SoftwareX 29 (2025) 101970

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Odatix: An open-source design automation toolbox for FPGA/ASIC implementation

Jonathan Saussereau *, Christophe Jego, Camille Leroux, Jean-Baptiste Begueret

Universite de Bordeaux, Bordeaux INP, Laboratoire IMS, UMR CNRS 5218, France

ARTICLE INFO

Keywords:
Design automation
Design space exploration
Hardware
Computer-aided design
Design flow
FPGA
ASIC

ABSTRACT

In modern hardware digital design, optimizing performance, resource utilization, and power consumption across different technological targets remains a critical challenge. Indeed, the drive for greater computational power, alongside the need to reduce power consumption, stems from a wide range of applications, from data centers to mobile devices. However, this push encounters significant cost barriers, as the manufacturing cost is closely tied to the technological nodes used and the area for integrated circuits, and is particularly influenced by the amount of available resources for FPGAs. These three criteria are inherently conflicting, as improving one often negatively impacts the others. Finding the best balance between these factors requires significant effort. To address these complexities, design automation tools are increasingly valuable. Odatix is an open-source toolbox designed for the automated implementation and validation of parametrizable digital architectures. It supports synthesis, placement and routing for various FPGA and ASIC tools and simulators. It simplifies key stages such as synthesis, place and route, simulation, and validation, allowing designers to efficiently navigate multiple configurations and identify optimal solutions tailored to specific application constraints. Indeed, Odatix enables comparative analysis of multiple architectural configurations through various metrics such as maximum operating frequency, resource utilization, and power consumption. This paper presents an overview of Odatix's capabilities and its application to the AsteRISC processor, demonstrating its utility in choosing the best architectural configuration, technological target and EDA tool for specific application constraints.

Code metadata

Current code version	v3.1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00485
Permanent link to Reproducible Capsule	
Legal Code License	GNU General Public License (GPL) v3
Code versioning system used	git
Software code languages, tools, and services used	python, tcl
Compilation requirements, operating environments & dependencies	Requires Linux, python 3.6+, make, at least one of the supported EDA tools
If available Link to developer documentation/manual	https://odatix.readthedocs.io/en/latest/
Support email for questions	jonathan.saussereau@ims-bordeaux.fr

1. Motivation and significance

When implementing a digital functionality, designers have several options open to them. The main ones are designing a software program to be executed by a processor, or creating a dedicated architecture that can be implemented on a reconfigurable device like an FPGA [1], or in integrated circuit technology to produce a chip.

In various contexts, the design of digital architectures for FPGA circuits and integrated circuit technologies can offer numerous advantages over a software approach [2]. However, it involves various challenges and requires carrying out a series of systematic steps. Fortunately, these steps can be automated. In both cases, the process typically begins with an RTL (Register Transfer Level) coding in a HDL (Hardware

* Corresponding author.
E-mail address: jonathan.saussereau@ims-bordeaux.fr (Jonathan Saussereau).

https://doi.org/10.1016/j.softx.2024.101970
Received 10 September 2024; Received in revised form 24 October 2024; Accepted 4 November 2024
Available online 5 December 2024

Description Language). This code is then synthesized to create a netlist, which maps the design to the FPGA's configurable logic blocks or the ASIC [3] standard cells. Next, place-and-route tools determine the physical layout, optimizing for speed, area, and power. Finally, on FPGA, the bitstream is generated and loaded onto the circuit for testing and deployment. The ASIC design [4] flow includes additional steps such as floorplanning before placement and routing, and design rule checking (DRC). Once the design passes these checks, it can be taped out and sent to a foundry for fabrication.

Among digital architectures, parameterizable designs are of significant interest. In the current landscape, stringent application constraints on performance, power consumption, and resource utilization require flexible architectures. In recent decades, evolutions of traditional HDLs like SystemVerilog [5], and new approaches such as Chisel [6], have simplified the design of parameterizable architectures. These architectures can be adapted to specific application requirements, achieving the best trade-off for a given context. This is particularly, but not exclusively, pertinent for processor architectures.

Indeed, with the emergence of open-source instruction sets, such as RISC-V [7], there has been a notable increase in the availability of open-source processor architectures, and flexible, configurable processors have started to emerge. For instance, the Rocket Chip generator [8] demonstrates a flexible core design using Chisel. BRISC-V [9] provides an architecture design space exploration toolbox that supports the generation of various processors, from single-cycle to pipeline processors, and multi-core systems. HL5 [10] is a processor designed for HLS. AsteRISC [11] is a multi-cycle processor with numerous configurations for design space exploration. AsteRISC also exists in a flexible pipelined version [12], enabling even more architectural configurations. In addition, cores that are not intended to be fully flexible still offer some configurability, like the PicoRV32 [13] for example.

The challenge lies in efficiently exploring architectural configurations in order to find the most appropriate one. Many tools exist in the state-of-the-art for the design automation of digital designs. However, their use can become complicated with parameterizable architectures. In such cases, design space exploration tools can be helpful by reducing the design space for many parameters. To reduce the number of parameters, tools like OpenTuner [14] can be useful. In the hardware domain, tools such as Heracles [15] and Aladdin [16] have emerged to optimize the performance of parameterizable architectures by employing design space exploration techniques. However, there appears to be a gap in the state-of-the-art for design automation tools specifically aimed at parameterizable architectures. Dovado [17] addresses this issue by providing an automation tool for FPGA design through Vivado, allying design automation and design space exploration. However, in cases where the number of possible configurations is not overwhelmingly large, it can be beneficial to implement all configurations and compare them across all available metrics, providing a comprehensive view of the design space. Another complementary use case is the comparison of different architectures, whether they are parameterizable or not. OpenLane [18], an open-source ASIC flow, offers a feature to run different configurations in parallel. However, this functionality is specific to OpenLane and cannot be used with other EDA tools. Additionally, existing tools also lack support for simple simulation and validation of architecture configurations, which is one of the primary challenges for configurable architectures. Moreover, it would be valuable to have a tool that works equally well for both ASIC and FPGA, allowing for simple and clear comparisons of architectural configurations. It would also be valuable to have an interactive interface for analyzing the results.

Odatix was developed to address these challenges by providing an automation toolbox to simplify the synthesis, implementation, simulation, and validation of configurable digital architectures. It makes it easier for designers to identify optimal solutions for their specific requirements. Odatix aims to provide the scientific community with a tool that enables rapid prototyping and evaluation of different architectural configurations. Users of Odatix input architectural details via a parameter file. They provide the tool with classic settings such as RTL paths, top-level modules, and clock signals. Each configuration is defined by a combination of parameters in with the syntax of the top level language, allowing seamless integration of any hardware description language such as Verilog, VHDL, SystemVerilog, and even higher level HDLs like Chisel. The software automates the generation of temporary directories, RTL synthesis and optional place and route, and simulation on selected technological targets. Thus, it optimizes the workflow for researchers and engineers.

2. Software description

Odatix is a toolbox designed to simplify synthesis, place and route, simulation, and validation of configurable digital architectures. It can be used side by side with various FPGA and ASIC EDA tools, listed in Table 1. It is worth noting that support for additional tools can be easily extended. The following subsections outline the prerequisites required for this process.

Table 1
Supported EDA tools for synthesis + place&route.

EDA tool	Flow	License type
AMD Vivado	Synthesis + Place&Route	Commercial
Synopsys Design Compiler	Synthesis	Commercial
OpenLane [18]	Synthesis + Place&Route	FOSS

Odatix allows, for any digital architecture:

- running parallel synthesis for specified configurations of an architecture
- finding the maximum operating frequency of each configuration
- running parallel simulations for each configuration
- exporting synthesis and/or place and route results in a common format
- exploring results in an interactive web interface

The primary feature of this toolbox lies in its capability to compare different architectural configurations using parameter files. With Odatix, users can effortlessly explore different architectural configurations and evaluate their performance based on numerous metrics including the maximum operating frequency (f_{max}), hardware resource utilization and power consumption.

Odatix also enables parallel simulations of different configurations of the same design. This is useful both for validation and for comparing configurations, as with benchmarks.

It is worth noting that parallel syntheses or simulations do not necessarily have to be from the same architecture. This feature is particularly useful for comparing different architectures.

Fig. 1 visually illustrates the capabilities of Odatix. On the left, various architecture configurations are used as inputs for different synthesis or simulation EDA tools. Each synthesis tool can have multiple targets, including both ASIC technologies and FPGA circuits. All results are automatically formatted into a common structure, which serves as input for a result exploration tool. There is a vast number of possible combinations between architectural configurations, EDA tools, and technological targets. Typically, a designer is familiar with just one EDA tool and manually implements a limited set of architectures on a few technological targets, formatting the results manually, which is a labor-intensive process. The combination representing an implementation of an architectural configuration on a specific technological target, using a specific EDA tool is highlighted in red. It emphasizes the immense effort required to perform the whole work manually, while the number of architectures and technological target illustrated here remains reasonable. Odatix automates this entire process. It enables users to find the best combination of architectural configuration, EDA tool, and

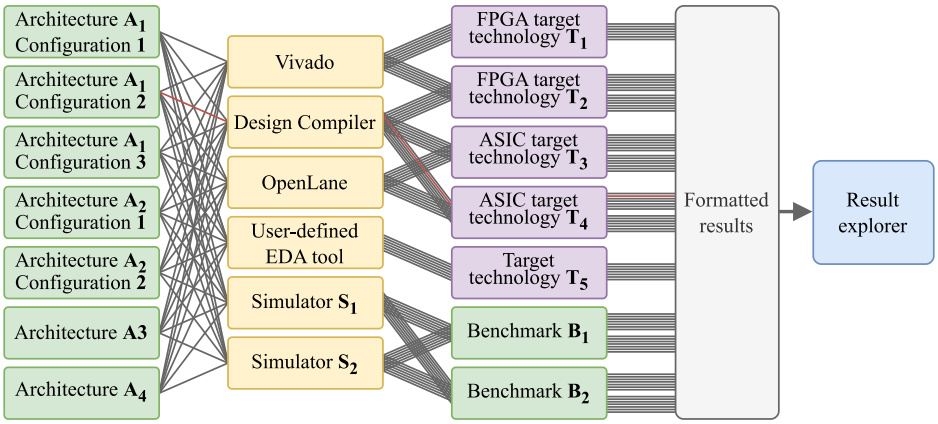


Fig. 1. Odatix functionality diagram.

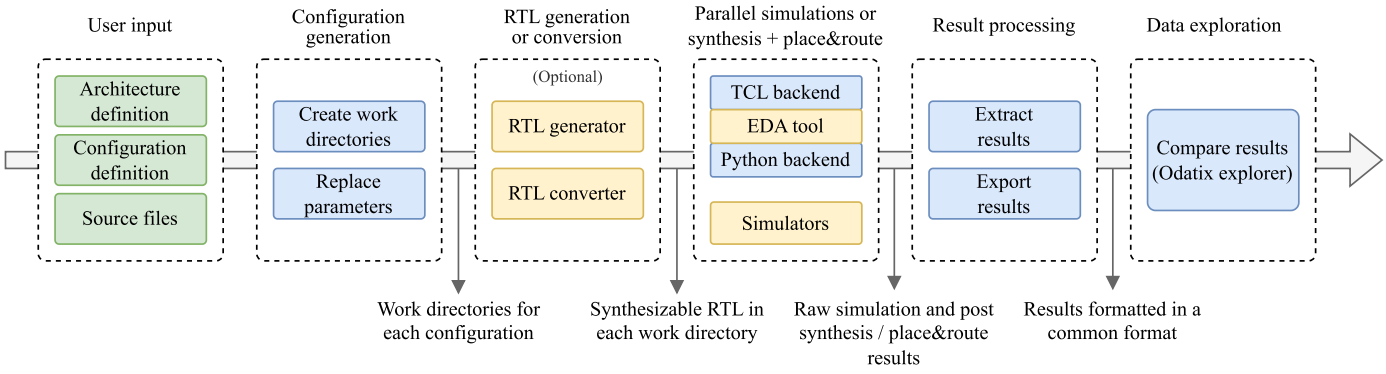


Fig. 2. Schematic of Odatix's workflow.

technological target for a given application. Additionally, through its data processing capabilities, Odatix simplifies the connection between validation, benchmarking, and implementation. In comparison, using Dovado, while offering unique capabilities in terms of design exploration, only provides results for Vivado on FPGA and does not offer the same result comparison features. Similarly, directly using OpenLane's architectural exploration capabilities does not yield implementation results with other tools or FPGA targets, and there is no exploration tool to easily compare the obtained results. Both tools lack from an interaction with simulation.

2.1. Software architecture

Odatix's architecture is shown in Fig. 2. The various components are described below.

In this figure, green represents the elements provided by the user, such as source files and configuration. Blue represents the tools that make up Odatix. Yellow corresponds to external tools, including synthesis/place-and-route tools, simulators, RTL generators, converters, and others.

2.1.1. User configuration

In order to use Odatix, the first step involves specifying the details of their hardware architecture in a configuration file. This includes the typical parameters of a digital architecture: top-level entity, clock signals, and reset signals.

In this file, the user will also define the delimiters that correspond to the sections of the code where the parameters defining an architectural configuration are located.

To implement a configuration, the tool replaces the content within the tags in the top-level entity with the content from the parameter file corresponding to that configuration.

The user can optionally specify frequency search bounds for each target technology, and even for each configuration within the same technology.

Listing 1 provides an example of a digital design described in SystemVerilog [5]. It is a configurable counter, where a parameter allows the user to select the number of bits for the counter. Listing 2 shows an example of a configuration file for this architecture, containing the elements described earlier.

Listings 3 and 4 illustrate two examples of configuration definitions for this architecture. These examples represent two different combinations of parameters. In this case, there is only one parameter, but for a design with more parameters, all additional parameters would also be defined here.

2.1.2. RTL generation

The principle behind the optional RTL generation in Odatix is to allow for the execution of a command before the parameter replacement stage. This command typically generates the RTL that will subsequently be synthesized or simulated. The tool itself is agnostic to what happens during this command execution. It only requires that the output is an RTL description. It also must match the characteristics defined in the settings files. This flexibility means the generation command can involve High-Level Synthesis (HLS), generation via Chisel [6], or even language conversion. Language conversion is particularly useful when the design language is not supported by the chosen tool. For example, OpenLane does not natively support VHDL, but this option enables the conversion of VHDL to Verilog using tools like GHDL [19].

2.1.3. Logic synthesis

In digital design, EDA tools for logic synthesis follow a multi-step process to convert RTL hardware descriptions into gate-level representations. These steps typically include source code analysis, generic synthesis, technology-specific synthesis, and various optimization stages.

```
module counter #(
  parameter BITS = 8
)(
  input  wire            clock,
  input  wire            reset,
  input  wire            i_init,
  input  wire            i_inc_dec,
  output wire [BITS-1:0] o_value
);
  logic [BITS-1:0] value;

  always_ff @(posedge clock) begin
    if (reset) begin
      value <= 0;
    end else begin
      if (i_init) begin
        value <= 0;
      end else begin
        if (i_inc_dec) begin
          value <= value + 1;
        end else begin
          value <= value - 1;
        end
      end
    end
  end

  assign o_value = value;

endmodule
```

Listing 1 Example of digital design

```
##############################################
# Settings for counter example
##############################################

rtl_path: "examples/counter_sv"

# design settings
top_level_file: "counter.sv"
top_level_module: "counter"
clock_signal: "clock"
reset_signal: "reset"

# copy a file into synthesis directory?
file_copy_enable: No

# delimiters for parameter files
use_parameters: Yes
start_delimiter: "counter #("
stop_delimiter: ")("

# optional target-specific bounds (in MHz)
xc7a100t-csg324-1:
  fmax_lower_bound: 250
  fmax_upper_bound: 900
xc7k70t-fbg676-2:
  fmax_lower_bound: 200
  fmax_upper_bound: 1800
XFAB180CMOS:
  fmax_lower_bound: 400
  fmax_upper_bound: 700
```

Listing 2 Example of architecture settings

```
 parameter BITS = 16
```

Listing 3 Example of a parameter file

```
 parameter BITS = 32
```

Listing 4 Another example of parameter file

Each step can be triggered through specific commands, allowing the synthesis process to be fully automated using scripts. Whether targeting FPGA or ASIC, the fundamental concepts are quite similar. In fact, generic synthesis tools, such as Yosys [20], can be used for both FPGA and ASIC designs.

To perform technology-specific synthesis (for ASIC or FPGA), the EDA tool needs access to technology data. For ASIC synthesis, the tool requires a standard cell library, which it uses to map the logic into cells from the library. For FPGA synthesis, the logic is mapped primarily onto lookup tables (LUTs), and potentially other components like DSP (Digital Signal Processing) blocks. With target-specific tools such as Vivado, these technology-specific data are integrated into the tool itself and automatically loaded when the target is specified. In contrast, for more general-purpose tools like Design Compiler or OpenLane, specific commands are required to link the technology files to the synthesis process.

Odatix is designed to work with both types of EDA tools. On one hand, the technological target is made available by the synthesis script, allowing the EDA tool to link the technology data itself. On the other hand, Odatix can make the EDA tool execute target-specific scripts that load the appropriate technology data based on the selected target. Thanks to this flexibility, a wide range of synthesis tools can be supported.

2.1.4. Maximum operating frequency synthesis

The critical path in a digital circuit refers to the longest sequence of dependent operations between two flip-flops or registers. This path determines the maximum delay through the circuit, as data must travel along it before the next clock cycle can begin. The length of the critical path directly impacts the maximum clock frequency at which the circuit can operate. The shorter the critical path, the higher the possible clock frequency.

To determine the maximum clock frequency of a design, the typical method involves synthesizing the design and using Static Timing Analysis (STA) tools to identify the critical path. This path gives the longest delay between registers, which dictates the highest possible clock frequency for that specific synthesized implementation. However, this approach provides the maximum frequency for a particular implementation of an architecture on a specific technology target, rather than an intrinsic value for the architecture itself. The results are influenced by the synthesis tool's decisions. Generally, synthesis tools require a target frequency input and adjust their algorithms accordingly, choosing different cells or applying optimizations throughout the flow to meet the desired frequency. As a result, it is common to synthesize an architecture with a higher maximum frequency than would be predicted from the critical path of the same architecture synthesized with a lower target frequency. This is because the tools can apply different strategies depending on the requested performance goal.

A solution to determine the true maximum frequency that can be achieved by synthesizing the design is to perform a binary search, running successive syntheses with a target frequency that converges towards the actual maximum operating frequency. The condition for this binary search is that the timing constraints must be met, meaning the critical path must have a delay shorter than the clock period. By iteratively adjusting the target frequency based on whether the timing constraints are satisfied, this method allows for progressively narrowing in on the highest frequency at which the design can reliably function.

Two backends have been defined for f_{max} synthesis in Odatix: a Python backend and a Tcl backend. The Tcl backend is designed to execute directly within the synthesis tool. It allows the tool to be launched only once and to analyze the source files a single time, which significantly reduces processing time. This backend is particularly advantageous for tools that support Tcl scripting natively.

The Python backend, on the other hand, can be used for tools that do not natively support Tcl or where the use of the Tcl backend would

be more complex. Currently, none of the supported tools use the python backend, but its role is to enable support for additional tools.

Both backends implement the same algorithm to evaluate the maximum operating frequency. This algorithm involves the binary search described earlier, between user-defined frequency bounds. The process starts by synthesizing at a frequency halfway between the lower and upper bounds. If the timing constraints are met, the synthesis continues at a frequency halfway between this value and the upper bound; otherwise, the synthesis continues at a frequency halfway between the lower bound and the current value.

These backends require specific functions for each tool: one function to set the synthesis frequency and another to determine if the timing constraints are met. Integrating multiple EDA tools presents compatibility challenges. The ability to easily add and edit new implementations of these functions enables simple support for new tools. It also helps to resolve potential compatibility issues caused by different tool versions.

Once the maximum frequency is found, the synthesis results and reports at this frequency are saved.

2.1.5. Simulation

Odatix provides a high degree of flexibility for simulation, allowing designers to make use of any simulation tool they prefer. Unlike the synthesis process, there is no requirement to use one of the supported tools. Instead, users define a simulation with a 'sim' rule in a Makefile. Odatix then executes this rule in a work directory for each configuration of each architecture specified by the user. This means that Odatix is agnostic to the specifics of what happens within the Makefile, enabling the use of any simulation tool installed on the user's machine.

2.1.6. Parallel job scheduling

For both simulation and synthesis, Odatix enables multiple jobs to run in parallel. Since these processes are typically single-threaded, this feature optimizes the use of multi-core processors, thereby reducing the overall computation time. As a result, on an n-core processor, running n simulations or syntheses simultaneously does not take significantly more time than running just one.

To avoid overloading the processor, the user can set a limit on the number of parallel jobs. Jobs that cannot be started immediately are queued and gradually launched as slots become available when other jobs finish. This feature is particularly useful when a large number of jobs are running simultaneously. The user simply needs to wait, as the scheduling is handled automatically. The slot allocation logic ensures that the maximum computing capacity of the machines is consistently utilized.

Jobs are launched using Python's `subprocess` module. A curses-based interface in the terminal enables the monitoring of job progress and viewing live job logs. The arrow keys can be used to select which process's logs are displayed. It helps the user monitor each job in real-time and quickly identify any potential errors, simplifying the troubleshooting process.

It is important to note that running jobs with high complexity, such as f_{max} synthesis for very large systems, is time-consuming. Therefore, execution time becomes a limitation when trying to execute many long-running jobs in parallel on a limited number of cores. On the over hand, this feature benefits from the availability of a large number of cores, reducing the number of queued jobs.

The status of each job is displayed continuously, allowing the user to see whether a job is running, has succeeded, failed, or is waiting.

A visual representation of the interface is provided in Fig. 3.

2.1.7. Result processing

Each job has a dedicated work directory. Results are stored there, making it easy to retrieve them later. The directory structure distinguishes between job types (simulation or f_{max} synthesis), and for f_{max} synthesis, there is also a hierarchy based on the tool used and the target technology and architecture. The results obtained from both synthesis and simulation are extracted from these directories and formatted into a common format. For each synthesis tool, available metrics are defined in a configuration file with the method to retrieve them. The tool

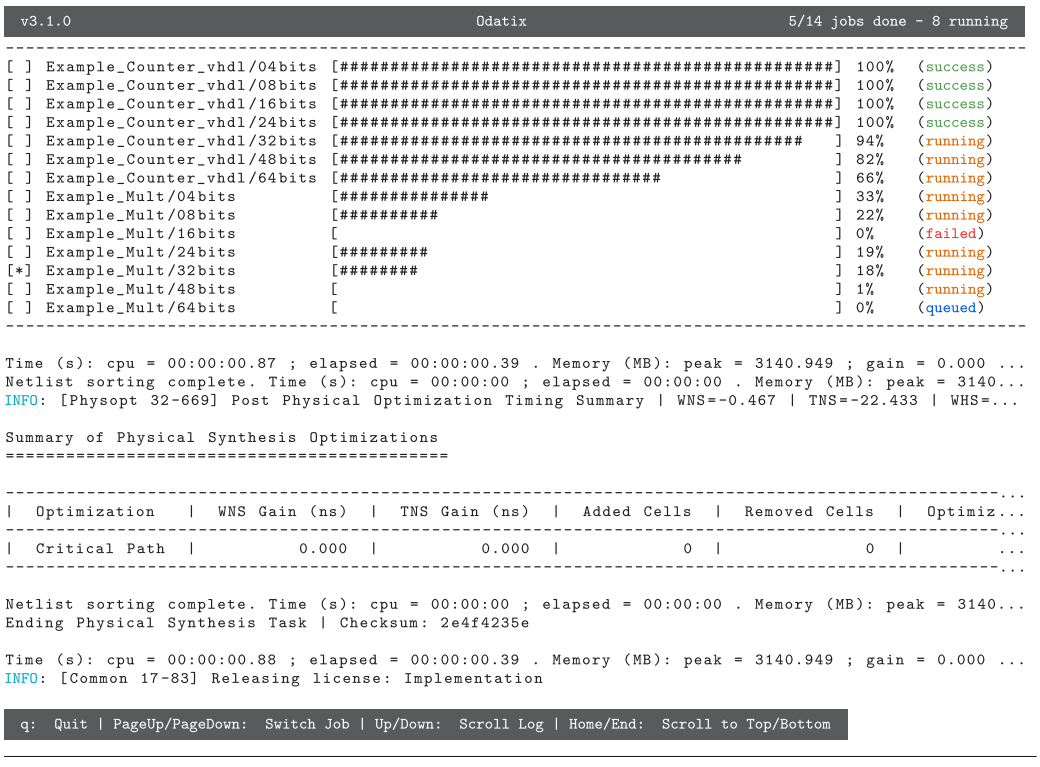


Fig. 3. Odatix interface while running parallel f_{max} synthesis.

```yaml
Fmax:
  type: regex
  settings:
    file: log/frequency_search.log
    pattern: "(.*)Max frequency: ([0-9_]+) MHz"
    group_id: 2
  format: "%d"
  unit: MHz

Cell_Area:
  type: csv
  settings:
    file: runs/odatix/reports/metrics.csv
    key: CoreArea_um^2
  format: "%.3f"
  unit: um^2

DMIPS_per_MHz:
  type: benchmark
  settings:
    key: DMIPS_per_MHz
  format: "%.3f"

Static_Power:
  type: regex
  settings:
    file: report/power.rep
    pattern: "\\| Device Static.*?([0-9.]+).*"
    group_id: 1
  format: "%.3f"
  unit: W

Dynamic_Power:
  type: regex
  settings:
    file: report/power.rep
    pattern: "\\| Dynamic.*?([0-9.]+).*"
    group_id: 1
  format: "%.3f"
  unit: W

Total_Power:
  type: operation
  settings:
    op: Static_Power + Dynamic_Power
  format: "%.3f"
  unit: W
```

Listing 5 Example of metrics definition

supports regular expressions, reading keys from csv and yaml files and even operations between metrics.

This flexible definition method and variety of possible formats allows the support of a wide range of EDA tools. Additionally, there are no constraints on which metrics must be implemented. This helps avoiding potential compatibility issues between tools. Moreover, since users can request Odatix to use their own definition files, they can customize output metrics to suit their specific needs.

Similarly, for simulation, parsers are defined to extract benchmark results for each configuration, such as Dhrystone [21] for processors. Thus, Odatix can be useful to compare the performance of architectures through benchmarking via simulation. Additionally, the tool is capable of merging these data with the results of the f_{max} synthesis. In the case of Dhrystone, the simulation provides a score in terms of DMIPS/MHz. By retrieving the f_{max}, we can thus obtain a score in terms of DMIPS. It allows for a more comprehensive evaluation of architectural performance across different configurations. This integration of simulation and synthesis results is helpful to make more informed decisions about design choices and optimizations.

Listing 5 shows a variety of possibilities for defining the metrics to be extracted. Indeed, one can see the definition of a metric extracted from a file through a regular expression, another from a csv, and the last one from a benchmark result file produced by Odatix. It also illustrates the operation capability, the total power is computed from the sum of two metrics extracted from a report file.

2.1.8. Result explorer interface

Finally, an interactive web interface is proposed to explore the results. This makes it possible to compare the configurations and architectures used across a wide range of metrics for each technological target. This tool can be configured to run locally only or to be accessible over the network, which is useful when working on a headless server. It relies on the python package `plotly` to render the charts. The proposed interface features multiple display modes.

The XY mode displays each configuration on the horizontal axis and the selected metric on the vertical axis. The VS mode allows for plotting one metric against another. The radar mode displays all metrics side by side on a radar chart, providing an overview.

Several display options are available, and the figures can be exported either as vector graphics in SVG format or as images in JPEG, PNG, or WEBP formats.

This visualization tool offers a significant advantage over existing tools by allowing users to easily compare architectural configurations and architectures. This spares the user from having to manually extract and format the data. Odatix provides a unified display, regardless of the EDA tool used. Thus, it also helps in identifying the most suitable target technology or EDA tool for a given application context.

2.2. User experience

Odatix was designed with usability in mind. Even for those with limited experience in Electronic Design Automation (EDA), the complexities of the supported tools are abstracted, hiding complicated tasks. Odatix requires the same inputs from users, regardless of the EDA tool used, ensuring that using an unfamiliar tool is no more difficult than using one they are accustomed to. The mandatory inputs are fairly standard and do not require advanced knowledge: top-level name, clock signal, and reset signal. These inputs are specified in YAML files, whose syntax is designed to be human-readable and easy to use. Integrated examples and documentation make it simple to get started with the tool.

Odatix also caters to advanced users, allowing the integration of custom scripts. However, since Odatix relies on third-party EDA tools, users need to be able to install them. The Odatix documentation therefore provides guidance for installing the supported tools.

Odatix offers an interactive web interface for formatting and displaying results, with the goal of minimizing the user's effort in obtaining valuable insights on the implemented designs. This interface simplifies the process of reviewing and analyzing data. This provides users with clear and actionable information without the need for manual extraction or formatting. It enhances the overall user experience by streamlining the workflow.

3. Illustrative examples

The environment of the processor AsteRISC [11] uses Odatix to automate both simulation and technological implementation [22]. This processor is characterized by its flexibility, offering a wide range of architectural configurations. It is an RTL-level design of an RV32I RISC-V core, implemented in SystemVerilog. AsteRISC's unique feature is the inclusion of optional registers at critical points within the datapath, enabling direct management of the critical path. By selecting the desired register configuration through parameters before logic synthesis, the microarchitecture can be easily adjusted without the need for manual RTL modifications. The control unit of AsteRISC is implemented as a state machine that adapts to the selected configuration, facilitating the exploration of different configurations to achieve the optimal balance among application constraints such as cycles per instruction, maximum operating frequency, resource utilization, and power consumption. The selected System-on-Chip (SoC) configuration for implementation tests

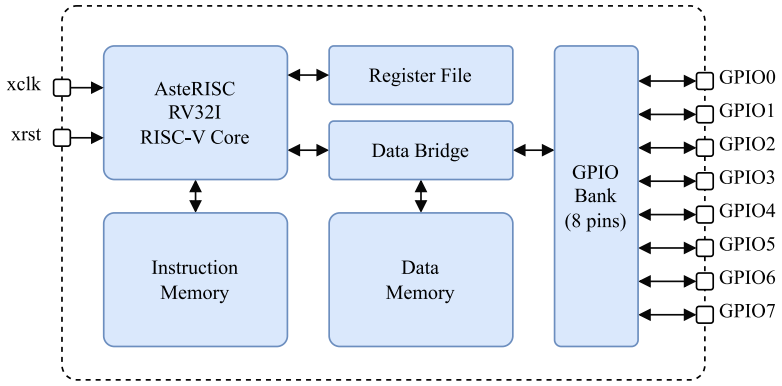


Fig. 4. Schematic of the minimalist AsteRISC SoC.

is depicted in Fig. 4. This SoC design incorporates AsteRISC, interfaced with its register file. The processor is directly connected to an instruction memory. On the data bus, a bridge facilitates the connection between the data memory and an eight-input GPIO module with the processor.

Several variants of the minimalist AsteRISC SoC were defined to compare configurations in different contexts, including: memory sizes, presence or absence of extensions, multiplication algorithms used.

The parallel simulation features of Odatix are utilized to validate each processor configuration using RISC-V tests. Subsequently, this simulation capability was employed to obtain processor performance results via benchmarks. Odatix then extracts these benchmark results and condenses them into a YAML file. The benchmarks provide a score in MIPS/MHz.

The f_{max} synthesis feature was also used to evaluate the maximum operating frequency of each configuration. The experimental setting is the CPU encased in a minimalist SoC including instruction and data memories, with a Total Memory Size of 2048 × 32, and an 8 I/O GPIO peripheral. Odatix automatically calculates the performance score in MIPS at the maximum operating frequency. Other metrics such as

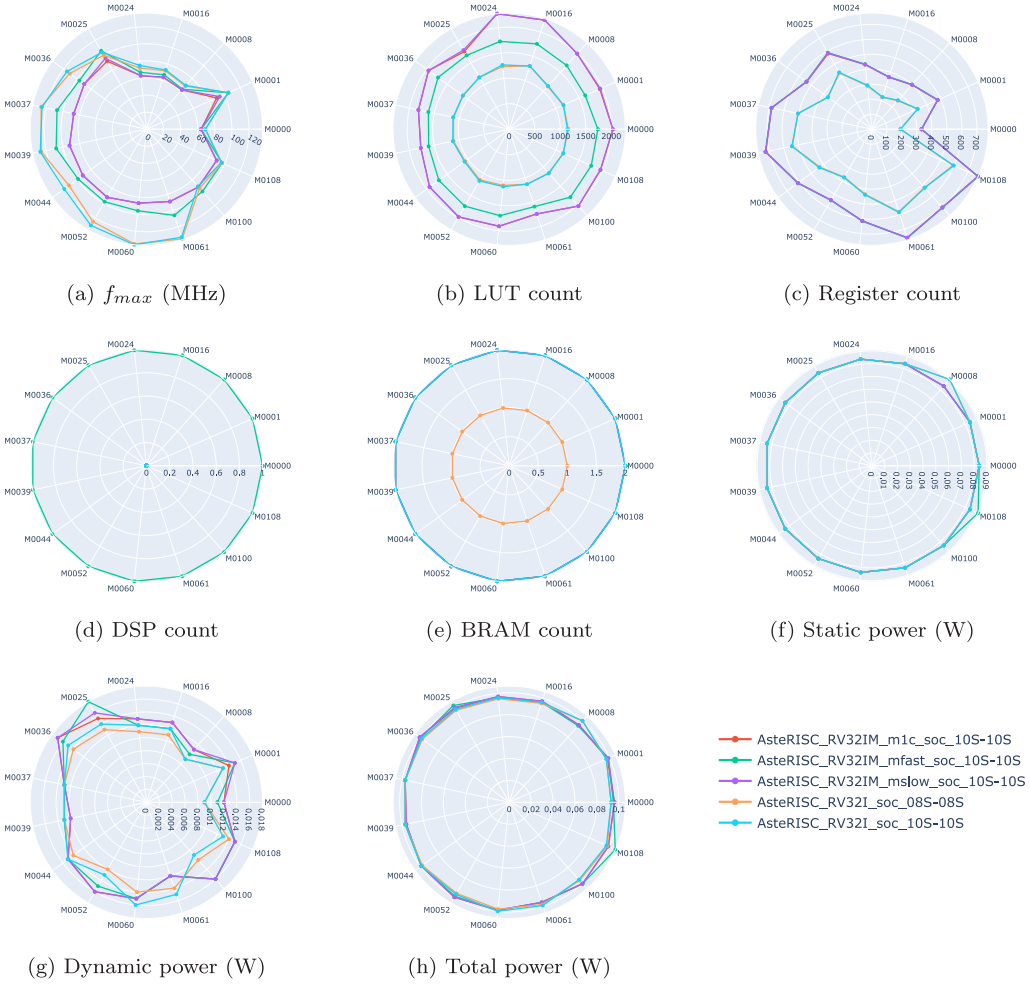
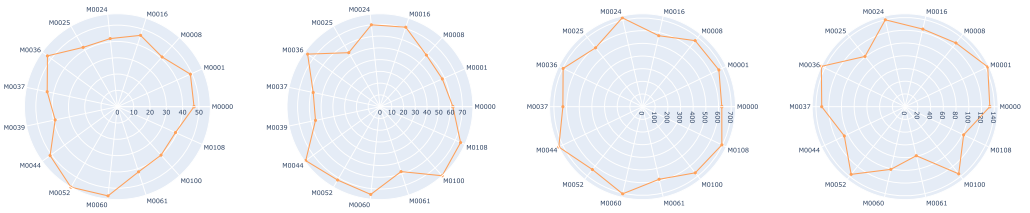


Fig. 5. FPGA performance results of the minimalist AsteRISC SoC.



(a) AMD xc7a100t (FPGA) (b) AMD xc7k70t (FPGA) (c) ST 28 nm CMOS (ASIC) (d) ST 130 nm BiCMOS (ASIC)

Fig. 6. DMIPS performance score of AsteRISC on various technological targets.

resource usage, and power consumption are similarly extracted and exported by the proposed tool.

Finally, the results visualization tool allows for the comparison of each configuration. Fig. 5 shows most of the valuable metrics from the FPGA implementation on AsteRISC, up to the place and route stage on an AMD xc7a100t-csg324-1 FPGA.

Fig. 6 illustrates the DMIPS performance score of AsteRISC across different target technologies: two FPGA circuits and two different ASIC technologies. These results were obtained using the Dhrystone benchmark and the maximum operating frequency. The differing shapes of the curves highlight how certain configurations are better suited to specific target technologies than others. Odatix allows users to visualize this effect, guiding them in making informed choices.

It is worth noting that all the curves presented in this section were generated using Odatix.

4. Impact

Odatix provides designers and users of configurable architectures with the capability to simulate, validate, benchmark, and implement their designs on both FPGA and ASIC technological targets. By using Odatix, one can work towards identifying the most suitable architectural configuration for a given application context. Moreover, it is important to note that this tool also allows for comparisons between target technologies, as well as the tools and design flows associated with them. Additionally, unlike existing tools, Odatix's simulation capabilities enable, for example, the comparison of specific program performance on the target architecture, particularly in the case of programmable architectures like processors. This helps identify the optimal combination of EDA tools, target technology, microarchitecture, and program for a given application.

Without using Odatix, obtaining the results from the previous section would have been particularly time-consuming and tedious. For each tested microarchitecture, one would need to manually select the correct parameters, initiate synthesis at a given frequency, and after several minutes of synthesis, check if the timing constraints were met. Another synthesis would then need to be launched at another frequency, determined by the previous results. The process should have been repeated until the maximum frequency was achieved. Then, one would manually extract the results from various report files for each microarchitecture. Finally, these results would need to be manually formatted into graphs. With Odatix, all these steps are automated, eliminating the need to worry about the number of implementations performed, as it requires no manual work, only computational time.

A major feature of Odatix, compared to the state of the art, is the unified formatting of results in a format that allows the display of results in an interactive web interface. This functionality greatly simplifies interpretation and enables users, even those with little experience in ASIC or FPGA implementation, to understand the trade-offs between architectures. It also enables to make the most appropriate implementation choice for a given application context.

It should be noted that Odatix was designed to be adaptable to any EDA tool and capable of handling any metric. This means it can be easily adapted to new tools as they emerge. Indeed, with the recent rise of open-source alternatives to existing commercial tools [23], Odatix's flexible design ensures its relevance in the face of new EDA tools and potential new research questions that could emerge in the future.

Odatix has been used in collaboration with Thales, an industrial partner, with the aim of identifying the most appropriate version of AsteRISC to replace the fixed architecture of an ASIC.

Another element is that Odatix enables easily reproducible results, even when dealing with a large number of architectural configurations. This is a valuable asset for the scientific community.

5. Conclusions

Odatix provides a powerful solution for the implementation and optimization of digital architectures. Its capabilities in automating simulation and implementation, evaluating multiple configurations, and visualizing performance metrics make it a valuable tool for researchers and engineers. Its flexible, open-source nature, combined with the ability to integrate multiple EDA tools and target both FPGA and ASIC implementations, empowers designers to optimize their architectures across various metrics, including performance, resource usage, and power consumption. The application of Odatix to the AsteRISC processor has demonstrated its effectiveness in tailoring processor designs to specific application constraints. Indeed, the tool's automated workflows drastically reduce the time and effort needed for design exploration, while its visualization features offer an intuitive method for comparing results.

The tool's adaptability ensures that it remains relevant as new research questions arise, offering the scientific and industrial communities a reliable platform for reproducible design automation.

Looking forward, Odatix's continued evolution will expand its support for additional EDA tools, such as Intel Quartus Prime and F4PGA [24]. It is planned to integrate Design Space Exploration algorithms into Odatix, similar to what Dovado [17] achieves, to extend its capabilities for architectures with a large number of parameters. Other features are currently under development, such as the ability to run syntheses in parallel on the same configuration at user-defined frequency intervals. This functionality will allow for the comparison of synthesized architectures based on frequency constraints, with architectures synthesized near the maximum frequency tending to use more resources and be more energy-intensive. Future work also includes enhancing the user interface to enhance further comparison.

CRediT authorship contribution statement

Jonathan Saussereau: Writing – original draft, Validation, Software, Conceptualization. **Christophe Jego:** Writing – review & editing, Supervision. **Camille Leroux:** Writing – review & editing, Supervision. **Jean-Baptiste Begueret:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] Babu P, Parthasarathy E. Reconfigurable FPGA architectures: A survey and applications, 102 (1) 143–156 http://dx.doi.org/10.1007/s40031-020-00508-y.

[2] Koch D, Ziener D, Hannig F. FPGA versus software programming: why, when, and how? In: Koch D, Hannig F, Ziener D, editors. FPGAs for software programmers. Springer International Publishing; p. 1–21. http://dx.doi.org/10.1007/978-3-319-26408-0_1.

[3] Kin J, Gupta M, Smith WM. Application specific integrated circuits.

[4] Newton A, Sangiovanni-Vincentelli A. CAD tools for ASIC design, 75 (6) 765–776, http://dx.doi.org/10.1109/PROC.1987.13798.

[5] Committee DAS, et al. IEEE standard for systemverilog unified hardware design, specification, and verification language standard IEEE 1800.

[6] Bachrach J, Vo H, Richards B, Lee Y, Waterman A, Avižienis R, et al. Chisel: Constructing hardware in a Scala embedded language. In: Proceedings of the 49th annual design automation conference. Association for Computing Machinery; p. 1216–25. http://dx.doi.org/10.1145/2228360.2228584.

[7] Waterman A, Lee Y, Patterson DA, Asanovic K. The RISC-V instruction set manual, volume I: User-level ISA, version 2.0.

[8] Asanović K, Avizienis R, Bachrach J, Beamer S, Biancolin D, Celio C, et al. The rocket chip generator. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html.

[9] Bandara S, Ehret A, Kava D, Kinsy M. BRISC-V: an open-source architecture design space exploration toolbox. In: Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays. Association for Computing Machinery; p. 306. http://dx.doi.org/10.1145/3289602.3293991.

[10] Mantovani P, Margelli R, Giri D, Carloni LP. HL5: a 32-bit RISC-V processor designed with high-level synthesis. In: 2020 IEEE custom integrated circuits conference. p. 1–8. http://dx.doi.org/10.1109/CICC48029.2020.9075913.

[11] Saussereau J, Leroux C, Begueret J-B, Jego C. AsteRISC: a size-optimized RISC-V core for design space exploration. In: 2023 IEEE international symposium on circuits and systems. p. 1–5. http://dx.doi.org/10.1109/ISCAS46773.2023.10181330.

[12] Saussereau J, Jego C, Leroux C, Begueret J-B. Design and implementation of a RISC-V core with a flexible pipeline for design space exploration. In: 2023 30th IEEE international conference on electronics, circuits and systems. p. 1–5. http://dx.doi.org/10.1109/ICECS58634.2023.10382774.

[13] Wolf C. PicoRV32—A size-optimized RISC-V CPU. URL https://github.com/YosysHQ/picorv32.

[14] Ansel J, Kamil S, Veeramachaneni K, Ragan-Kelley J, Bosboom J, O'Reilly U-M, et al. OpenTuner: An extensible framework for program autotuning. In: 2014 23rd international conference on parallel architecture and compilation techniques. p. 303–15. http://dx.doi.org/10.1145/2628071.2628092.

[15] Kinsy MA, Pellauer M, Devadas S. 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. Association for Computing Machinery; p. 125–34. http://dx.doi.org/10.1145/2435264.2435287.

[16] Shao YS, Reagen B, Wei G-Y, Brooks D. 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. p. 97–108. http://dx.doi.org/10.1109/ISCA.2014.6853196.

[17] Paletti D, Conficconi D, Santambrogio MD. Dovado: an open-source design space exploration framework. In: 2021 IEEE international parallel and distributed processing symposium workshops. p. 128–35. http://dx.doi.org/10.1109/IPDPSW52791.2021.00027.

[18] Shalan M, Edwards T. Building OpenLANE: A 130 nm OpenROAD-based tapeout-proven flow : Invited paper. In: 2020 IEEE/ACM international conference on computer aided design. p. 1–6.

[19] Gingold T. GHDL. URL https://github.com/ghdl/ghdl.

[20] Wolf C, Glaser J. Yosys - A free verilog synthesis suite.

[21] Weicker RP. Dhrystone: A synthetic systems programming benchmark 27 (10) 1013–1030. http://dx.doi.org/10.1145/358274.358283.

[22] Saussereau J. AsteRISC. URL https://github.com/jsaussereau/AsteRISC.

[23] Edwards RT, Shalan M, Kassem M. Real silicon using open-source EDA 38 (2) 38–44. http://dx.doi.org/10.1109/MDAT.2021.3050000.

[24] Alliance C. F4PGA - FOSS flow for FPGA. URL https://github.com/chipsalliance/f4pga.