *ACM SIGPLAN Notices*, vol. 37, no. 5, pp. 165–176, 2002.

[17] C. Pilato, A. Tumeo, G. Palermo, F. Ferrandi, P. L. Lanzi, and D. Sciuto, "Improving evolutionary exploration to area-time optimization of fpga designs," *Journal of Systems Architecture*, vol. 54, no. 11, pp. 1046–1057, 2008.

[18] M. S. B. Altaf and D. A. Wood, "Logca: A high-level performance model for hardware accelerators," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 375–388, 2017.

[19] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009.

[20] M. Siracusa, L. Di Tucci, M. Rabozzi, S. Williams, E. D. Sozzo, and M. D. Santambrogio, "A cad-based methodology to optimize hls code via the roofline model," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[21] T. Parr, *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.

[22] C. Wolf, "Yosys open synthesis suite," 2016.

[23] C. Lavin and A. Kaviani, "Rapidwright: Enabling custom crafted implementations for fpgas," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 133–140.

[24] K. E. Murray, M. A. Elgammal, V. Betz, T. Ansell, K. Rothman, and A. Comodi, "Symbiflow and vpr: An open-source design flow for commercial and novel fpgas," *IEEE Micro*, vol. 40, no. 4, pp. 49–57, 2020.

[25] K. Deb and T. Goel, "Controlled elitist non-dominated sorting genetic algorithms for better convergence," in *International conference on evolutionary multi-criterion optimization*. Springer, 2001, pp. 67–81.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[27] A. Shokri, O. B. Haddad, and M. A. Mariño, "Algorithm for increasing the speed of evolutionary optimization and its accuracy in multi-objective problems," *Water resources management*, vol. 27, no. 7, pp. 2231–2249, 2013.

[28] M. I. Shapiai, S. Sudin, Z. Ibrahim, and M. Khalid, "Investigation on different kernel functions for weighted kernel regression in solving small sample problems," in *2011 UKSim 5th European Symposium on Computer Modeling and Simulation*. IEEE, 2011, pp. 64–69.

[29] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[30] J. Blank and K. Deb, "pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[31] K. Deb, R. B. Agrawal *et al.*, "Simulated binary crossover for continuous search space," *Complex systems*, vol. 9, no. 2, pp. 115–148, 1995.

[32] A. Forencich, A. C. Snoeren, G. Porter, and G. Papen, "Corundum: An open-source 100-Gbps NIC," in *28th IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2020.

[33] S. Nolting, "The neorv32 processor," https://github.com/stnolting/neorv32.
[34] A. Comodi, D. Conficconi, A. Scolari, and M. D. Santambrogio, "Tirex: Tiled regular expression matching architecture," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 131–137.
[35] D. Conficconi, E. D'Arnese, E. Del Sozzo, D. Sciuto, and M. D. Santambrogio, "A framework for customizable fpga-based image registration accelerators," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 251–261.