Using RTL Architect for Designing the Ultra-Low Power EdgeVision SoC

Mikail Yayla, Clifford Leon D'mello, Georg Ellguth, Uwe Steeb, Tim Leuchter, Marcus Pietzsch, Holger Eisenreich

Racyics GmbH

Dresden, Saxony, Germany

racyics.de

ABSTRACT

The SoC design process from specification to tapeout requires functionally correct RTL and several steps during physical design, such as synthesis and place & route. During design, a multitude of challenges can arise that demand RTL modifications. Assessing their impact on PPA involves time-intensive feedback loops, possibly delaying tapeout. RTL Architect by Synopsys aims to optimize this process by enabling RTL designers to estimate quickly the impacts of their RTL changes on PPA.

In this paper, we use RTL Architect in the early design stages of our ultra-low-power EdgeVision SoC that uses Adaptive Body Biasing based on GlobalFoundries 22nm FDX technology. We show that assessing the PPA-impact of several types of RTL and clock modifications guides our design, while uncovering potential issues early. We present the results of our analyses and conclude by discussing the benefits and challenges of integrating RTL Architect in our design process.

Table of Contents

Using RTL Architect for Designing the Ultra-Low Power EdgeVision SoC 1

1. Introduction 3

2. SoC Design Process from Specification to Tapeout 4

3. Design Challenge 6

3.1. Challenges in the SoC Design Process 6

3.2. RTL Architect 7

4. RTL Architect Prerequisites and Flow 8

4.1. RTL Architect Prerequisites 8

4.2. RTL Architect Flow 9

5. Results 10

5.1 Experiment Setup 10

5.2. AI Accelerator Configuration 10

5.3. Memory Configuration 12

5.4. Modifying Clock Constraints 13

5.5. Discussion 15

6. Conclusion 15

References 15

Table of Figures

Figure 1. EdgeVision SoC developed at Racyics in cooperation with project partners. Additional components are connected to the SoC interconnect, here we show a subset for brevity. 4

Figure 2. SoC design flow. 5

Figure 3. Flow for using RTL Architect. 9

Figure 4. Area (left) and power (right) of the EdgeVision SoC. AI accelerator configurations: 256, 512. 11

Figure 5 Floorplan of the EdgeVision SoC with two AI accelerator configurations (left: 256 MACs, right 512 MACs). The figures show the logic areas (blobs, colored points show contention) and the memory components (rectangular units). The I/O pins are the squares on the edges. The floorplan is created with the autofloorplan feature of RTL Architect. 11

Figure 6. Area (left) and power (right) of the EdgeVision SoC with different SRAM sizes. 12

Figure 7. EdgeVision SoC with 1 MB (left, for comparison) and 2 MB (right) SRAM modules. There is logic (blobs, coloured points show contention) and the memories (rectangular units). Created with the autofloorplan feature of RTL Architect. 13

Figure 8. Area (left) and power (right) of the EdgeVision SoC with different clock constraints. 14

Figure 9. Layout view of the EdgeVision SoC in RTL Architect. Highlighted are the memories (rectangular blocks), the main processor (left side, purple), the AI accelerator (middle right side, magenta), and the bus interconnect (middle, blue). The other SoC components are comparatively small and barely visible in this view, therefore they are not labelled. 14

1. Introduction

A System-on-Chip (SoC) integrates multiple components into a single chip, such as embedded processors, memories, interfaces to external devices, interconnecting bus matrices and potentially many other components. As the requirements on modern SoCs evolve, increasing the demand on functionality, the number and complexity of the integrated components rises as well. This leads to several benefits. SoCs are cost-efficient for the functionality they offer, since required IP are included inside one single chip, which is tailored towards the power, performance, and area (PPA) sweet spots of dedicated use cases. Due to the proximity of the different blocks included in SoCs it is possible to optimize performance and power efficiency. Short paths enable custom communication techniques between them. SoCs also operate at lower voltages and thus lower power and require less area & volume compared to discrete systems. This makes them an ideal choice for portable and embedded systems.

A typical example of an SoC is our EdgeVision SoC (shown in Figure 1). It is an ultra-low power SoC based on GlobalFoundries 22nm FDX technology that exploits Adaptive Body Biasing [1]. The target application of the EdgeVision SoC is executing image-processing machine learning tasks. It therefore features a powerful AI engine, an advanced CPU, multiple graphics components, display and camera interfaces, highly efficient Racyics SRAM, and a high-speed bus to connect all the latter and many other peripheral components.

The design of SoCs from specification to tapeout, also called RTL2GDS flow, is a challenge. Multiple complex design steps need to be followed, i.e. specification, RTL design, synthesis, verification, place & route, and more. Several iterations of these steps are typically necessary to reach a functionally correct design that also meets the PPA targets. This process takes significant amounts of time, many months up to years, and needs to be planned well in advance to finish before the date of the tapeout.

In the RTL2GDS flow, RTL changes are frequently requested by the physical designers. When the RTL is changed by the RTL designers, the physical designers need to estimate again the PPA metrics to check whether the targets are met. The RTL designers in turn need to wait. This issue is further exacerbated by the extensive run times of state-of-the-art EDA tools. Therefore, this feedback loop is one of the most time consuming and therefore costly steps in SoC design.

The RTL Architect [2] software by Synopsys directly addresses this issue by enabling RTL designers to quickly estimate the impact of the RTL changes on the PPA metrics, avoiding the costly feedback loop. To the best of our knowledge, there exists no publication or report that demonstrates the use of RTL Architect in a cutting-edge real-world industry design process.

Contributions of the paper:

- In Chapter 2, we give an overview of the steps in the RTL2GDS design flow, which is required to successfully design an SoC. Specifically, we cover the steps from specification to tapeout.
- We discuss the details of the design challenges above and how we aim to overcome them with RTL Architect in Chapter 3.
- In Chapter 4, we present the RTL Architect design flow and our approach of using RTL Architect for our SoC design.
- In Chapter 5, we present a real-world industry scenario in which we evaluate early in the design process the PPA of our low-power EdgeVision SoC. Specifically, we assess the impact of different RTL and clock configurations on PPA.

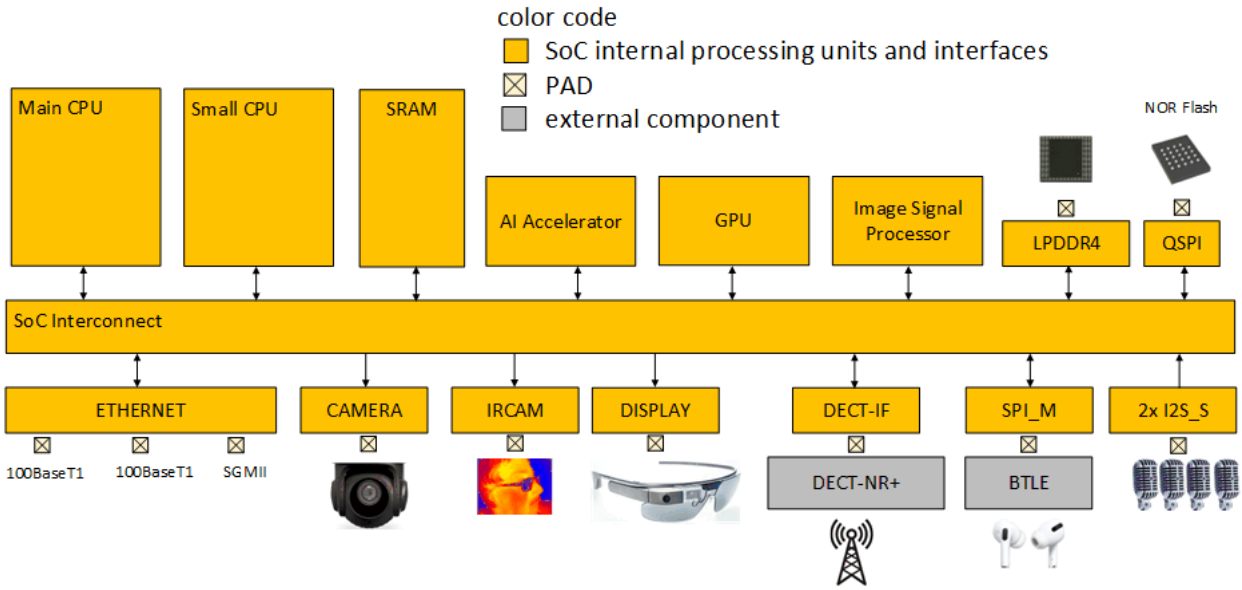

Figure 1. EdgeVision SoC developed at Racyics in cooperation with project partners. Additional components are connected to the SoC interconnect, here we show a subset for brevity.

2. SoC Design Process from Specification to Tapeout

A predefined design flow, i.e. the RTL2GDS flow, enables the planning and managing of the design process. Resources in manpower and tools are allocated according to the needs of the flow steps. The time required in the individual steps and thus in the entire design process is estimated well in advance. In short, following the predefined flow makes the design process predictable while at the same time minimizing the chances for unexpected time bottlenecks and errors.

Still, the RTL2GDS design flow is highly complex. The complexity is managed by abstractions provided by the electronic design automation (EDA) tools, where details are abstracted away and brought back to the forth only when required. Engineers working in one of the steps of this flow should know the abstractions and issues that are “above and below” their respective steps. This leads to more communication among engineers, higher efficiency, better decisions, and ultimately to shorter development cycles towards successful tapeouts. In this flow, targets for key PPA metrics serve as a guide and can be estimated with increasing confidence the further the design process.

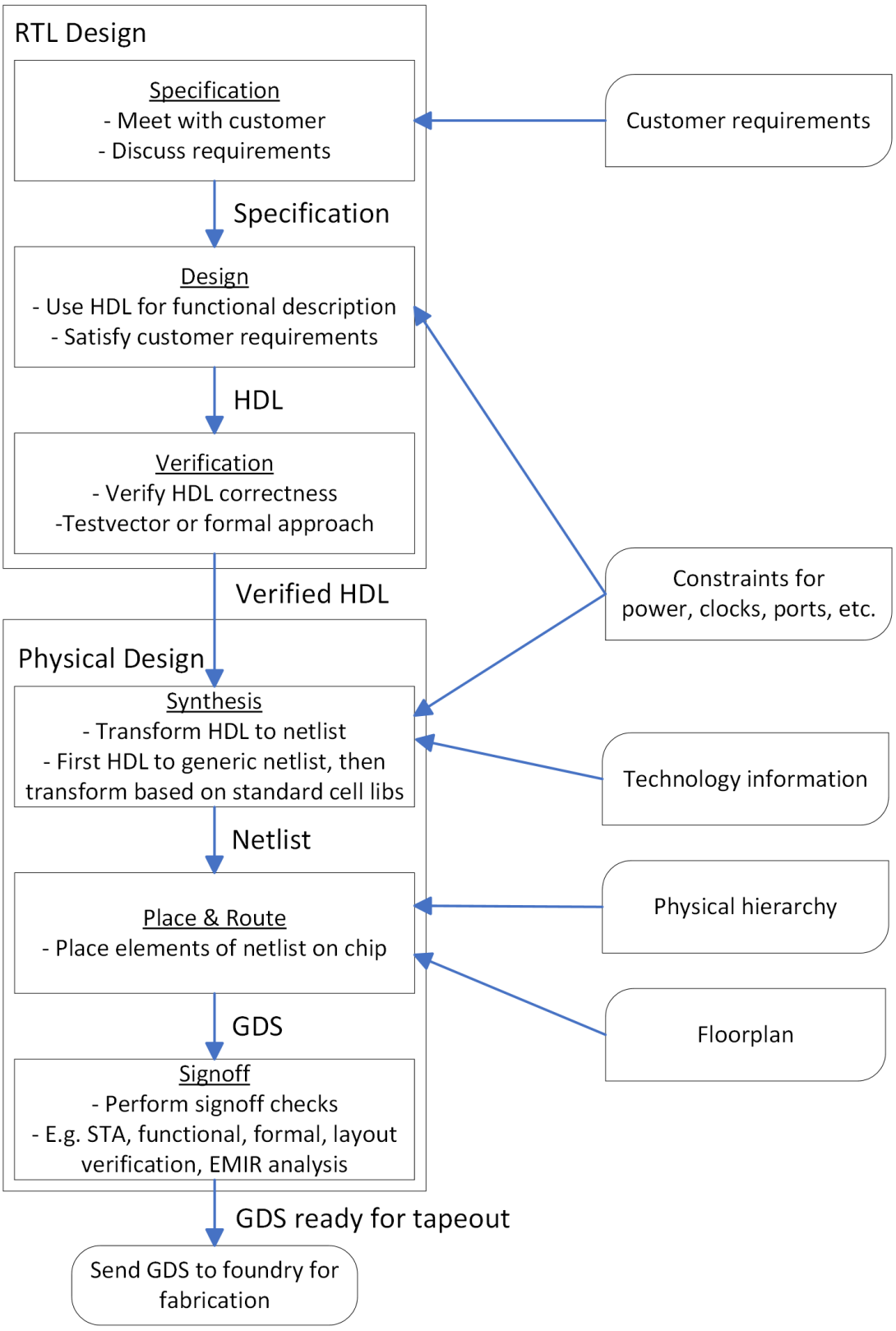

Figure 2. SoC design flow.

In the following, we present the **steps in the RTL2GDS design flow of SoCs**. The flow is also summarized in form of a diagram in Figure 2.

- **Specification:** At first, the service provider and the customer meet to discuss the requirements of the design. Requirements are typically high-level descriptions of the design and its use cases, along with targets such as functionality, area, power, and price. The requirements are recorded and may need to be revisited again at a later design stage. It is important that the two parties are “on the same page” before the development begins.
- **Design of Required Functionality:** The design functionality is realized using a Hardware Description Language (HDL), such as SystemVerilog or VHDL. Typically, a complex component is broken down into less complex components, until simple components remain. This way, individual tasks can be distributed and after completion connected again to form complex components.
- **Verification:** The implemented HDL is typically verified using a testvector based approach, i.e. using testbenches, adhering to the UVM specification, or by formal verification. It is of utmost importance that the implemented HDL is functionally correct. Any bugs that can be fixed on the HDL-level should also be fixed on this level. It becomes increasingly difficult to correct errors the further the design process advances, therefore achieving functional correctness early in the design process is highly valuable. The functional equivalence between the HDL and its lower-level representations needs to be proven at any transformation step, for which the verification techniques or other equivalence checkers can be employed.
- **Synthesis:** The HDL of the circuit is transformed into a gate-level netlist. This includes several steps. First, the HDL is translated into a generic netlist, which consists of generic logic cells such as AND, OR, Flip Flops and many others. Then these generic cells are mapped to cells that are specified in the standard cell libraries. Each standard cell is a transistor design based on the given technology and is representing a certain logic function. Afterwards, the netlist is optimized based on the given constraints with regards to metrics such as timing, area, and power consumption.
- **Place and Route (P&R):** The elements of the gate-level netlist are placed on the chip along with the interconnections. It is important here that the parasitic effects and wire characteristics are considered.
- **GDS**: A final layout file is produced that encapsulates the information from the steps above. This file is the last step of the P&R. It is sent out to the foundry for fabrication after all signoff checks are completed and clean.
- **Signoff:** To verify that the GDS represents the original HDL and that the sent GDS will lead to a working chip, several signoff checks are necessary, such as static timing analysis, functional, formal, layout verification, as well as electromigration and IR drop analysis.
- **Foundry:** The Actual chip is produced on the wafer, using techniques such as photolithography, etching, and many others.

3. Design Challenge

3.1. Challenges in the SoC Design Process

In the RTL2GDS flow described above, RTL-level modifications are required to solve a multitude of design issues. Larger RTL changes or modifications to design constraints (e.g. clock frequencies) require to rerun the RTL2GDS flow, which causes a costly feedback loop. Even state-of-the-art EDA tools require extensive run times for complex SoCs that may span multiple days. These delays may impact the tapeout and project schedule, increasing the risk of late silicon arrival and customer/partner dissatisfaction.

In the design process, PPA metrics are critical for a successful and profitable design, as they determine the cost in power, performance, and area. They are also used to compare different design

configurations among each other to determine the most suitable one. Being able to analyze the influence of individual changes to the design can help guiding the design process early on and show potential problems in the specification or architecture before a real physical implementation has taken place.

In the following examples we list several design changes, related to RTL, constraints, or power configurations, that may lead to a high impact on the PPA results after physical implementation. Usually, the long run-times of the RTL2GDS flow does not allow a quick feedback of these results to the designers.

RTL Modification: During design specification, participating parties agree on a set of features to be included in the design. IPs that realize the desired functionality then need to be integrated. Typically, IPs have multiple configuration options. To save resources such as area, IPs can be configured to exclude functionality that is not required (for example floating point units in microprocessors). When the specification changes, different configurations or parametrizations of IPs may be required, which in turn lead to RTL modifications. Other examples of RTL-level modifications include fixing design issues. It may for example turn out that the specified RTL is not able to be realized in physical design, or RTL bugs may become evident in later stages of the design process.

Clock Configuration: The clock frequency determines the speed at which the design operates and may have a large impact on PPA. In complex SoCs, several portions of the design receive different clocks. A collection of design elements that receive an identical or synchronous clock (e.g. the same clock frequency) is called a clock domain. Clock domains enable to control the timing of the design, leading to benefits in synchronization among components in the same clock domain and in power management. The challenge is to find suitable clock frequencies for the clock domains, which may require multiple design iterations. Different clock frequencies may lead to different PPA results since techniques such as pipelining or other lower-level methods in physical design are employed. It is not desirable to specify a clock frequency that is too high, it may lead to timing violations which need further investigation. A low clock frequency also may not exhaust the full potential of the design. Furthermore, too many clock domains require logic to resolve issues caused by clock domain crossing, while too few clock domains may cause long timing paths.

Power Configuration: The supply of a design determines the available energy for it. In complex SoCs, several portions of the design have different power supplies. A collection of design elements that are connected to the same power supply is called a power domain. The benefit is that each power domain can have independent power control, enabling power optimizations, such as turning on or off on power domains on demand or dynamic voltage scaling. Finding suitable power domain configurations is also a challenge. A design with many power domains requires additional components (e.g. level shifters) so that communication across power domains is possible. It is also not desirable to integrate all components using too few power domains, as some components are not used very often (e.g. they can be turned off) or may have different power requirements.

In summary, finding suitable RTL, clock, and power configurations forms a challenging co-optimization problem. When one ingredient changes, the RTL2GDS flow steps and therefore EDA tool runs have to be repeated. Note that the earlier in the RTL2GDS flow the modifications, the more impact they can have on the below levels, and to assess the impact also requires more effort.

3.2. RTL Architect

RTL Architect enables RTL engineers to assess the impact of RTL changes on key metrics such as power, performance, area, and congestion. It enables the fast and accurate estimation of these key metrics, which leads to faster development and time to tapeout. Once RTL Architect is configured with information from physical implementation, the costly feedback loop between RTL engineers and physical design engineers is avoided.

We use RTL Architect in a real-world industry example, i.e. our EdgeVision SoC with Adaptive Body Biasing based on GlobalFoundries 22nm FDX technology, to solve the above challenges. We first

supply RTL Architect with the required information from physical implementation, with the constraints, and with the RTL. Then, we define several different scenarios with respect to IP configurations, memory size settings, and clock configurations. The results of this analysis allow us to quickly estimate the effect of design decisions and react on them in an early stage, saving cost and development time for the entire design process.

4. RTL Architect Prerequisites and Flow

Our goal is to estimate the PPA metrics for different configurations of our EdgeVision SoC. To this end, we supply RTL Architect with the files and information it requires. Our used flow for RTL Architect is also shown in Figure 3.

4.1. RTL Architect Prerequisites

We provide RTL Architect with the following physical implementation files.

- **.tf:** The technology file specifies technology information about the target technology of the design. Physical and electrical descriptions for the metal and via layers, and technology routing information are included as well.
- **.lib:** The reference library is in ASCII format. It is used to specify timing characterization of each cell (cell delays, limits on cell input transition and output load, timing checks) and power parameters (leakage and internal power) of cells, derived from simulations.
- **.db:** This is the compiled version of .lib, it is in binary format.
- **.lef:** The library exchange format is used for abstracted physical layout information like cell size, allowed cell orientation, pin positions, routing blockages.
- **.ndm** (new data model): This is a library container including physical data from lef/gds and in addition timing information from .db files. The ndm files are created based on all the above physical implementation files using Synopsys Library Manager.
- **.tluplus:** It provides extraction technology information for calculating the parasitics of interconnects.

Furthermore, constraints are required to specify information regarding power, clocks, and ports.

- **Clocks:** Information about clocks, such as period, latency, jitter or uncertainty (for setup and hold times), transition limits, their source, etc., are given in the clock constraints.
- **Ports:** Input and output conditions for the ports, such as input transition, output load, input and output delay, are given in the port constraints.
- **UPF**: This is an IEEE standard for specifying the power intent [3]. It describes the power supplies, power domains, power control/retention/isolation/level-shifter strategies. It also describes the different power states of a system in a Power State Table (PST).

Finally, the RTL of the SoC needs to be available.

- **RTL files:** They describe the functionality of the design. The RTL will be synthesized to produce the final ASIC, based on the information from the physical implementation and the constraints.

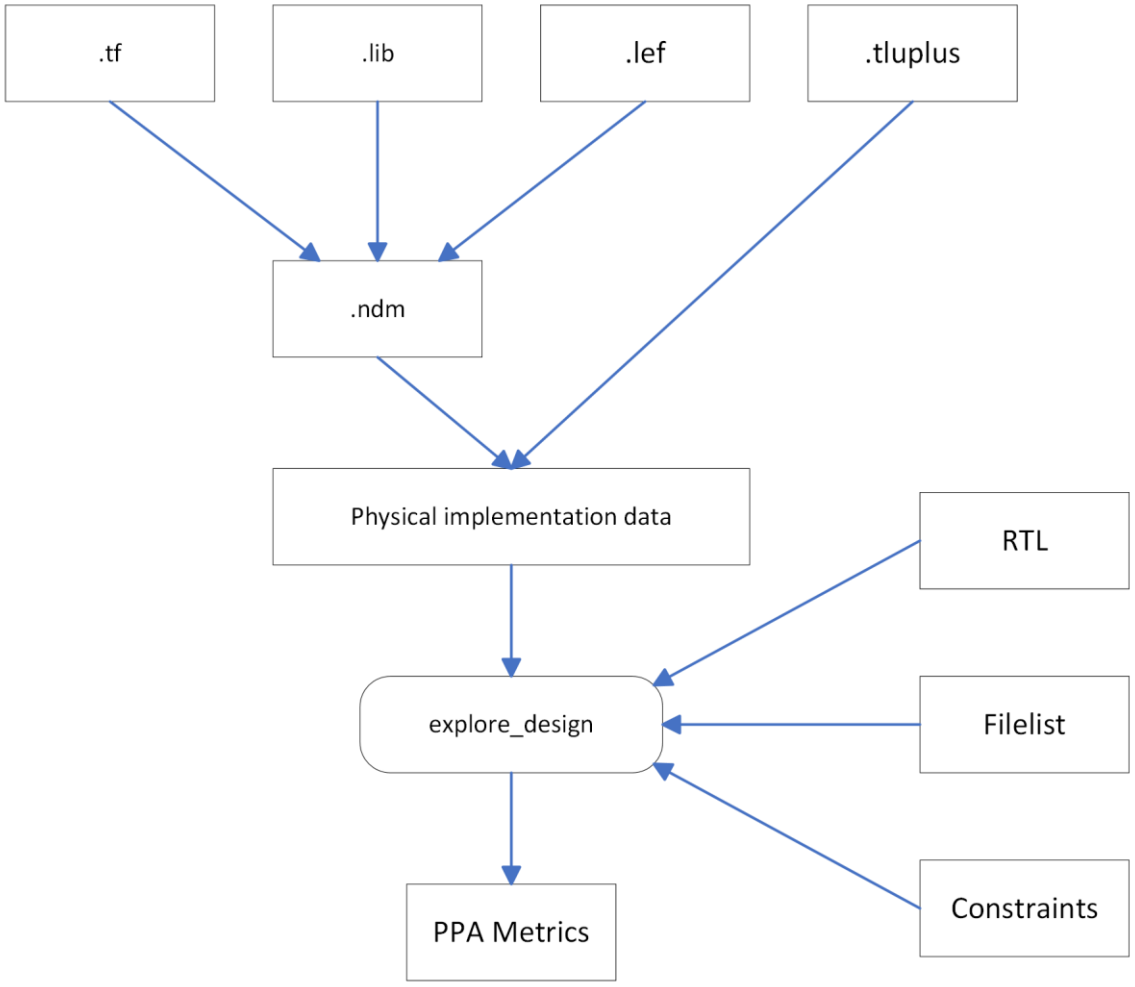

Figure 3. Flow for using RTL Architect.

4.2. RTL Architect Flow

In the following section we describe the RTL Architect flow. Our goal is to estimate the PPA metrics of our configuration with the explore_design command, for which several steps have to be completed first.

- **RTL Generation:** We use RTL generation tools to create the RTL of our SoC. Many of our components are third party IPs, generated through the tools provided with them. To connect the individual IPs and create the RTL of our SoC, we use the open-source tool ICGlue [4].
- **RTL Filelists:** Based on the created RTL, we create filelist for our RTL in form of tcl-scripts, so that RTL Architect can find and load them.
- **Open Libraries:** We create the libraries (in ndm-format, see Section 4.1) and provide them to RTL Architect for later usage.
- **Analyze Step**: This step in RTL Architect is initialized by the “analyze” command. It analyzes the RTL source and converts it into a format required for further processing.
- **Elaborate Step**: This step compiles the RTL and builds the design based on the information generated in the analyze step.
- **Setting the Top Module**: The specified module is set as the top-level design. It links the entire design, and creates a single block to be used for the remainder of the synthesis flow.
- **Load Constraints**: This loads the power, timing, and port constraints of our SoC.

- **Set Explore Options:** In this step, the options for the explorations runs are set. We configure the target utilization and the aspect ratio of the floorplan**.**
- **Explore the Design**: Finally, the "explore_design" command starts exploration runs for the exploration options above.
- **Assessing Experiment Data:** After the experiments finish, we can view a table provided by the RTL Architect reference flow that reports the PPA metrics of the configurations. Based on this information, further design decisions can be made.

5. Results

We consider three different scenarios to perform experiments using RTL Architect for our EdgeVison SoC. In the first scenario (Section 5.2), we explore design options for the AI accelerator. In the second scenario (Section 5.3), we explore different embedded memory configurations of the SoC. In the third scenario (Section 5.4), we present the results of constraining the main clock domain of the SoC with three different clock frequencies.

5.1 Experiment Setup

To run the exploration experiments, we use RTL Architect with the flow described in Section 4. We execute a modified version of the supplied reference flow that includes our RTL, constraints, and physical implementation data. For each experiment we reserve a machine with 100 GB RAM and two CPU cores.

We constrain the clock frequencies of the main CPU and the AI accelerator to the specified value in the experiments (e.g. 400 MHz). The power intent is set up with two power domains, where the first one is the toplevel power domain operating at 0.8V and a nested power domain operating at 0.55V including the memories and main core logic. We choose the utilization factor of 0.6 and a 1:1 aspect ratio for the device. These values are typical for an SoC design. The experiments are run in typical PVT corner at 25°C.

We use the autofloorplan feature of RTL Architect. Using this feature, no information regarding the floorplan is required for PPA estimations. The tool decides the dimensions of the chip itself, without any user input. This helps us in estimating early the required chip dimensions for tapeout.

The metrics computed by RTL Architect include congestion, timing, power, area, and others. The results are returned by the RTL Architect reference flow in form of an interactive website at the end of the exploration runs.

5.2. AI Accelerator Configuration

The AI accelerator is a highly efficient accelerator IP tailored towards use in SoCs. It is compatible with the CPUs and other common processors. An ecosystem with a mature toolchain is also available. The use cases of the AI accelerator include tasks related to vision or voice, such as object detection, image segmentation, and language detection and processing. These properties match the requirements of our EdgeVision project.

The configuration of the AI accelerator is decided on the specification level and configured on the RTL level. At this stage, the impact of RTL changes on the PPA results of the SoC is unknown. In the AI accelerator, the number of 8x8 MAC operations performed per cycle can be configured with 256 or 512. With a higher number of 8x8 MACs, a larger number of parallel operations can be computed per cycle. This enables the implementation of more advanced applications. However, the cost of a higher-performing IP is in chip area and power usage. The extent of the costs depends on the implementation and the technology. This motivates us to perform an evaluation with regards to the number of 8x8 MAC operations per cycle.

RTL Architect enables us to perform evaluations with respect to the number of 8x8 MACs per cycle. The returned PPA results are used to evaluate the costs of the different AI accelerator configurations early in the design. This not only provides metrics-based decisions in the design process, but also enables the estimations to be communicated to the customer.

We perform the RTL Architect explore run and report the area and power usage. The clock frequency for the AI accelerator and the main CPU is constrained to 400 MHz (the other components are constrained based on their information in the datasheets).

In Figure 4, we observe that the area increase in die size between an AI accelerator with 256 and 512 8x8 MAC units is 1.1%, while the total power increase is 18%. Based on this data, a low die area increase is expected, however, the total SoC power is increased by a non-negligible amount. In Figure 5 we show the floorplans of the two configurations. The two configurations lead to different placements of logic and memory, and the used area does not increase by a large margin. Note however that the 512-version has around 20% more flip flops. Our conclusion is that an SoC with an AI accelerator using 512 8x8 MACs should only be used when as low power as possible is required.

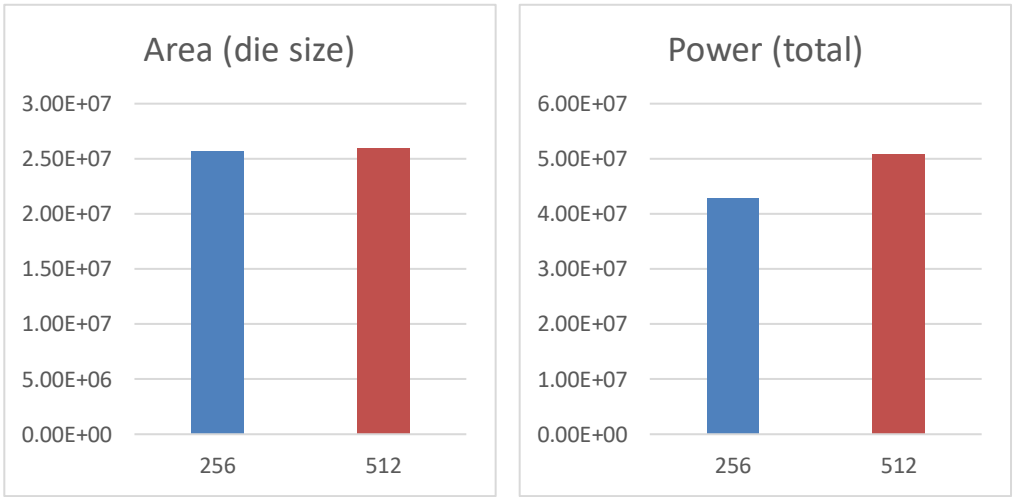

Figure 4. Area (left) and power (right) of the EdgeVision SoC. AI accelerator configurations: 256, 512.

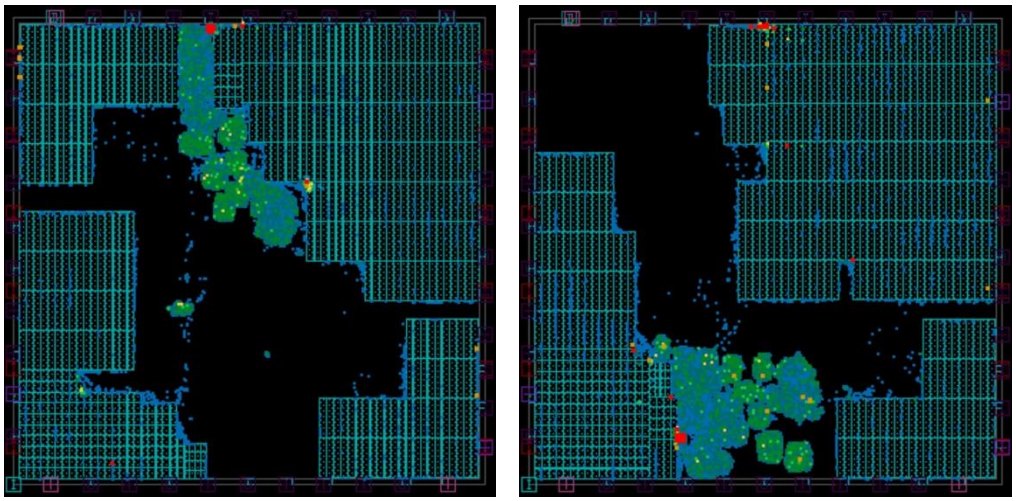

Figure 5 Floorplan of the EdgeVision SoC with two AI accelerator configurations (left: 256 MACs, right 512 MACs). The figures show the logic areas (blobs, colored points show contention) and the memory components (rectangular units). The I/O pins are the squares on the edges. The floorplan is created with the autofloorplan feature of RTL Architect.

5.3. Memory Configuration

In our SoC, we use Racyics SRAM that employs ABX technology realizing ultra-low power storage with one-cycle data-access. It also features power down and retention mechanisms.

The floorplans (see Figure 5) of our EdgeVision SoC show that a large portion of the area is used for integrating the SRAMs. Although information regarding area and energy usage of the memories is available in data sheets, logic surrounding the memory for connecting to the logic of the SoC needs to be considered. This motivates us to perform an exploration with different SRAM configurations using RTL Architect. Our goal is to find out how much chip area and power different memory configurations would use when integrated and connected in the SoC, so that early decisions can be made about the memory size. This not only helps the SoC designers to estimate resource usage and cost, but it also helps the customers. They can get quick feedback on the cost of different SoC configurations.

Our customers or partners who develop the applications require a range of 1 – 2 MB per SRAM memory module. Our SoC has in total four SRAM memory modules. With larger memories, they can also implement more advanced features. However, the SoC costs also need to be considered. Therefore, we explore the memory sizes 1 and 2 MB for each module leading to a total size of 4 or 8 MB respectively.

With explore runs using RTL Architect, we report the die area and the power usage of the SoC. In the results in Figure 6 the chip area usage has an increase of 53% when comparing 1 to 2 MB. For power the increase from 1 to 2 MB is 7%. Based on this evaluation, the main cost of more memory lies in area. We show the floorplan comparison between the 1 MB (same as in Figure 5, for reference) and 2 MB experiment in Figure 7.

This gives useful early insight into the costs associated with different memory configurations. The evaluations indicate that 1 MB SRAM modules are a reasonable choice for the EdgeVision SoC, therefore we use it in further explorations. If required by customers, the memory module size can also be increased at higher cost.

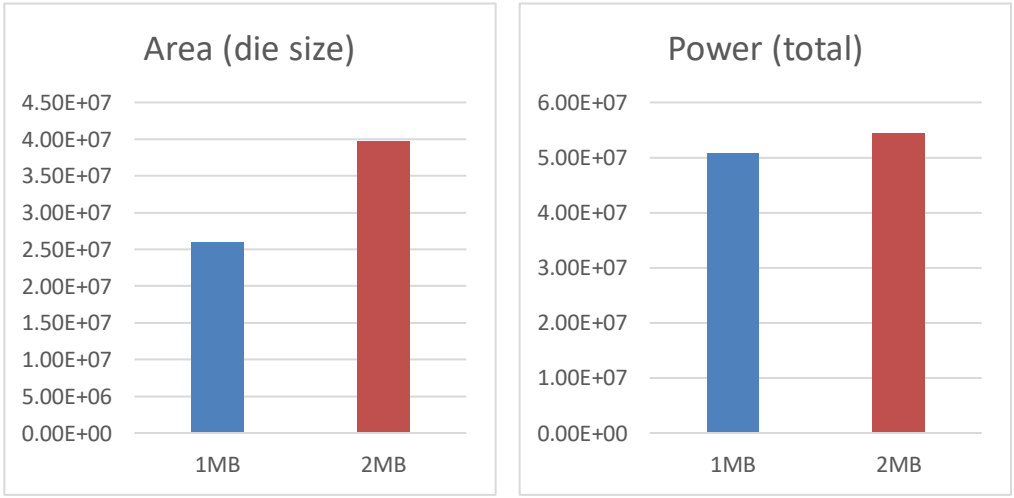

Figure 6. Area (left) and power (right) of the EdgeVision SoC with different SRAM sizes.

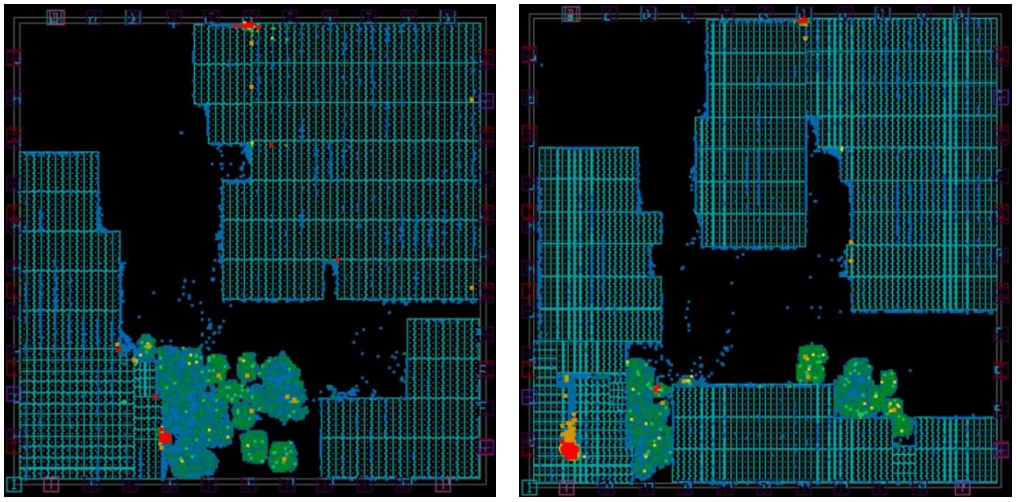

Figure 7. EdgeVision SoC with 1 MB (left, for comparison) and 2 MB (right) SRAM modules. There is logic (blobs, coloured points show contention) and the memories (rectangular units). Created with the autofloorplan feature of RTL Architect.

5.4. Modifying Clock Constraints

Clock constraints determine the clock frequency that the clocks in the design must achieve. In other words, they determine how fast individual parts of the design should operate. Suitable values are typically determined using domain knowledge or empirical evaluations. The experiments above constrain the clock of the AI accelerator, main CPU, and the SRAMs to 400 MHz. Although RTL Architect does not optimize the design for timing closure, effectively eliminating all violated timing paths, it supports frontend designers in finding modules or structures in the design that may not reach the expected frequencies. It reports both the worst negative slack (WNS) and total negative slack) (TNS) that give a good estimation about the margin and number of affected logic elements in the design.

Here, our goal is to evaluate the impact of changes in the clock constraints on the PPA of the EdgeVision SoC. To this end, we reduce the frequency of the main clock domain of the EdgeVision SoC containing the AI accelerator, main CPU and SRAM cells. In addition to the target frequency of 400 MHz, we selected 200 MHz and 100 MHz to see the impact on area and power consumption.

The area and power results are shown in Figure 8. When increasing the clock frequency from 100 MHz to 200 MHz, the used area approximately stays the same, but the power increases by 95%. When increasing from 200 MHz to 400 MHz, the area again stays approximately the same and the power increases by an additional 150%.

Based on this evaluation, 100 MHz is a suitable preliminary clock frequency for the EdgeVision SoC, for which timing closure can be easily achieved. This shows that this or a slightly higher clock frequency can realistically be achieved. The timing results look reasonable for 100 and 200 MHz, but for 400 MHz we run into challenges to close the timing for our SoC architecture in the target technology. This evaluation with RTL Architect helps us in deciding early the range of the clock frequencies for the design, which will speed up the remaining steps towards tapeout. The floorplan of the EdgeVision SoC with 200 MHz and labelled logic regions is shown in Figure 9.

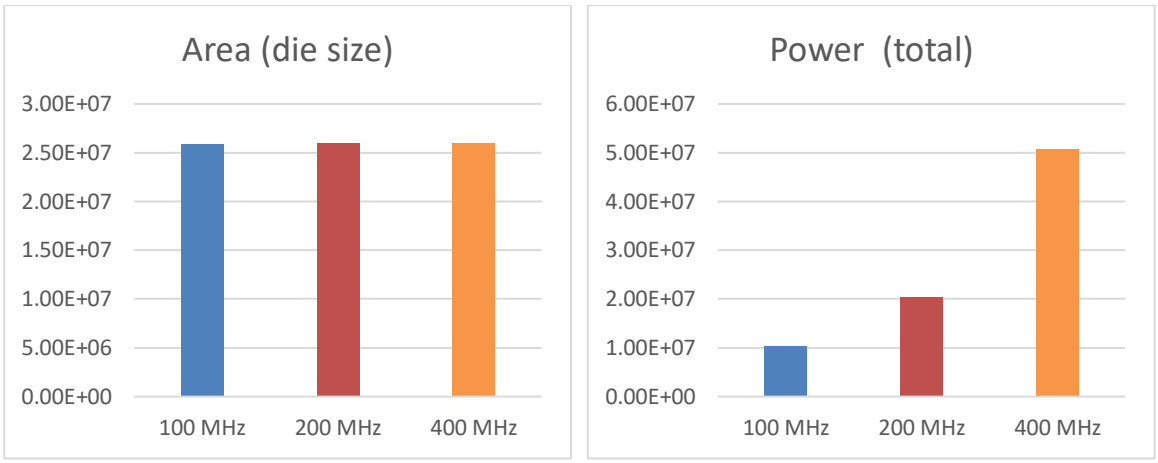

Figure 8. Area (left) and power (right) of the EdgeVision SoC with different clock constraints.

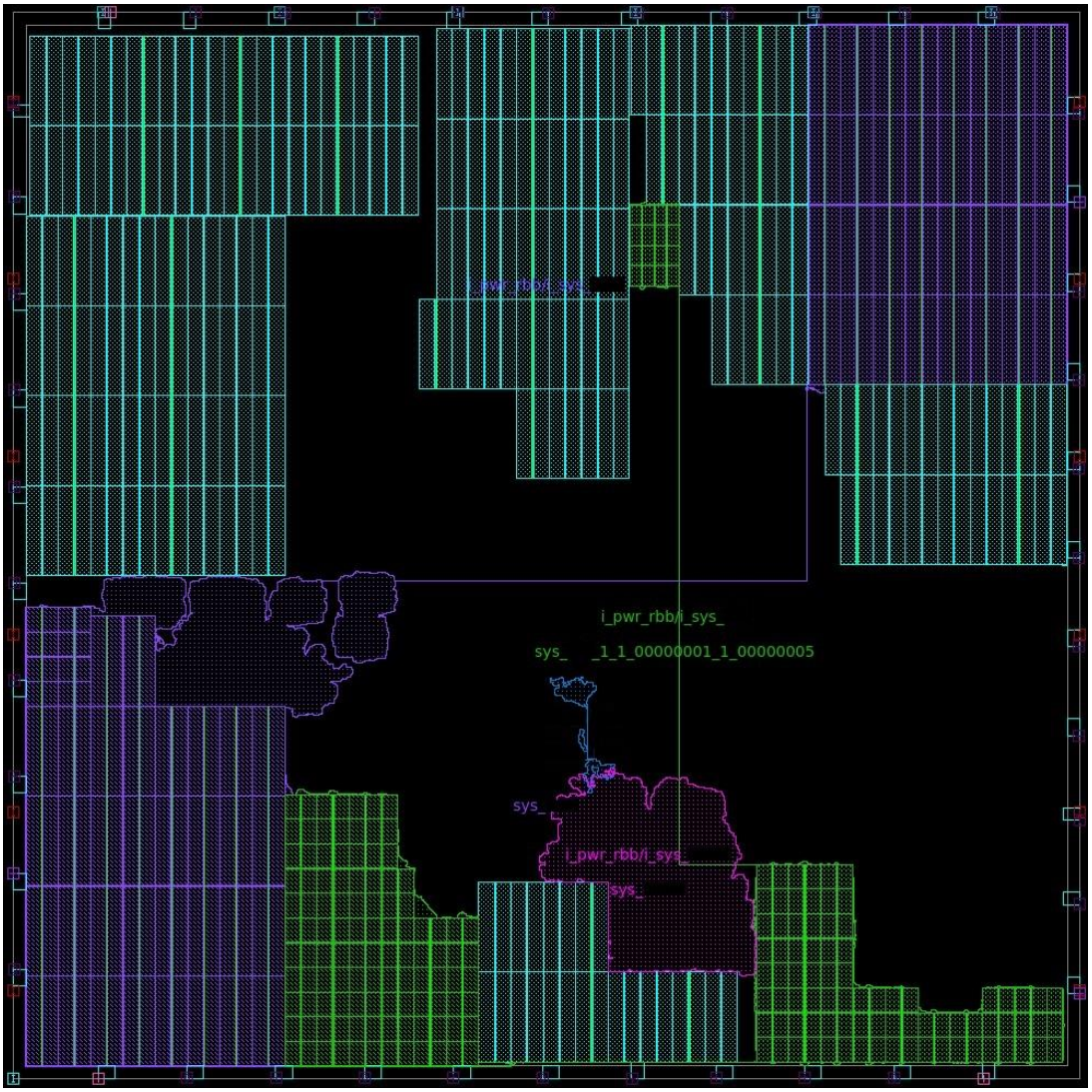

Figure 9. Layout view of the EdgeVision SoC in RTL Architect. Highlighted are the memories (rectangular blocks), the main processor (left side, purple), the AI accelerator (middle right side, magenta), and the bus interconnect (middle, blue). The other SoC components are comparatively small and barely visible in this view, therefore they are not labelled.

5.5. Discussion

RTL Architect enables collaboration between engineers that work on different steps of the RTL2GDS flow. The physical design and RTL engineers sat together to solve design issues. We expect that the increased communication will lead to mature designs within a shorter time frame, while less time is wasted on avoidable issues or errors. RTL Architect is also considerably faster than other tools for performing PPA estimations. One experiment to achieve PPA results required around 5 hours in our setup, while other state-of-the-art tools may require several days of experiment run time for a similar estimation. Furthermore, the estimation of the preliminary metrics lead to valuable early explorations of SoC configurations as they guide us towards better design decisions early in the design process.

There are also a few challenges when using RTL Architect. The tool has a steep learning curve for RTL engineers that are unfamiliar with physical design. Support by physical design engineers is required, especially when setting up the physical design data for RTL Architect. However, when the environment with regards to physical design is set up, valuable results can be acquired in a short timeframe. To be able to use RTL Architect, other tools or licenses may also be required, such as Synopsys IC Compiler II [5] (for ndm files) and Synopsys PrimePower [6] (for UPF analysis). For resources, although RTL Architect estimates PPA metrics significantly faster than e.g. Fusion Compiler, it requires considerable amounts of compute RAM when running multiple tasks in parallel.

6. Conclusion

Synopsys RTL Architect addresses the RTL2GDS flow challenges by enabling fast PPA estimations of RTL-level modifications. We showed how we used RTL Architect in the early design stages of our EdgeVision SoC. We first presented the PPA results from RTL Architect with regards to AI accelerator configurations, memory sizes, and clock frequencies. Then we discussed the benefits and challenges of integrating RTL Architect in our design process. We expect that using RTL Architect in our design flow will lead to various benefits, including increased communication between engineers on different RTL2GDS steps, early and precise estimates for key PPA metrics, and tapeout within a shorter time.

References

[1] S. Höppner and others, "Adaptive body bias aware implementation for ultra-low-voltage designs in 22FDX technology," *IEEE TCAS II: Express Briefs,* 2019.

[2] „Synopsys RTL Architect,“ [Online]. Available: synopsys.com/implementation-and-signoff/rtl-synthesis-test/rtl-architect.html.

[3] „IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems, IEEE Std 1801-2018,“ [Online]. Available: ieeexplore.ieee.org/document/8686430.

[4] „ICGlue GitHub,“ [Online]. Available: github.com/icglue/icglue.

[5] „Synopsys IC Compiler II,“ [Online]. Available: synopsys.com/implementation-and-signoff/physical-implementation/ic-compiler.html.

[6] „Synopsys PrimePower,“ [Online]. Available: synopsys.com/implementation-and-signoff/signoff/primepower.html.