SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Development of an Online ONoC (Optical Network-on-Chip) Design Platform

Laurin Rippel

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Development of an Online ONoC (Optical Network-on-Chip) Design Platform

Entwicklung einer Online ONoC (Optical Network-on-Chip) Design-Plattform

Author:	Laurin Rippel
Supervisor:	Prof. Dr.-Ing. Ulf Schlichtmann
Advisor:	Yushen Zhang
Submission Date:	August 15, 2024

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, August 15, 2024 Laurin Rippel

In preparing this thesis, I utilized Grammarly for grammar and style correction and DeepL to enhance language quality. I used ChatGPT to generate initial drafts and expand on ideas, providing valuable suggestions and examples. Additionally, I used GitHub Copilot to generate code snippets for the developed functionality.

Abstract

Designing optical networks is a task with many manual steps and requires extensive in-depth knowledge. Compared to their electrical counterparts, the design workflow for optical networks is inefficient and fragmented. One reason is that no platform integrates different optimization tools and provides an automated design process. We address these challenges by proposing a platform tailored explicitly for optical networks, focusing on design automation and an efficient workflow. The primary goal is to streamline the design process by integrating various tools into one application, using cloud patterns and a microservice approach.

One of the platform's key features will be the ability to quickly create and test different chip layouts, providing immediate feedback and enabling rapid iteration. We can easily integrate new tools and components as they become available because the platform is designed with extensibility in mind. It includes an online editor that allows users to create designs, run tools, and visualize results, even if they only have basic knowledge about optical networks. The platform's unified environment reduces the need for manual data transfer between tools, thus minimizing the risk of error. Moreover, the cloud-based infrastructure ensures that the platform is accessible from anywhere and on any consumer hardware while running compute-intensive tools on a powerful server architecture.

With further refinement and optimization, this online design platform has the potential to become a critical asset for optical network designers. It offers a more efficient and effective way to handle the complexities of optical network design, promising better performance and higher-quality results.

Contents

1 Introduction

The rising demands for high bandwidth and low latency in Network-on-Chip (NoC) are becoming more and more challenging. In recent years, Optical Network-on-Chip (ONoC) has emerged as a promising solution to these issues.

Unlike electrical NoC, ONoC offers specialized advantages in their niche, potentially outperforming their electrical counterparts in terms of bandwidth and energy consumption [1], [2]. By taking advantage of wavelength-division multiplexing and the properties of light in silicon, ONoC meets the required high bandwidth demands while maintaining low latency and power efficiency [3]. Wavelength-Routed Optical Networks-on-Chips (WRONoCs) can further reduce latency overhead by reserving a dedicated transmission line for each pair of senders and receivers. Additionally, this design approach ensures that communication can happen without any data collisions [1].

1.1 Problem

In recent years, the design flow of WRONoCs has improved significantly. However, much progress is still required to achieve a level of design automation comparable to that of electrical chips. A prime example of this challenge is crosstalk analysis, an important characteristic of WRONoCs that we will explore in detail later. This analysis focuses on the intersections of certain components and how light behaves within them [4]. Although research exists to perform this analysis, it requires a sophisticated setup and configuration.

However, crosstalk analysis is just one of many optimization challenges. Other tools, such as *FAST+* or *PSION 2*, highlight the fact that many tools require similar but different inputs and configurations [5], [6]. These differences and various dependencies make interoperability of these tools complicated and time-consuming. In addition, some tools do not run in a standard desktop environment. Instead, they often require server hardware or clusters due to their long processing times.

1.2 Motivation

Cloud technology and a microservices architecture effectively address these issues. Therefore, we propose an application consisting of small, loosely coupled services that communicate over lightweight protocols, each operating in its own process. These microservices are designed around business functions and deployed independently. Their small sizes simplify maintenance and improve system stability by isolating the point of failure [7]. Besides, we can decide on a service-by-service basis which communication protocols and programming languages we want to choose for a WRONoC tool.

This thesis explores the development of an online optical chip design platform, utilizing the advancements in microservice architecture to address several of the introduced challenges. The platform separates tools into different services according to their requirements, such as execution time. We use synchronous communication for fast tasks and asynchronous communication for long-running tasks. One notable feature of this platform is its *no code* approach, which allows users to configure and simulate optical networks through an easy-to-use drag-and-drop chip designer without the need for prior programming knowledge or extensive training. This feature set enhances usability and supports independent scaling and dynamic adjustment based on system load, improving the optical chip design process.

1.3 Outline

The following chapter begins with a short background on ONoC. In chapter 3, we will analyze the requirements of our platform, discuss scenarios and features, and plan the implementation by using different models to ensure the architecture is correct. Chapter 4 will cover some implementation details and explain design choices we made. In chapter 5, we will reflect on the development, compare the system's status with the requirements and show examples of the future potential of our application.

2 Background

This section introduces fundamental concepts of ONoCs that are essential to understanding the components we will create in our application. We then look at the physical structure of an Integrated Circuit (IC), followed by the core elements used in ONoC. Next, we will present a selection of design automation tools relevant to this area. Finally, we look at state-of-the-art alternatives for design automation tools currently available for ONoC.

2.1 Fundamentals

An ONoC is a specialized type of NoC that uses specific wavelengths of light to transmit data between the components of a System-on-Chip (SoC). All components are called nodes, whether they send, receive, or do both. The data transmitted between a sender and receiver pair is called a signal [8].

The key components of ONoCs are silicon waveguides for signal transmission and optical switching blocks for signal routing [9]. They function by routing signals between a set of input and output ports [10]. Based on the routing mechanism, ONoCs are divided into two categories: 1) *active-networks*, which change the routing-path mechanism in real-time, and 2) *passive-networks*, which automatically route the optical signals based on their wavelength [8], [9]. The latter are therefore known as WRONoCs, which are the primary focus of this thesis.

For each pair of communicating nodes in WRONoCs, a dedicated signal path is reserved at design time. This allows the sending node to transmit data to its destination node at any time, regardless of other ongoing communication [11]. In this way, WRONoCs avoid the network latency and reduce the energy overhead of path setup, providing highly predictable communication [6], [11].

However, these benefits come with certain drawbacks: WRONoCs trade performance and predictability for scalability limitations. As the number of I/O connections grows beyond 16, passive networks are less power efficient than their active counterparts [6], [12]. Furthermore, current WRONoCs struggles to provide high bandwidth for all connections between many nodes. One approach to address these issues is to aggregate processing cores into clusters of 8 to 16 node WRONoCs [6].

2.2 Physical Architecture

The physical structure of an ONoC consists of multiple layers. Figure 2.1 shows that the bottom layers are electrical layers, where transistors are etched into a silicon substrate as usual [1], [8]. On top of this, we place optical layers to allow communication between different components on the SoC. The electrical and photonic layers are then connected by through-silicon vias, which provide enough space for isolation [1]. A heat spreader, heat sink, and thermal interface complete the design of the IC. Since our application focuses mainly on the design and optimization process of the routing and power layers, we will not explain the other layers in more detail.

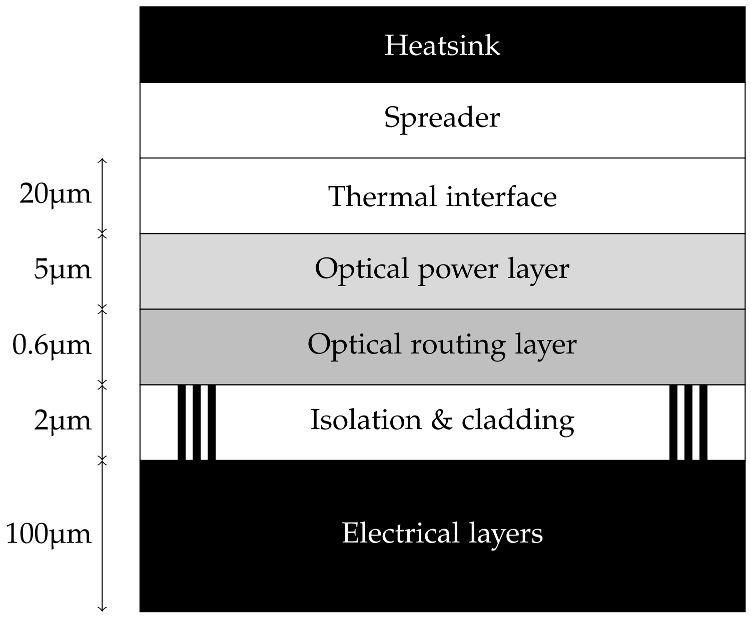


Figure 2.1: An exemplary 3D stack of an Integrated Circuit with an ONoC (adapted from [8]).

Routing Layer: This layer contains all the elements necessary to route signals from the sender to the corresponding receiver nodes and is essential for all ONoCs. The elements of this layer are optical routing devices (see section 2.3) and waveguides, which are the equivalent of wires in the electrical world. Depending on the power source, optical power distribution is done on this layer as well [8].

Power Layer: The optical power for an ONoC is provided by one or more laser sources. These can be placed *on-chip* or *off-chip*. In the first case, we place the laser source on the power layer above the routing layer. Then, the optical power is routed from the sources to each sending node below. In the second case, a power receiver must be placed on the routing layer to accept the power from outside the chip [8], [13].

2.3 Network Elements

Now that we have seen the general structure of an ONoC, we will explore the various elements commonly used in the construction of the optical routing and power layers. Each element will be discussed in detail to better understand its role and functionality within the optical network.

2.3.1 Laser Sources

A laser source is an element in the ONoC that emits a certain number of wavelengths or a range of wavelengths [14]. As mentioned before, there are two fundamentally different ways to integrate this component on a chip:

On-Chip: Here, the laser source or sources are placed close to the modulators of the node they supply, adding complexity to the manufacturing process [8]. In addition, challenges, such as heat dissipation, are a new concern [3]. However, no optical coupler is required, thus the power distribution between each source and its respective modulators is straightforward [2], [8], [13].

Off-Chip: A laser source outside the circuit provides wavelengths ($\lambda 1..\lambda n$) that are guided into the chip through an optical fiber and coupled into a waveguide [3]. From there, the Power Delivery Network (PDN) splits and delivers power to all the modulators (see section 2.3.3) [8]. In contrast to the on-chip variant, the router is more complex to design in the off-chip case. On the one hand, the additional PDN makes the design more complex, but on the other hand, the fact that the laser source is located outside the IC reduces the complexity again [8], [13].

In addition, there are two other categories of laser sources. These are either *Type-X* or *Type-Y* and are independent of the on-chip/off-chip configuration [8]. Although knowing about them is helpful, we do not need to talk about them in more detail.

2.3.2 Waveguides

Waveguides are the optical equivalent of electrical wires used to transmit a range of frequencies of light. They enable the simultaneous transmission of multiple frequencies in both directions [8]. Wavelength-division multiplexing allows a single waveguide to accommodate multiple wavelengths, achieving energy-efficient on-chip communication with high bandwidth and low latency [1]. An advantage of waveguides is their ability to cross each other in the same plane. These crossings increase optical loss and noise but do not change the light's path. Waveguide terminators must be placed at the end of waveguides to stop the propagation of light by absorbing as much light as possible to reduce reflections [8].

2.3.3 Power Delivery Network and Splitters

When the power source is positioned off-chip, the external power must be transferred to all modulators by strategically placing splitters and waveguides on the optical routing layer [15]. This network is called a Power Delivery Network (PDN) and typically follows a binary tree structure as shown in Figure 2.2. The optical coupler is the root node, followed by layers of laser splitters as intermediate nodes. The modulators are the leaf nodes and each waveguide represents an edge of the graph. This structure ensures that each modulator receives the minimum power required [8], [15]. The splitters have an internal ratio, which can be configured at design time. They split the incoming power into two outputs according to the configured ratio [15].

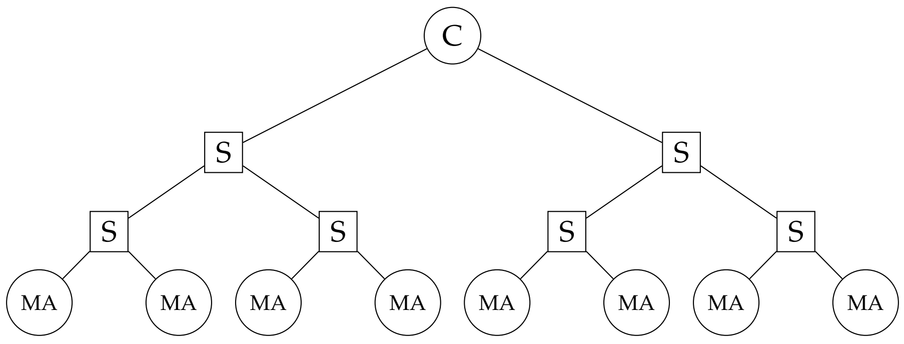


Figure 2.2: Visualization of the PDN structure as a binary tree with depth three, where the root node represents the coupler (C), the intermediate vertices represent the splitters (S), and the leaf nodes represent the Modulator Arrays (MA).

2.3.4 Modulators and Demodulators

Modulators are devices that receive optical power as input and output an optical signal that carries data. To achieve this, the input power is modulated to encode the electrical signal onto the optical signal output. A modulator is constructed using a Micro-Ring Resonator (MRR)[1] adjacent to a waveguide using On-Off Keying (OOK) as the modulation scheme. The electrical signal is coupled to the MRR by shifting its resonance wavelengths [8]. Additional MRRs are placed in sequence to modulate more wavelengths. Such a sequence is called a Modulator Array. We require a sender node to have at least one Modulator Array [4], [8].

Demodulators are built similarly to modulators using one or more MRRs along a waveguide, but their functionality is reversed. Demodulators convert a modulated optical signal back to its electrical signal by decoding the input with a photodetector [8]. Like modulators, demodulators can be placed in series to create a sequence of decoders, each configured to decode a different set of resonance wavelengths. Similar to Modulator Array, we call this sequence a Demodulator Array.

To simplify the elements in our chip designer, we create a modulator or demodulator using its array variant by placing only one MRR in the array. This approach reduces the total number of components in our designer by two.

2.3.5 Optical Router

The optical router is the core of an ONoC, responsible for routing optical signals through the chip on their dedicated path. At the lowest level, the router consists of waveguides, waveguide terminators, and MRRs [8]. However, we place the MRRs in higher-level components such as Parallel Switching Elements (PSEs), Crossing Switching Elements (CSEs), or Generic Routing Units (GRUs).

Micro-Ring Resonator

To route messages, or more precisely, to direct light within the waveguides of the optical router, we rely on MRRs. A MRR is a silicon ring structure whose radius can be tuned to route specific wavelengths [8].

A signal traveling on a waveguide with a MRR near another waveguide is split between the two. The amount of energy deflected onto the second waveguide depends mostly on the resonance wavelengths of the MRR and the wavelengths of the signal traveling through the waveguide [8], [16]. As shown in Figure 2.3, the effect of the MRRs works equally well in parallel and perpendicular waveguide structures.

[1]see section 2.3.5

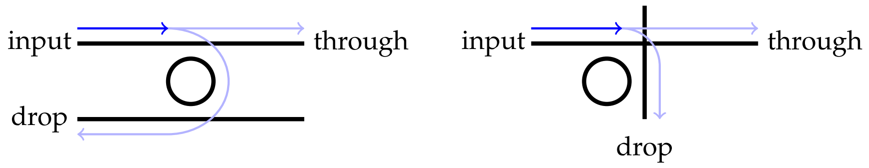


Figure 2.3: Routing behaviour of MRRs. A fraction of the energy in the input waveguide continues its path; the other fraction is deferred to the other waveguide adjacent to the MRR (adapted from [8]).

The resonance wavelengths Λ^{res} of a MRR with radius r can be approximated by

$$\Lambda^{res} = \left\{ \mu \frac{2\pi r}{i} \mid i \in \mathbb{Z}^{+} \right\} \tag{2.1}$$

where μ is a physical parameter [8]. Since not all radii and corresponding wavelengths are available in an ONoC, our design application will constrain the radius of a MRR between $r \in [5\mu m; 30\mu m]$ [8], [11]. Furthermore, this equation shows us that the result is a distinct set of resonance wavelengths rather than a single resonance wavelength.

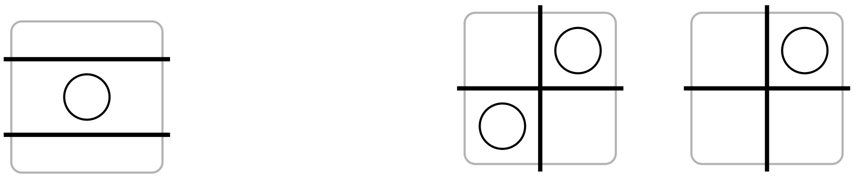
Parallel Switching Element and Crossing Switching Element

To simplify the design and simulation process, the optical router of a WRONoC uses templated components. As shown in Figure 2.4, there are two structurally different components, the Parallel Switching Element and the Crossing Switching Element. These components have predefined MRR positions and waveguide alignments to route signals based on their resonance wavelengths [8]. The PSE (Figure 2.4a) has one MRR between two parallel waveguides, while the CSE (Figure 2.4b) has one or two MRRs near a waveguide crossing. Their routing behavior can be derived from the general routing behavior of MRRs. Using these templated components, we can accurately model the on-resonance and off-resonance behavior of signals in the network, ensuring efficient and reliable message routing.

Generic Routing Unit

The CSE and PSE components are a good basis for designing optical routers. However, there are several shortcomings with the CSE component as A. C. Truppel [8] explains:

- The number of MRRs is limited to one or two, while the CSE can theoretically have up to four.

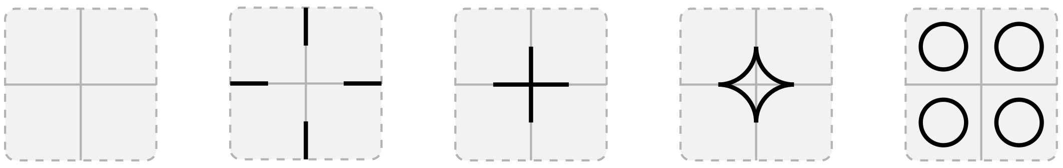


(a) Parallel Switching Element **(b)** Crossing Switching Element

Figure 2.4: a) PSE and b) 2x2 and 1x2 CSE.

- All MRRs in a 2x2 CSE are configured with the same radius.
- The waveguides are fixed to the cross-like structure, although other structures are possible and sometimes even more desirable.

These are the reasons why A. C. Truppel [8] proposed the Generic Routing Unit (GRU) to solve some of these inflexibilities. The GRU has four bidirectional ports and is initially an empty structure which can be configured with different building blocks to achieve the desired routing behavior.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 2.5: Overview of GRU structures: (a) Empty GRU, (b) GRU edges, (c) GRU crossings, (d) GRU bends, (e) GRU MRRs.

To configure a GRU, we follow several rules to ensure the design is valid (see [8] for details): First, we start with an empty building block for the GRU, as shown in Figure 2.5a. Next, an edge 2.5b can be placed in each direction (left, right, top, bottom) to allow incoming or outgoing connections. The third step is to choose the routing elements. We can use a combination of crossing waveguides 2.5c, bending waveguides 2.5d, and MRRs 2.5e according to the following rules:

- The crossing waveguides are mutually exclusive with the bending waveguides.

- For each corner, the bend and the MRR are mutually exclusive.
- Two corner bends leading to the same edge are mutually exclusive.
- Any MRR can only be placed if both adjacent sides have a waveguide (in the form of an edge, crossing, or corner bend).
- In addition, a waveguide terminator must terminate any unconnected waveguide.

2.4 Design Automation Tools

As mentioned in the Introduction, various design automation tools are currently available to assist users in their design process. A few of these tools are selected to be integrated into the platform we propose. Unfortunately, we cannot immediately integrate the entire set of tools due to time constraints within the scope of this thesis. However, we would like to introduce a broader list of tools to allow a better estimation of future capabilities.

- **Topology Generation:** These tools automatically generate common topologies such as the Crossbar Router [1], based on the number of nodes on an IC [9]. Examples are shown in section 4.5.1.
- **Topology Optimization:** These tools take the Communication Matrix (CM)[2], and a prebuilt topology as input, and output an optimized design. Notable examples are the FAST+ [5] or CustomTopo [9] optimization tools.
- **Placement and Routing:** These tools need a schematic design of a chip as input and use that blueprint to generate a fitting physical layout of the network elements. Examples include tools like ToPro, Proton+, and PlanarONoC [1], [13], [17], [18].

In addition, some tools do not fit into any of the above categories but solve a more specific problem related to the design of an ONoC or WRONoC.

- **PSION2:** This tool takes a network of GRUs and tries to configure the internal state of the GRU (see section 2.3.5) to optimize for a given set of parameters [6].
- **X-Talk:** The X-Talk tool takes a chip's CM and its schematic or layout design, producing the crosstalk analysis for each element. Useful to verify and optimize the chip's design [4].

[2]A Communication Matrix contains sender and receiver pairs, more details in section 4.3.1.

- **Bandwidth Optimization:** This tool focuses on refining the schematic design of an ONoC by further configuring the radius size for each MRR [11].

For this thesis, we will first focus on Topology Generation and then on the integration of the X-Talk analysis to demonstrate the feasibility and advantages of the proposed system.

2.5 Current State of Design Automation

Industry-standard design automation tools have been developed for various areas of optical computing. Synopsys[3] offers a software suite for multiple levels of abstraction in Photonic Integrated Circuit (PIC) design, such as *OptSim*.

This tool addresses the design and simulation of optical communication systems at the signal propagation level. Using advanced simulation techniques and a graphical user interface, OptSim provides high accuracy and ease of use for its customers [19].

Despite OptSim's capabilities, the tool is primarily focused on PIC design rather than ONoC or WRONoC. Research-specific analysis like X-Talk is not available in their application suite. Its performance also depends on the user's computer, as it is a client-side application. While it runs well on powerful machines, it does not offer the advantages of a web-based approach, such as easy accessibility and the ability to run tools on server hardware. In addition, the licensing model is often unsuitable for researchers.

The field of WRONoCs is a highly specialized subcategory of optical networks. With rapid advances in topological and physical design, this niche's need for design automation tools is growing further. However, challenges remain in optimizing performance factors and closing gaps in the design workflow [1].

Our approach addresses these gaps by developing tools tailored explicitly for WRONoC design, focusing on efficiently solving closley related optimization problems and providing a more accessible and specialized solution.

[3]https://www.synopsys.com/

3 Requirements Analysis

The previous chapter introduced the theoretical background of ONoC, giving us valuable details on what our Online ONoC Design Platform aims to model. This chapter will now build on that information by defining the requirements that will guide the implementation process and set the implementation goals.

We will first look at selected design automation applications to gain insight into the key components of these systems. Next, we will use this knowledge to define the broader vision for our proposed system and then go into detail by listing functional and non-functional requirements. Finally, we will talk about different system models that capture how we imagine users interacting with our application.

3.1 Related Systems

Cloud Columba[1]

Cloud Columba is a cloud-based application built to streamline the design of microfluidics. It aims to reduce manual steps in the design workflow, minimize errors, and lower the barrier of entry in microfluidics. It features a schematic design mode, shown in Figure 3.1, where elements can be placed on a canvas, connected with channels, and used for further simulations. Special Design Rule Checks (DRCs) are available to prevent errors during the design step. Cloud Columba's feature set, particularly its schematic design mode and DRCs, shows us great examples of components we can incorporate into our application.

Flui3d[2]

Flui3d is an interactive software platform designed specifically for developing 3D-printed microfluidic devices. It addresses the complexity of designing these devices by providing a parameterized component library and support for multi-layer designs. One of its key features is the Design-for-Manufacturing capability, which ensures that microfluidic devices can be seamlessly fabricated using consumer-grade 3D printers.

[1]https://cc2.cloud-columba.org/
[2]http://flui3d.org/

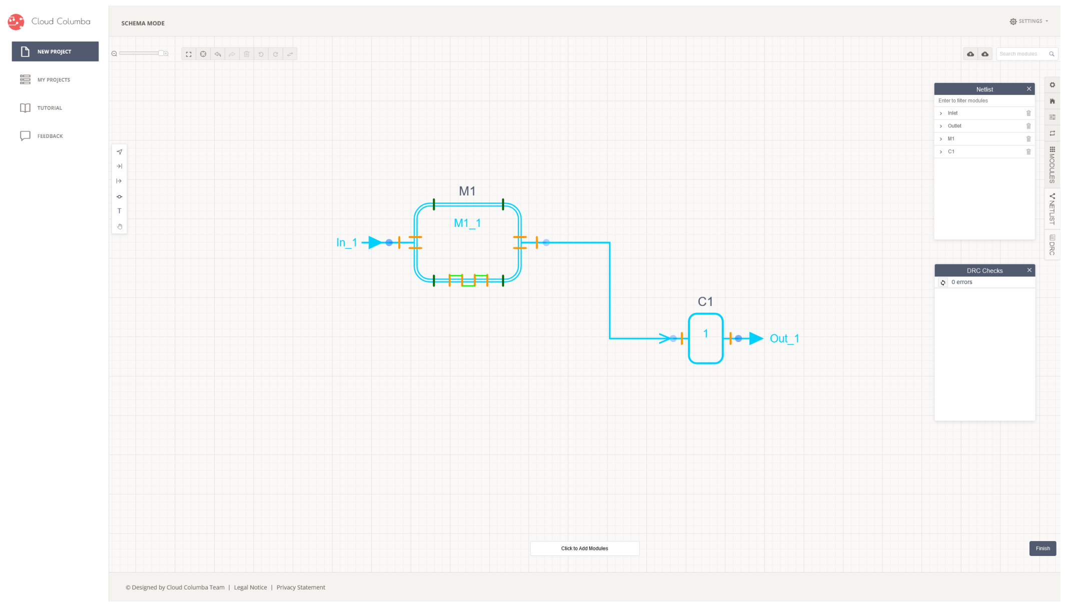


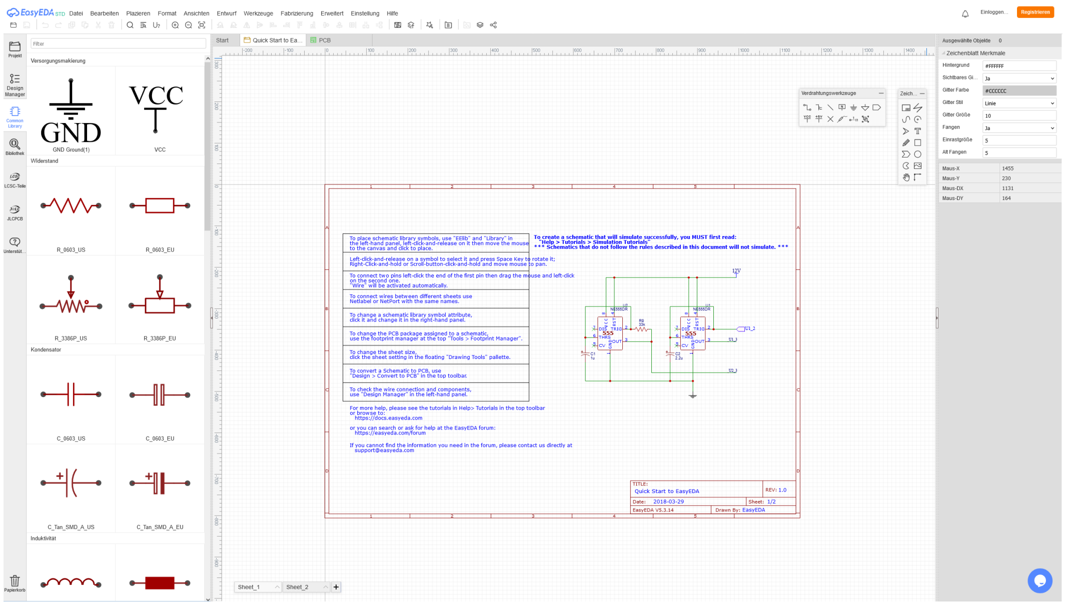
Figure 3.1: Schematic Editor in Cloud Columba.

This system does not require specialized fluid mechanics or 3D modeling knowledge, making it accessible to a broader range of users. Flui3d opens up new possibilities in biology, chemistry, and medicine by simplifying the design process and enabling the creation of complex microfluidic devices [20]. Flui3d's ease of use and accessibility is particularly inspiring. Our system will build on this approach by offering similar ease of use but instead focusing on the specific requirements of ONoCs design.

EasyEDA[3]

EasyEDA, shown in Figure 3.2, is a web-based tool for creating and simulating electrical circuits. It offers a comprehensive set of features for schematic design, Printed Circuit Board (PCB) layout, and circuit simulation. The intuitive user interface makes it accessible to both beginners and experienced engineers. EasyEDA supports various components and libraries, enabling the efficient creation of complex designs. Its editor guides users through the entire design process, from concept to production, making it an excellent reference for design automation software.

[3]https://easyeda.com/



To place schematic library symbols, use "EElib" and "Library" in the left-hand panel, left-click-and-release on it then move the mouse to the canvas and click to place.

Left-click-and-release on a symbol to select it and press Space Key to rotate it; Right-Click-and-hold or Scroll-button-click-and-hold and move mouse to pan.

To connect two pins left-click the end of the first pin then drag the mouse and left-click on the second one.
"Wire" will be activated automatically.

To connect wires between different sheets use Netlabel or NetPort with the same names.

To change a schematic library symbol attribute, click it and change it in the right-hand panel.

To change the PCB package assigned to a schematic, use the footprint manager at the top "Tools > Footprint Manager".

To change the sheet size, click the sheet setting in the floating "Drawing Tools" pallette.

To convert a Schematic to PCB, use "Design > Convert to PCB" in the top toolbar.

To check the wire connection and components, use "Design Manager" in the left-hand panel.

For more help, please see the tutorials in Help> Tutorials in the top toolbar or browse to:
https://docs.easyeda.com

or you can search or ask for help at the EasyEDA forum:
https://easyeda.com/forum

If you cannot find the information you need in the forum, please contact us directly at
support@easyeda.com

**To create a schematic that will simulate successfully, you MUST first read:
"Help > Tutorials > Simulation Tutorials"
*** Schematics that do not follow the rules described in this document will not simulate. *****

TITLE: Quick Start to EasyEDA
REV: 1.0
Date: 2018-03-29
Sheet: 1/2
EasyEDA V5.3.14
Drawn By: EasyEDA

Figure 3.2: Editor User Interface (UI) in EasyEDA standard edition.

3.2 Proposed System

With the inspiration of the related systems above and the technical specifications in chapter 2, we propose a system that allows users to quickly design and configure an ONoC. Our platform should offer a user-friendly interface with draggable components and a canvas for placing and connecting elements. A dedicated side menu will allow further customization of these elements.

Our system will feature two distinct canvas panels: one for schematic design and one for physical layout. The schematic design panel focuses on the connections and signal flow, allowing users to visualize and optimize the logical design of their chip. Likewise, the layout panel allows users to manage the physical placement and routing of waveguides, which is critical for actual chip fabrication. This dual-canvas approach is analogous to PCB design in the electrical engineering domain, as seen in EasyEDA. It allows users to simultaneously optimize logical connections in the schematic view and manage physical placements in the layout view, improving overall design efficiency and accuracy.

We include operations for project management, such as creating, editing, renaming, deleting, cloning, exporting, and importing projects. This feature set ensures efficient organization and accessibility of all projects by our users. Furthermore, as mentioned in section 2.4, our system will integrate design automation tools and provide an interface for running simulations, configuring parameters, and displaying results. Our goal is to create an efficient workflow from design to simulation and, in the future, to fabrication.

By incorporating these features, we aim to simplify the ONoC design process, making it accessible to users without specialized knowledge while providing powerful tools for experts. This combination of user-friendly design and robust functionality sets our system apart from existing solutions and offers a unique approach to ONoC design.

3.3 Functional Requirements

This section details the functional requirements of our proposed system, focusing on specific user expectations. We group the requirements into three categories:

Frontend: Chip Designer and Tool Interaction

FR1.1 **Project Management:** Users can create, import, export, save, delete, rename, and clone projects.

FR1.2 **Chip Design:** Users can place, drag, connect, delete, and configure elements.

FR1.3 **Panel Sync:** Users can work with the same chip in both schematic and layout panels, with automatic updates on both panels.

FR1.4 **Settings Panel:** Users can view and configure project settings in a dedicated panel.

FR1.5 **Tool Integration:** Users can run tools via an interface on the current design with minimal configuration, displaying results as overlays on the design.

Backend: User Authentication and Project Storage

FR2.1 **User Registration:** Users can create an account with a username, email, and password.

FR2.2 **User Login:** The system authenticates users based on their credentials and grants access.

FR2.3 **Project Storage:** Stores all project data (metadata, canvas configuration, simulation results, ...) and links them to a user.

Wrapper Service: Encapsulation of ONoC Tools

FR3.1 **Tool Interface:** Provide an interface to export data to run ONoC related tools.

FR3.2 **Sync Tool Exec:** Support synchronous tool execution with real-time results.

FR3.3 **Async Tool Exec:** Support asynchronous tool execution with later result retrieval.

3.4 Non Functional Requirements

The next analysis follows the FURPS+ model [21] and lists all nonfunctional requirements. However, we will not cover the functional category, as it has already been addressed in the previous section.

Usability

NFRU.1 **Quick Add:** Users should be able to add new elements in three clicks or less.

NFRU.2 **Consistent Style:** Designed elements should be consistent, clearly distinguishing waveguides, outlines, and connection points.

NFRU.3 **Easy Navigation:** Users should be able to find any UI element within 30 seconds.

Reliability

NFRR.1 **System Stability:** The designer should still be accessible even if one or more Tool Services are currently unavailable.

Performance

NFRP.1 **Component Limit:** The system should handle approximately 1000 concurrent elements.

NFRP.2 **Node Limit:** The system should remain stable with designs up to 16 nodes.

NFRP.3 **Tool Speed:** Synchronous tools should execute in 10 seconds or less.

Supportability

NFRS.1 **Browser Support:** The system should support Chrome and Firefox, with hotkeys mapped for Windows and macOS.

3.5 System Models

The following sections visualize different perspectives on the system. User scenarios provide insight into how we imagine the interaction between actors and the proposed system, followed by an abstract view in the form of a communication diagram. Lastly, we explain parts of our UI and the underlying design decision.

3.5.1 Scenarios

Realistic Scenario: Collaborative Optical Chip Design Enhancement

Actors: Emily, a researcher specialized in ONoC.

Entry Condition: A colleague has shared a new optical chip design with Emily for review and enhancement.

Emily logs into the ONoC-Designer through her web browser and is greeted by the platform's dashboard, which displays options for viewing previous projects or creating new ones. Opting to import a project, Emily uploads a file shared by her colleague. She confirms the project's settings, names it, and then clicks *Create Project*. As the project loads, an animation shows her design environment is being prepared.

The interface presents Emily with three main panels: a schematic panel showing the connections within the chip, a layout panel displaying the physical placement of components in micrometers, and a settings panel for global configurations like wavelengths and type of laser source.

Emily navigates to the tools menu and selects the *X-Talk* tool, which prompts her to configure additional parameters. The analysis begins after she submits her configuration with the *Run Tool* button. The results are immediately visualized on the platform's canvas, highlighting potential areas of signal interference.

She quickly reviews the analysis before adjusting the connections of the components. These changes are synchronously updated between the schematic and layout panels, maintaining consistency across the design. Emily then re-runs the crosstalk analysis to validate the effectiveness of her changes.

The revised analysis confirms a reduction in signal interference. Emily then saves and downloads the updated project as a JSON file containing the modified ONoC design. This file is ready to be shared with her colleague for further review. By using the ONoC-Designer, Emily has efficiently improved her design process and the overall design quality of the ONoC.

Visionary Scenario: Using a Predefined Topology and Adjusting the Layout with Design Automation

Actors: Emily, a researcher specializing in ONoC.

Entry Condition: Emily has little time and wants to set up a common chip design using a predefined topology, but it should be optimized for her needs.

Emily logs into the ONoC-Designer from her web browser and navigates to the dashboard. She chooses to start a new project, enters the project settings, names the project, and creates it by clicking *Create Project*. As the project loads, an animation shows her design environment is being prepared.

She accesses the tools menu and selects the *Crossbar* topology, further specifying the number of nodes and the spacing between elements. The system generates the initial schematic design based on her input and displays it in the schematic panel. Emily then reviews the automatically placed components and connections.

Emily uses the platform's design automation tools to optimize the layout. She selects the *Auto Layout* tool from the tools menu, which adjusts the component placement to minimize waveguide spacing. The layout updates in real time to show the automated adjustments.

Satisfied with the automated layout, Emily makes a few manual adjustments to fine-tune the design. She then uses the *X-Talk* tool, which analyzes the design for potential signal interference. The tool highlights areas for improvement, and Emily manually makes the necessary adjustments.

Once the design is optimized and verified, Emily saves the project and exports the design files for fabrication. The ONoC-Designer's predefined topology and design automation features have accelerated the design process, allowing Emily to efficiently create and optimize complex optical chip layouts.

3.5.2 Sequence Diagrams

This section will analyze the communication flow of events within our system. We will focus on two examples: 1) opening the ONoC designer and 2) executing design tools. These diagrams illustrate the interactions between the user, the web front-end, the wrapper service, the authentication service, and the database to clearly understand the system's behavior during these essential operations.

The first scenario, shown in Figure 3.3, details the flow of events when a user opens the ONoC designer, logs in, views their projects, and opens a specific project. This flow is a typical action we expect users to take when opening our application. We focus on secure access and efficient retrieval of project data in this sequence, allowing users to begin their actual design workflow seamlessly.

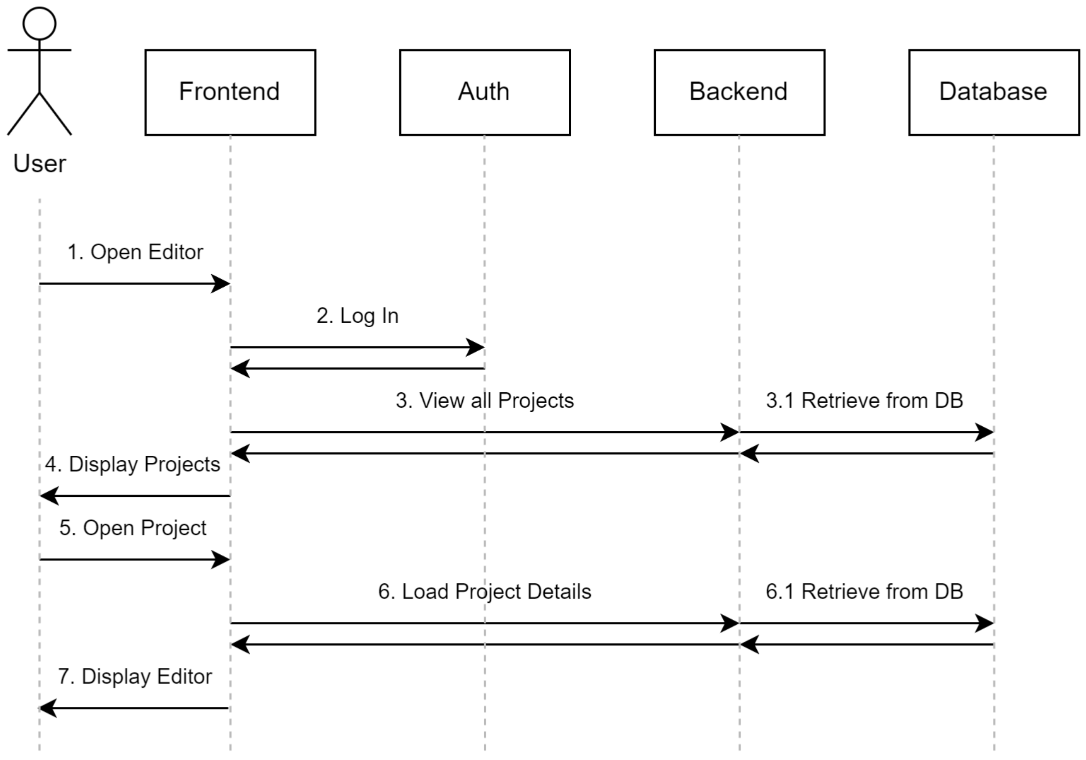


Figure 3.3: Sequence Diagram for opening the ONoC Designer.

Figure 3.4 illustrates the second scenario, describing the sequence of events when a user executes a design tool within the ONoC designer. This interaction is essential for performing simulations and analysis on the design to provide meaningful results.

This analysis highlights the critical role of authentication in our application, which is an essential step in the entire system. It also underscores the importance of our back-end service and database, as this, in combination with auth, is a key step before we can enter the design phase. This deep understanding of these interaction sequences helps us optimize the system, ensuring it meets user needs and remains reliable.

3.5.3 User Interface

As a final section of the requirement analysis, we want to define some UI components of the proposed system. Since the system was already implemented at the time of writing, we presented the actual interface rather than our sketches and mock-ups.

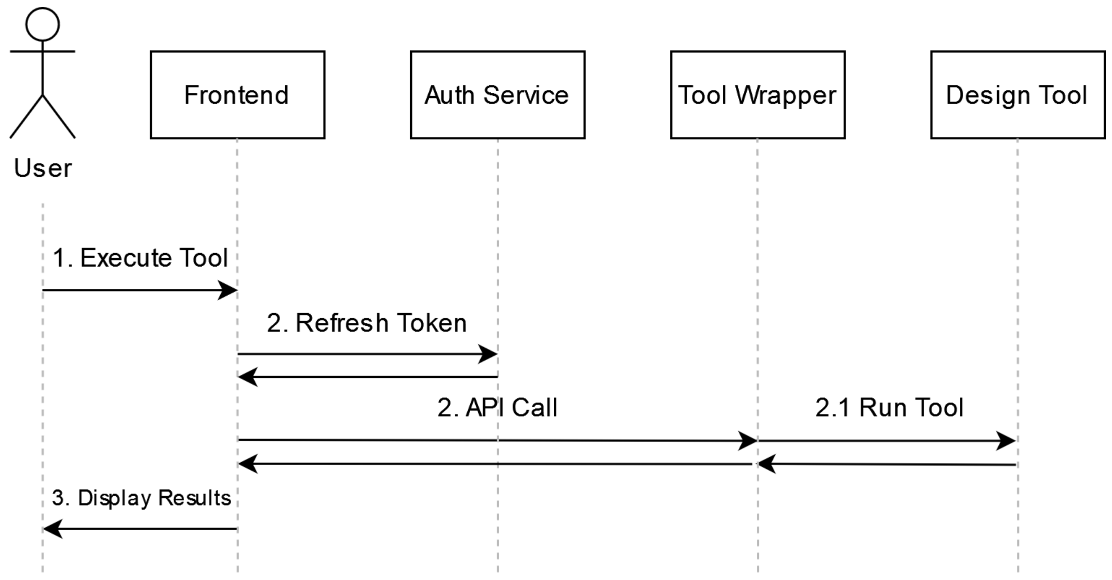


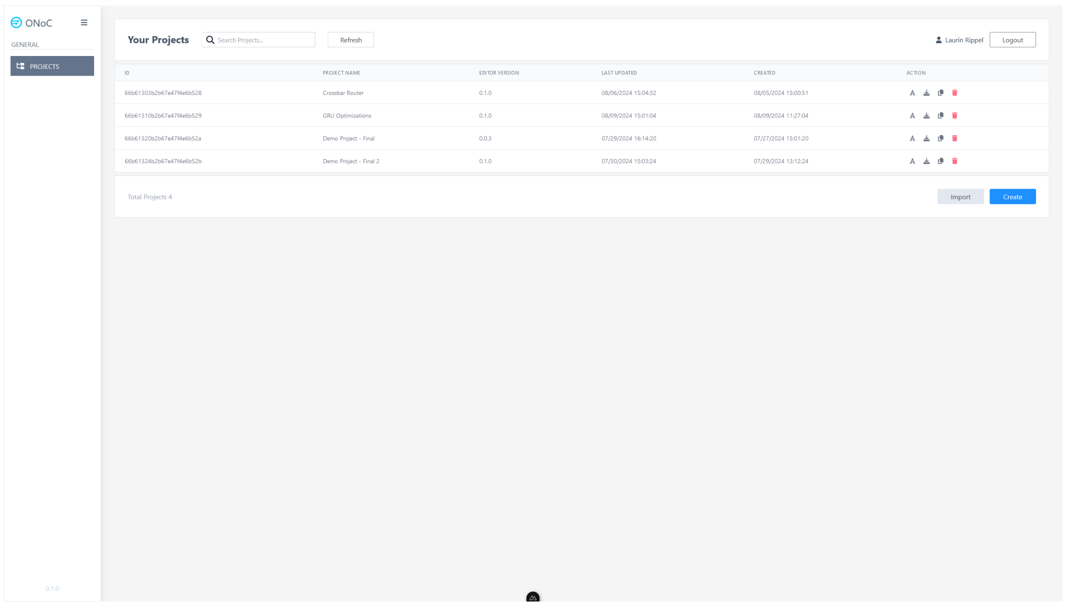
Figure 3.4: Sequence Diagram for Design Tool Execution.

Project Overview

Figure 3.5 illustrates the project overview, which is the landing page when a user opens our editor. The main content of this page is a searchable table that lists all projects associated with the user. Users can quickly open a project by double-clicking on a row. In addition, each row has action buttons for renaming, downloading, cloning, and deleting a project. This layout helps users quickly find and manage their projects, increasing productivity. Moreover, user authentication options are available in the top right corner, and the sidebar can be collapsed to free up screen space. This extra space is beneficial when working in the canvas editor, which we will look at next.

Schematic and Layout Canvas

When creating a project, the user's initial view is the canvas shown in Figure 3.6. This canvas has two menu bars on the left: the top one for opening a list of all available components and tools, and the bottom one for familiar editor functions such as fit to window, center, zoom, reset zoom and snap to grid. There is also a help button that lists all hotkeys. On the right is a configuration panel for the currently selected element (highlighted in light blue). In the center of the canvas, we see a sample design consisting of a laser source, a modulator array, two GRUs, and three demodulator arrays.



ID	PROJECT NAME	EDITOR VERSION	LAST UPDATED	CREATED	ACTION
66b61303b2b67e47f4e6b528	Crossbar Router	0.1.0	08/06/2024 15:04:32	08/05/2024 15:00:51	
66b61310b2b67e47f4e6b529	GRU Optimizations	0.1.0	08/09/2024 15:01:04	08/09/2024 11:27:04	
66b61320b2b67e47f4e6b52a	Demo Project - Final	0.0.3	07/29/2024 16:14:20	07/27/2024 15:01:20	
66b61324b2b67e47f4e6b52b	Demo Project - Final 2	0.1.0	07/30/2024 15:03:24	07/29/2024 13:12:24	

Total Projects 4

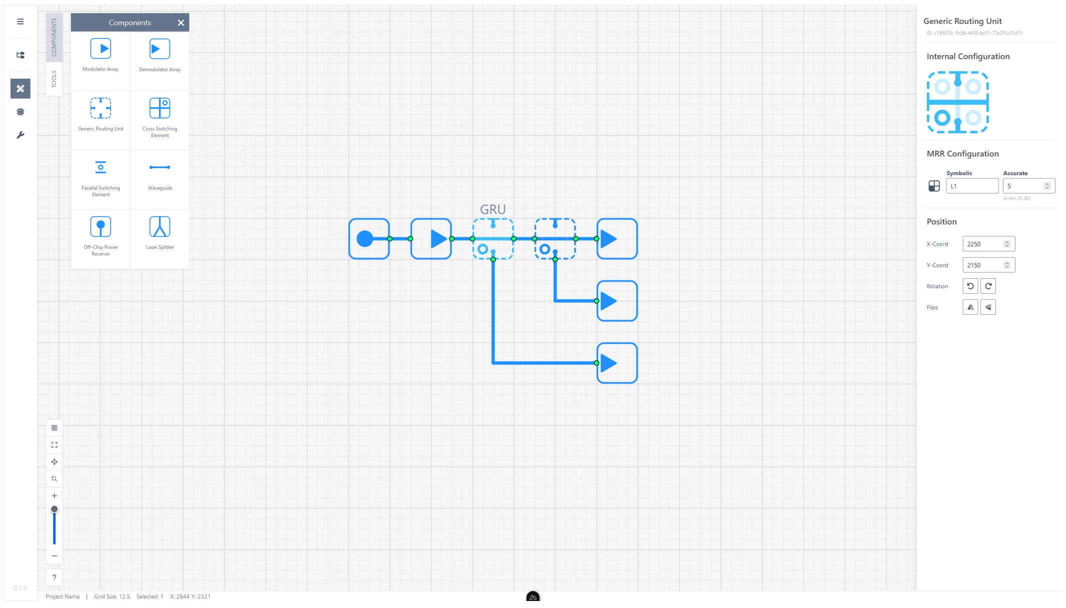
Figure 3.5: Example of the projects view.

The actual placement and routing of these elements may differ from their schematic positions, where the layout canvas comes into play. Figure 3.7 shows how the elements from the schematic canvas can be routed differently on the layout canvas. The outlined box represents the physical dimensions of the chip, with elements automatically synchronizing between schematic and layout view. This real-time synchronization ensures that logical and physical designs remain consistent, providing immediate visual feedback and error detection while keeping each element's individual position and rotation attributes so they can be placed according to the user's needs.

Settings Panel

Figure 3.8 shows how users can modify project settings. These settings are grouped into expandable categories for ease of navigation. All configuration parameters (see section 4.3.1) can be adjusted here.

The CM menu is one of these configuration details and is shown in Figure 3.9. This menu allows the user to specify whether there is a connection (a value other than 0) and the exact bandwidth value for each pair (i, j). The user can also switch from matrix input to graph input, providing an alternative input mode that is especially useful as the number of nodes grows.



Generic Routing Unit
ID: c169623c-9c84-4458-bc01-77e201c01d15

Internal Configuration

MRR Configuration

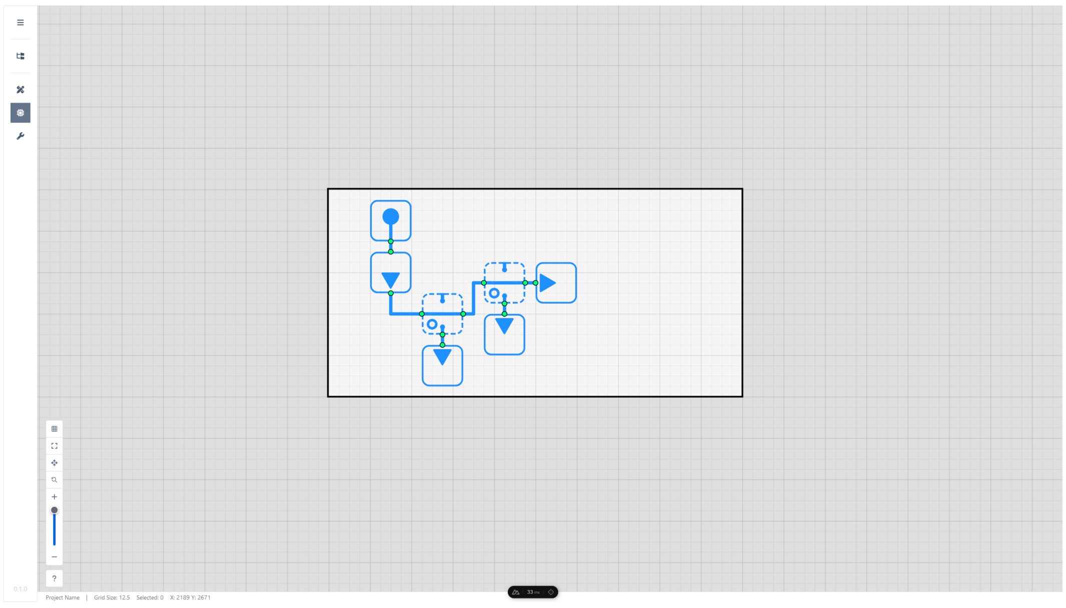
Symbolic: L1
Accurate: 5
in nm (5-30)

Position

X-Coord: 2250
Y-Coord: 2150
Rotation
Flips

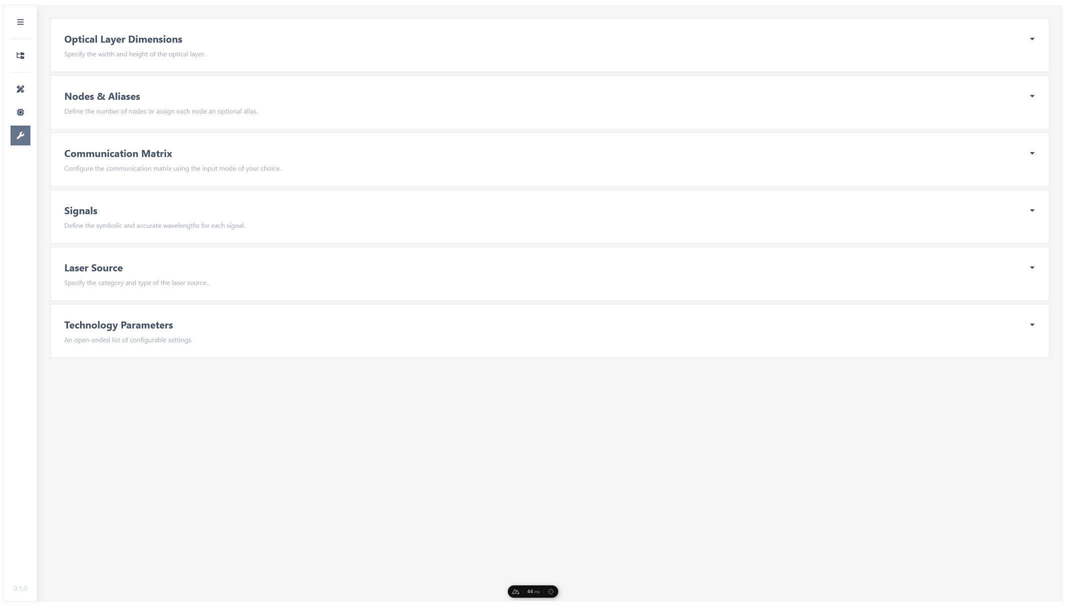
Project Name | Grid Size: 12.5 Selected: 1 X: 2844 Y: 2321

Figure 3.6: Example of a design schematic.



Project Name | Grid Size: 12.5 Selected: 0 X: 2189 Y: 2671

Figure 3.7: Example of a design layout.



Optical Layer Dimensions
Specify the width and height of the optical layer.

Nodes & Aliases
Define the number of nodes or assign each node an optional alias.

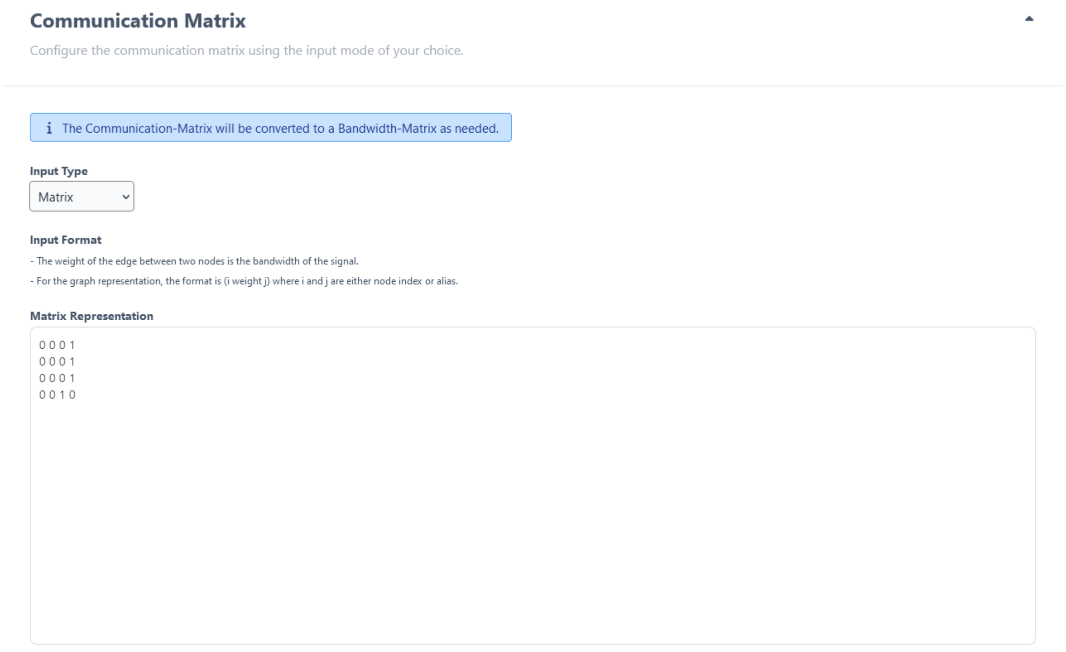
Communication Matrix
Configure the communication matrix using the input mode of your choice.

Signals
Define the symbolic and accurate wavelengths for each signal.

Laser Source
Specify the category and type of the laser source..

Technology Parameters
An open-ended list of configurable settings.

Figure 3.8: Example of the settings panel.



Communication Matrix
Configure the communication matrix using the input mode of your choice.

The Communication-Matrix will be converted to a Bandwidth-Matrix as needed.

Input Type
Matrix

Input Format
- The weight of the edge between two nodes is the bandwidth of the signal.
- For the graph representation, the format is (i weight j) where i and j are either node index or alias.

Matrix Representation
```
0 0 0 1
0 0 0 1
0 0 0 1
0 0 1 0
```

Figure 3.9: Example of the CM setting.

4 System Design

In this chapter, we will present an architectural overview and show ways to meet the requirements introduced in chapter 3. Our goal is to create an application capable of designing and optimizing the routing layer of ONoCs, as introduced in chapter 2.

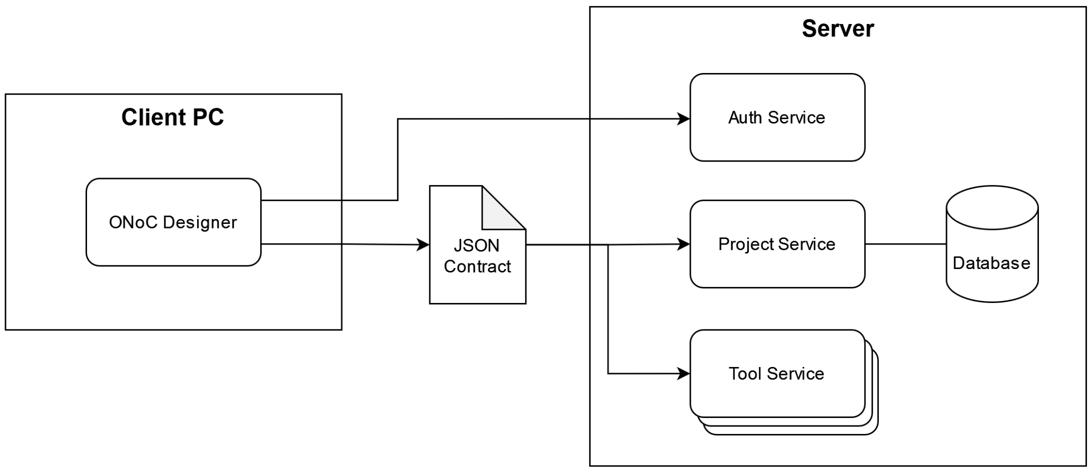


Figure 4.1: Sketch of the system architecture and relationship of the components.

Figure 4.1 illustrates the proposed system architecture from an early design stage. Using a client-server model, we separate the user interface from the back-end services. The main component is the *ONoC Designer*, which allows users to plan, route, and configure a chip design in their browser. The server provides various services, such as an Auth Service for authorization, a Database for managing project storage, and a Project Service that exposes data through an Application Programming Interface (API). It also includes various Tool Services that enable automatic simulation runs and improve the chip design process.

4.1 Technology Stack

Now that the general architecture and features have been defined, this section will outline the technology stack we use to develop the application and the reasons for these choices.

- **Front-End Development - Vue3[1] and Nuxt3[2]**: We chose Vue3 and Nuxt3 for the front-end because they align with other projects at the chair, simplifying maintenance and code reuse. Vue3 offers good performance and a flexible composition API, while Nuxt3 provides server-side rendering and a structured framework for organizing application code.
- **Back-End Development - Spring Boot[3] and Spring Data MongoDB**: The back-end utilizes Spring Boot, which simplifies application development with its convention-over-configuration design. Spring Data MongoDB interfaces with MongoDB, our NoSQL database of choice, which is known for efficiently handling large amounts of structured and unstructured data, especially JSON documents.
- **Tool Services - TypeScript[4]**: Our tool services are designed to be flexible and efficient, often encapsulating binaries within minimal HTTP wrapper services using TypeScript. This setup allows for easy integration with our REST-based synchronous calling conventions. By using a microservice architecture, we can easily switch or update the technologies used for these wrappers, allowing us to use the best languages available for each tool without impacting the overall system. Furthermore, this architecture allows us to switch to a different protocol, for instance, a message broker, when we need asynchronous communication due to long-running tools.
- **Authentication and Authorization: Keycloak[5] and OAuth 2.0**: For authentication and authorization, we use Keycloak, which provides token-based authentication and integrates well with the other components of our technology stack. OAuth 2.0 offers secure, authorized access to resources without exposing user credentials, improving security.

[1]https://vuejs.org/
[2]https://nuxt.com/
[3]https://spring.io/projects/spring-boot
[4]https://www.typescriptlang.org/
[5]https://www.keycloak.org/

4.2 System Components

Now that we have outlined the goal for the application and defined our technology stack, we can specify the components we implement and integrate to realize the proposed system, combining information from the architecture sketch and the technology stack.

ONoC Designer: The front-end application, built with Vue3 and Nuxt3, is accessible through the user's web browser. It is a key component in our system, including two significant subcomponents: The Project Store and the Editor. The Project Store is responsible for retrieving project data and managing tasks such as creating, renaming, and deleting projects. The Project Editor, which consists of the canvas, settings, and various menus, allows users to configure designs and execute tools. Given the central role of the ONoC Designer, it has to communicate with all other system components.

Auth Service: For authentication and authorization, we will use Keycloak along with its OAuth 2.0 capabilities. This component ensures only authorized users can access the system's services by validating credentials and issuing access tokens. Given the wide adoption of OAuth 2.0, many client libraries, such as nuxt-oicd-auth and Spring Security, can efficiently integrate the Auth Service into both the front-end and back-end components.

Project Service: This service handles all project-related operations (e.g., saving, cloning, updating, exporting) and exposes an API to the front-end application. It will use Object Document Mapping to store, manage, and retrieve projects for each user. We will use Spring Boot and Spring Data MongoDB to efficiently interact with the database through an abstraction layer.

Tool Service: As an initial example, we plan to integrate the X-Talk tool. We will use a modified version of the crosstalk analysis described by A. Truppel, Tseng, and Schlichtmann [4], and wrap this tool's compiled binary with a minimal TypeScript HTTP service that exposes its functionality through an API. The tool operates based on a predefined export contract provided by the front-end editor, which standardizes how all tools in the system interact with project data (see section 4.3). With the wrapper service handling these adjustments, the tool can internally modify the data as needed, whether making significant or minimal transformations to the contract.

Database: We use MongoDB as the system's database, responsible for persisting the data managed by the Project Service. MongoDB provides flexibility and adaptability through an Object-Document Mapper in the back-end through its unstructured NoSQL approach, which allows for easy adjustments based on current demands.

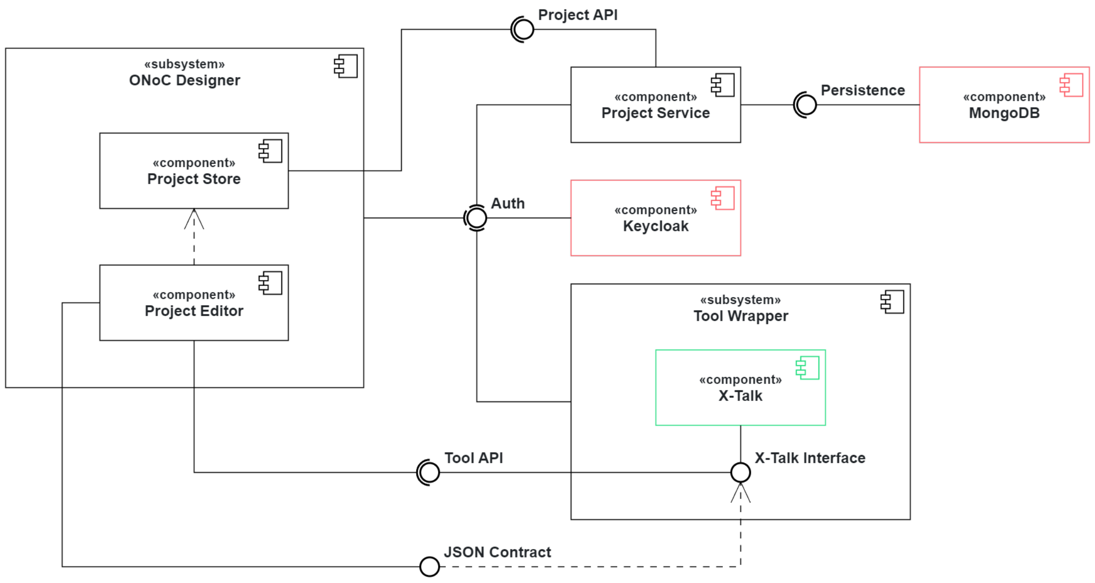


Figure 4.2: UML Component Diagram of the System. Red-outlined components represent off-the-shelf solutions, while green-outlined components highlight interchangeable components, with the Tool Wrapper adapting to the required interface.

To illustrate the relationships between these components, we use a UML component diagram, as shown in Figure 4.2. The diagram effectively demonstrates how components are separated by concerns, each exposing their specific functionality to our application. The ONoC Designer, as the front-end application, is our key component and should be highlighted. It provides initial user authentication by communicating with the Keycloak and retrieves projects from the persistence layer using the Project Service. Additionally, it provides a JSON contract that the Tool Wrappers can use to produce results displayed in our project editors.

4.3 Data Model

This section describes the user input our application requires to create, configure, and simulate a chip design. In addition to the technical data discussed in sections 4.3.1 and 4.3.2, we also need project metadata, such as project name, creation timestamp, userId, and editor version. These data pieces are fairly obvious and will not be discussed further. Instead, we will focus on the technical specifications of ONoCs.

4.3.1 Settings Specification

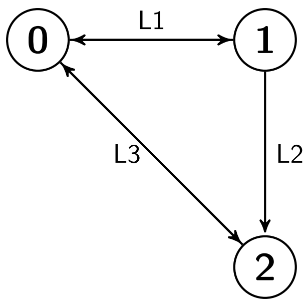
nodes	The number of nodes on the IC
chipWidth	The physical dimension of the chip's width in μm
chipHeight	The physical dimension of the chip's height in μm
nodeAliases	Key-Value Pair: Node indices as keys and corresponding aliases as values
signals	List of Signals as defined in Figure 4.4
laserCategory	`off-chip` or `on-chip`
laserType	`type-x` or `type-y`
technologyParameter	Key-Value Pair: User-defined set of config pairs; can be empty

Table 4.1: List of all available settings.

For each project, we need to identify configuration parameters the user can set according to their needs. The Table 4.1 contains the configuration settings available in our application. The number of nodes on the IC is denoted by the `nodes` parameter. The physical dimensions of the routing layer are defined by `chipWidth` and `chipHeight`, measured in μm. These values are relevant for chip layout but can be ignored when only planning a schematic chip design. The `nodeAliases` parameter can assign meaningful aliases to the `nodes`. The laser configuration, as described for the laser source in section 2.3, can be configured with `laserCategory` and `laserType`. Finally, `technologyParameter` is an open-ended list of configuration pairs that can be used to set various parameters required by some future simulation tools.

The `signals` parameter encodes two pieces of information: First, it carries the CM with the bandwidth requirements encoded in the matrix. Second, it carries the wavelength information of a signal. The wavelength is configured either symbolically (as a label) or by a list of resonance wavelengths, which depends on the size of the MRRs in the optical router.

Communication Matrix: This is an adjacency matrix M with dimensions $N \times N$, where N is the number of nodes. A value at position M_{ij} indicates a signal sent from node i to node j, with the value representing the minimum bandwidth requirement. Typically, the diagonals of the matrix M are zero because self-communication is not required [8]. We provide a matrix and a graph form input to configure the CM. The graph representation of the matrix is described as a tuple (i, j, v), where i and j are vertices and v describes the bandwidth value of the edge.



(a) Graph with Signal Labels

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

(b) Communication Matrix

Figure 4.3: Example Network with its corresponding Communication Matrix and symbolic signal wavelengths.

Signal Wavelength: This can be configured in two different ways. It can be a symbolic label, allowing for easy configuration, or a list of resonance wavelengths that travel on this signal. This list of wavelengths depends on the size of the MRRs as described in section 2.3.5.

Considering these explanations, we can parse a design and its CM to its corresponding signals. We use the graph in Figure 4.3a as an example design, and the corresponding CM of this graph can be seen next to it in Figure 4.3b. We have three nodes (0, 1, 2) and five signals by counting the non-zero values in A_{ij}. The labels on the edges of the graph represent the symbolic wavelengths used to carry the data. If we now export the signals to JSON, it would look like the example shown in Figure 4.4.

```
"signals": [
  { "from": 0, "to": 1, "value": 1, "accurate": [], "symbolic": "L1" },
  { "from": 0, "to": 2, "value": 1, "accurate": [], "symbolic": "L3" },
  { "from": 1, "to": 0, "value": 1, "accurate": [], "symbolic": "L1" },
  { "from": 1, "to": 2, "value": 1, "accurate": [], "symbolic": "L2" },
  { "from": 2, "to": 0, "value": 1, "accurate": [], "symbolic": "L3" },
]
```

Figure 4.4: An excerpt of the settings JSON displaying the signals.

4.3.2 Network Elements Specification

In the previous section, we discussed the data required for the project as a whole. Now, we will focus on modeling the network elements as draggable canvas components, their relationship in the hierarchy diagram, and the attributes required for exporting.

Since the components share many attributes, we use an inheritance hierarchy (Figure 4.5) to avoid code repetition and reuse common functionality. Interfaces and abstract classes isolate the functionality relevant to the chip designer from the technical specifications of the elements. For example, canvas interactions are handled by the `Draggable` interface, independent of any ONoC-related data.

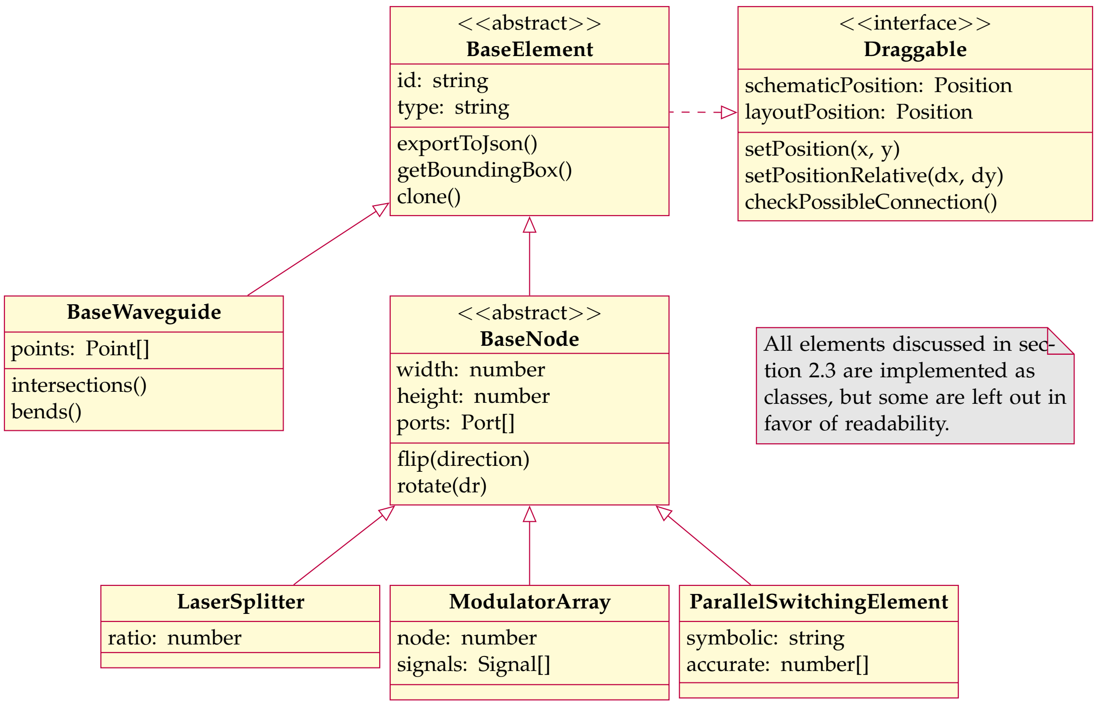
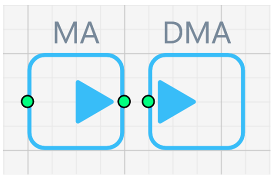


Figure 4.5: UML Class Diagram.

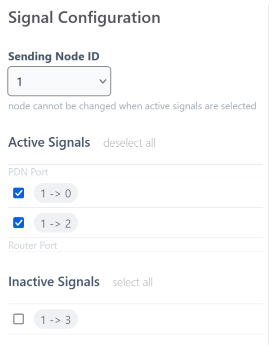
We differentiate between waveguides and nodes using the `BaseWaveguide` and `BaseNode` classes because of their different rendering methods on the canvas. Connection points draw waveguides, while nodes use predefined SVG images rendered inside containers defined by their `width` and `height` attributes. To further specify the elements, we subclass `BaseNode` to include all the necessary attributes based on the technical specifications discussed in section 2.3.

Modulator and Demodulator

Even though the Modulator Array (MA) and Demodulator Array (DMA) are built similarly, they have different ports for connecting to the chip. The MA requires two ports: one for the PDN and one for the optical router. In contrast, the DMA has only one port because the internal waveguide is terminated after decoding the received signal. Examples of their visual representations on the canvas can be seen in Figure 4.6a.



(a) MA and DMA components.



Signal Configuration

Sending Node ID

1

node cannot be changed when active signals are selected

Active Signals deselect all

PDN Port

- [✓] 1 -> 0
- [✓] 1 -> 2

Router Port

Inactive Signals select all

- [ ] 1 -> 3

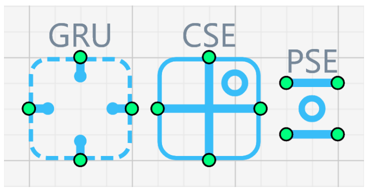
(b) MA signal configuration.

Figure 4.6: Overview of Components and Configuration UI for MA and DMA.

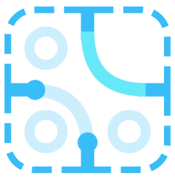
To configure these components, we can choose which signal is sent by a MA or received by a DMA. First, select the corresponding node. In the case of a MA a sender node. Next, select one or more signals to be encoded or decoded by the component. Again, in the case of a MA, we select the encoded signals. The selected signals can then be ordered by dragging them in the list closer to the input port or output/terminator. This ordering is important because we do not have singular modulators or demodulators, and we need a way to configure the internal ordering of the hidden MRRs of the arrays. An example of the configuration UI of a MA is shown in Figure 4.6b.

GRU, CSE, and PSE

All three components, GRU, CSE, and PSE, are part of the optical router and use MRRs to route signals. Each element requires configuration fields similar to the MRR. The PSE and CSE have a single configuration for all rings, whereas the GRU has a pair of configuration fields (accurate and symbolic) for each MRR placed. The MRR that interacts with adjacent waveguides needs a radius corresponding to the resonance and a field for a symbolic wavelength label. Since we do not have MRRs as standalone elements, these configuration parameters are used directly in the PSE, CSE, and GRU.



(a) Examples of optical router elements



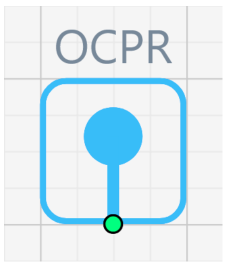
(b) Configuration for GRU internals

Figure 4.7: Optical Router Components.

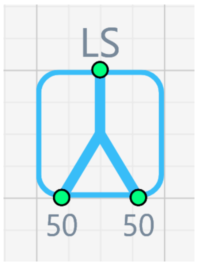
The CSE and especially the GRU also require a field for internal configuration that stores which elements are toggled on or off. An example of this can be seen in Figure 4.7b, highlighting the configuration of the internal state of a GRU. It automatically adapts to the design rules of the elements and shows possible elements with low opacity. To better visualize what will be toggled on click, the part is highlighted on hover in a lighter blue, e.g., the top right bend.

PDN Components

The Off-Chip Power Receiver (OCPR) does not require any component-specific properties. All of its properties are configured project-wide with *laserType* and *laserCategory* (see Table 4.1). Figure 4.8a shows an example of its visual representation.



(a) Off-Chip Power Receiver



(b) Laser Splitter

Figure 4.8: PDN Components.

The Laser Splitter, on the other hand, has an internal splitting ratio r, which is set in percent. It describes how much incoming power is sent through the left port and how much of the remaining $1 - r$ is sent through the right port. The default Laser Splitter shown in Figure 4.8b has a splitting ratio of 50%. The splitting ratio is also displayed when the user hovers over the element to help identify the ports.

Waveguide

A series of points define waveguides. These points are connected to form a line that represents the channel through which the light travels. For optimization purposes, the total length of the waveguide, the number of bends, and the intersections with other waveguides are critical. Additionally, we need to know this information in order, e.g., first one bend, then one intersection, followed by two bends.

It is important to note that the number of points does not automatically equal the number of bends. Additional calculations are required to determine whether a point is a bend. For example, consider a waveguide consisting of three points. Naively, the number of bends can be calculated by subtracting two (one for the start and one for the end). However, if the three points are on a straight line, the number of bends is zero. Therefore, we must analyze the geometric arrangement of the points to get the accurate number of bends.

4.4 JSON Contract

Now that we have modeled our settings and components in detail, this section outlines the JSON contract allowing us to export this data to design automation tools. This interface will serve as the backbone for data exchange and integration with all the external tools in our system. The contract ensures that our application can seamlessly communicate and update designs based on input from various optimization and simulation tools. There are several advantages to adopting a well-defined import and export API:

- **Standardization:** Enforces a consistent format for data exchange, reducing errors and conflicts.
- **Scalability:** Makes it easy to scale the application by integrating new external tools or systems with minimal changes.
- **Flexibility:** Exporting the entire design with all attributes makes our interface flexible, allowing us to integrate a broad set of tools with the same interface.
- **Maintenance:** Simplifies system maintenance and upgrades because changes to external interfaces do not affect internal processing logic.

The interface is defined using a JSON-based structure (see Figure 4.9) that includes the interface version, project name, `settings` parameters (see section 4.3.1), and `elements`. The `elements` object contains all components indexed by their IDs. This structure allows for easy indexing and efficient element resolution, especially for connections referenced by ID. Internally, elements are stored in a map, so we aim for a similar structure after export.

```
{
  "version": "v1",
  "project_name": "Crossbar_Router_Example",
  "settings": {...},
  "elements": { "<id>": {...}, ... }
}
```

Figure 4.9: Top-level structure of the exported JSON.

Node Specification

A node has a component type (because the class is lost when we export to a JSON object), two positions (one for each canvas), connection information for all ports, and modulator component-specific data (`node` and `signals`).

```
{
  "type": "ma",
  "ports": {
      "input": <waveguide_id | null>,
      "output": <waveguide_id | null>
  },
  "sch_pos": { "x": 10, "y": 20, "rotation": 0, "flipHorizontal": false, "flipVertical":
       false },
  "lay_pos": { "x": 15, "y": 25, "rotation": 0, "flipHorizontal": false, "flipVertical":
       false },
  "node": 0,
  "signals": [...]
}
```

Figure 4.10: Example of a MA in JSON format.

Waveguide Specification

Waveguides include schematic and layout positions consisting of lists of points. They also contain a list of bends and a list of crossings with other waveguides, which are required for tools like X-Talk.

```json
{
  "type": "wg",
  "sch_points": [{ "x": 10, "y": 20}, { "x": 30, "y": 40}],
  "layout_points": [{ "x": 15, "y": 25}, { "x": 35, "y": 45}],
  "connections": [<node_id | null>, <node_id | null>],
  "crossings": [{ "id": <waveguide_id>, "x": 25, "y": 35}],
  "bends": [{ "x": 20, "y": 30}]
}
```

Figure 4.11: Example of a waveguide object in JSON format.

4.5 Integrated Tools

Due to the limited time frame of this thesis, we will integrate a small subset of tools to demonstrate that the ideas and concepts we have developed are practical and lead to successful outcomes. We have chosen to focus on topology generation tools to bootstrap the setup of standard chip designs and crosstalk analysis (*X-Talk* tool) to perform meaningful simulations on our designs.

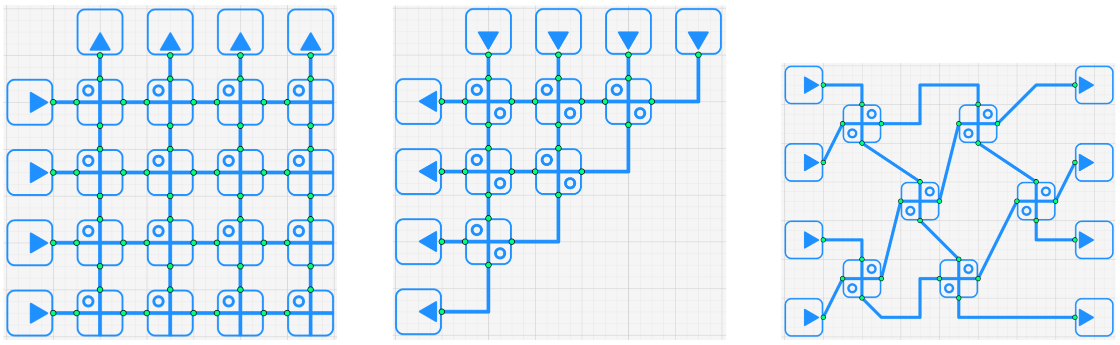
4.5.1 Topology Generators

Topology generation tools are essential for quickly bootstrapping standard chip designs. These tools automate the initial design phase by generating standard topologies that are optimized starting points for chip designs. Since these designs must be placed in our chip designer and are relatively computationally lightweight, we implement them purely as editor functions in the front-end. Although they are not tools in the sense of being separate services, they serve a similar purpose for the user and are listed in the tools window.

These common topologies in WRONoCs are primarily constructed using CSEs. The CSE, a critical building block in WRONoC architectures, routing signals based on the resonance wavelengths dictated by the radii of the internal MRR. Essentially, it determines the path a signal takes through the waveguides. These designs provide a schematic layout describing the connection of elements and the internal configuration needed for communication but do not include the actual placement on the chip, which is another complex optimization problem [8]. For our system we focused on three well-known CSE-based topologies connecting n nodes, with their corresponding n MAs and n DMAs:

- **Crossbar**: This topology utilizes a squared (n^2) grid of 1x2 CSEs configured with n different MRR radii [8]. An example is shown in Figure 4.12a.

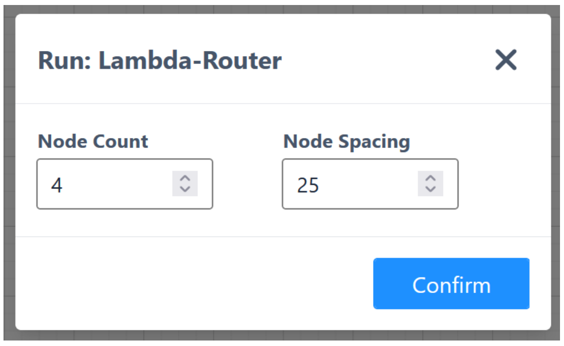
- **Lambda and Snake**: This topology uses $\frac{n(n-1)}{2}$ 2x2 CSEs with n different MRR radii [11], [15]. Examples of these structures are shown in Figure 4.12b and 4.12c.



(a) Crossbar Router **(b)** Snake Router **(c)** Lambda Router

Figure 4.12: Example topologies generated for a design with $N = 4$ nodes.

Topologies can be automatically generated as starting points in our chip designer. As shown in Figure 4.13, the user is prompted for the number of nodes and the spacing between elements when running one of these tools. This enables quick testing and simulation of known layouts without manual construction. Furthermore, these topologies are fully configured at generation time. Their symbolic wavelengths are pre-filled to their specifications in the papers to further reduce the setup process.



Run: Lambda-Router

Node Count: 4

Node Spacing: 25

Figure 4.13: Modal window for Lambda Router generation.

4.5.2 X-Talk (Crosstalk)

Crosstalk analysis in ONoCs is essential due to the sensitivity of optical signals to noise. It describes the interference caused by signals in adjacent waveguides within the network. This interference can significantly reduce the Signal-to-Noise Ratio (SnR), which affects the performance and scalability of ONoCs. Accurate crosstalk analysis is, therefore, key to predicting and mitigating noise in designs. Since WRONoCs uses multiple wavelengths to route optical signals, it is even more important to accurately calculate crosstalk for our designs [4].

To perform a comprehensive crosstalk analysis for ONoCs, the following inputs are necessary. However, we have already provided most of the required data with our interface, and only a tiny part needs to be configured manually.

- **Network Design and Signal Paths:** Detailed layout of the ONoC, including the configuration and interconnections of all optical elements ($\rightarrow$ canvas).
- **Wavelength Information:** Specific wavelengths used for signal transmission in the network ($\rightarrow$ settings).
- **Component Characteristics:** Properties of the optical components, such as insertion loss, crosstalk coefficients, and resonance wavelengths of MRRs [4].

The power levels calculated by the crosstalk method are displayed on the waveguides as shown in Figure 4.14. Each waveguide transmits optical power that is characterized by several factors. First, light can travel in two directions on a waveguide: *left to right* and *right to left*, visually represented by arrows indicating the direction. Second, each wavelength has a distinct power value since each waveguide carries light at multiple wavelengths used by the ONoC represented by the opacity of the arrow. Third, the power can be either a signal (intended power, shown in green) or noise (unintended power, shown in red). Therefore, the tool produces values for each waveguide based on travel direction, wavelength, and type (signal/noise). In addition, the tool calculates the SnR for each signal the demodulator receives. This ratio indicates signal reception by balancing the intended signal against the interfering noise [4], [8].

Due to time constraints, the integration of the X-Talk Tool in our system has to be omitted. Although the overlay for the crosstalk analysis is already scaffolded, the tool needs adjustments, which is out of the scope of this thesis work. Thus, we cannot integrate it yet in the design automation process.

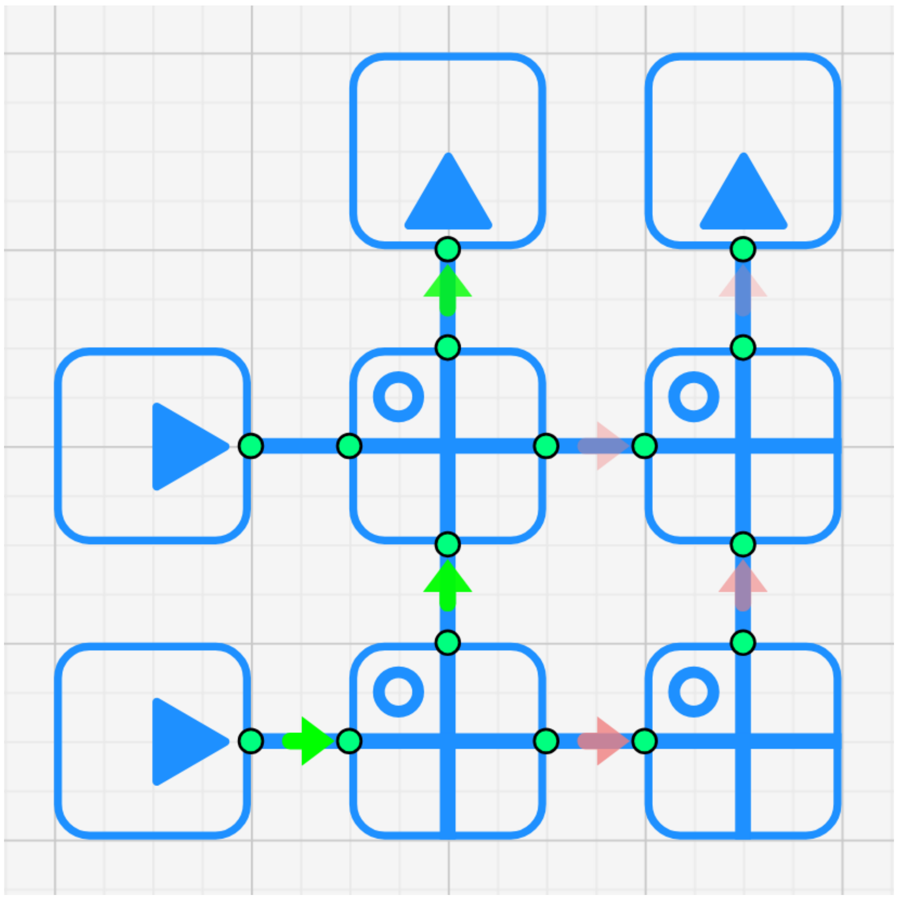


Figure 4.14: Exemplary crosstalk analysis results for a 2x2 Crossbar Router to visualize the interface.

5 Conclusion

This chapter concludes the thesis by evaluating the current status of the proposed system and its realized goals, as well as the open goals in section 5.1. After that, we will explore possible future work in section 5.2 and summarize the results of this thesis in the final section 5.3.

5.1 Status

We now compare the requirements described in chapter 3 with the current status of the system. We use the following notation to display the status of each requirement:

> ● **Implemented:** The requirement is fully implemented.
>
> ◐ **Partially Implemented:** The requirement is only partially implemented, and more work is needed.
>
> ○ **Not Implemented:** The implementation of this requirement has not been started.

As shown in Table 5.1, the developed system fully meets seven of the eleven functional requirements. Users can create, import, save, export, delete, and clone projects as expected (FR1.1). Chip design functionality allows users to place, drag, connect, delete, and configure chip elements (FR1.2). Panel synchronization is partially implemented, enabling users to work with the same chip in both schematic and layout panels, but true-to-life element sizes and measurement features are still missing (FR1.3). The settings panel is fully functional, allowing users to view and configure all settings in a dedicated panel (FR1.4). Tool integration is partially implemented. Users can run topology generation in the designer. However, the crosstalk analysis is not yet fully integrated, which was the intended scope of the thesis (FR1.5).

The back-end supports user registration and login functionality, ensuring secure access to the platform (FR2.1, FR2.2). Project storage is fully implemented, storing all project data (metadata, canvas configuration, simulation results, ...) linked to a user. (FR2.3). Furthermore, the tool interface required by the wrapper services is fully specified, allowing interaction with ONoC related tools (FR3.1). Synchronous tool execution is, in theory, fully supported, providing real-time results for tools like the *X-Talk*, but not tested because the integration is missing due to time constraints. Thus, we count it as partially implemented (FR3.2). Asynchronous tool execution has not yet been implemented because we do not need it for our selected tools (FR3.3).

Functional Requirements	**Status**
FR1.1 Project Management	●
FR1.2 Chip Design	●
FR1.3 Panel Sync	◐
FR1.4 Settings Panel	●
FR1.5 Tool Integration	◐
FR2.1 User Registration	●
FR2.2 User Login	●
FR2.3 Project Storage	●
FR3.1 Tool Interface	●
FR3.2 Sync Tool Exec	◐
FR3.3 Async Tool Exec	○

Table 5.1: Overview of the status of the functional requirements.

Having reviewed all the functional requirements, we can now look at the non-functional ones. As shown in Table 5.2, the developed system meets seven of the eight non-functional requirements to varying degrees. A quick add feature allows users to efficiently add new elements within three clicks and little mouse movement, enhancing overall usability (NFRU.1). Design elements maintain a broadly consistent style, clearly illustrating waveguides, outlines, and connection points (NFRU.2). Navigation is fully supported, enabling users to locate any UI element quickly within 30 seconds (NFRU.3). Systems stability is independent of tool services. Although the X-Talk integration is currently missing, we can still use the ONoC Designer without problems (NFRR.1). The system can manage approximately 1000 concurrent elements. However, further performance optimizations are necessary for larger designs (NFRP.1). Stability is maintained with designs up to 16 nodes, fulfilling the specified requirements (NFRP.2). With the crosstalk analysis not being integrated yet, tool execution speed for synchronous tools is not measurable (NFRP.3). Full browser support is provided for Chrome and Firefox, with hotkeys mapped for both Windows and macOS, thus fully meeting the requirement (NFRS.1).

Non-functional Requirements	**Status**
NFRU.1 Quick Add	●
NFRU.2 Consistent Style	●
NFRU.3 Easy Navigation	●
NFRR.1 System Stability	●
NFRP.1 Component Limit	◐
NFRP.2 Node Limit	●
NFRP.3 Tool Speed	○
NFRS.1 Browser Support	●

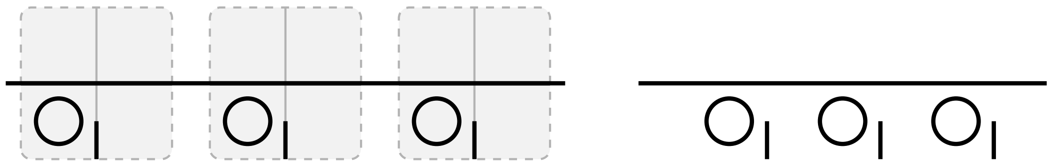
Table 5.2: Overview of the status of the non-functional requirements.

5.2 Future Work

In addition to the unfulfilled requirements listed previously, we will present some visionary features in this section to further improve our system's capabilities and usability.

MRR Component

We have focused on implementing eight distinct elements, allowing users to create and configure chip designs. By using pre-built components like CSEs, PSEs, and GRUs, we have abstracted away the challenge of routing signals manually with MRRs adjacent to waveguides. It is possible to introduce a new freely movable MRR element that will attach to one or more nearby waveguides if they are close enough. This addition would allow for a more flexible fine-grained chip design. Examples of this can be seen in Figure 5.1, which compares the current approach using GRUs with the possibilities of the new system.



(a) Using GRUs **(b)** Using Waveguides and MRRs

Figure 5.1: Comparison of GRU based approach versus freeform MRRs.

At first glance, this seems like it does not add any advantage, but we must remember that waveguides can have arbitrary shapes. Freeform MRRs can adapt very well to, for instance, diagonal placements, whereas GRUs are restricted to their positions on the canvas grid.

Design Automation Tools

Due to time constraints, it was only possible to implement some of the tools introduced in section 2.4 within the scope of this thesis. For the actual usability of the system, it would be a great addition to integrate a broader list of tools. This would introduce new challenges, such as asynchronous execution, possibly using a message broker (e.g., RabbitMQ[1]) to handle long-running and computation-heavy jobs.

The interface we designed and implemented in chapter 4 should already contain most of the data required to run other tools. However, these tools must be adapted to this interface and made accessible in a small wrapper service by exposing an API.

Performance Optimizations

We have tested to create designs with up to 16 nodes, which, for example, results in around $512(=16x16x2)$ elements in a crossbar topology. With this amount of nodes, the application runs smoothly. However, as the number of nodes increases, the element count snowballs, slowing down the chip designer and eventually crashing the browser (e.g., a 128-node crossbar router). By identifying bottlenecks, the application can be adjusted to work as performant with larger numbers of nodes as it does with 16 nodes today.

5.3 Summary

In this thesis work, we have presented a comprehensive approach to developing a design automation platform for ONoCs. We have successfully implemented and tested key components, demonstrating the potential and advantages of using a cloud-based microservice architecture for such a system.

With the started integration of the crosstalk analysis, our system offers the first fundamental feature that can contribute to further research and improvement of optical networks. Compared to the current approaches, we have created a user-friendly way to create chip designs for ONoCs and increase the accessibility of ONoC related tools overall. However, as outlined in section 5.2, there are many opportunities to build upon our application to further improve the *Online ONoC Chip Designer*.

[1]https://www.rabbitmq.com/

Abbreviations

API Application Programming Interface

CM Communication Matrix

CSE Crossing Switching Element

DMA Demodulator Array

DRC Design Rule Check

GRU Generic Routing Unit

IC Integrated Circuit

MA Modulator Array

MRR Micro-Ring Resonator

NoC Network-on-Chip

OCPR Off-Chip Power Receiver

ONoC Optical Network-on-Chip

OOK On-Off Keying

PCB Printed Circuit Board

PDN Power Delivery Network

PIC	Photonic Integrated Circuit
PSE	Parallel Switching Element
SnR	Signal-to-Noise Ratio
SoC	System-on-Chip
UI	User Interface
WRONoC	Wavelength-Routed Optical Networks-on-Chip

List of Figures

List of Tables

Bibliography

[1] T.-M. Tseng, A. Truppel, M. Li, M. Nikdast, and U. Schlichtmann, "Wavelength-routed optical nocs: Design and eda — state of the art and future directions: Invited paper," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–6. DOI: `10.1109/ICCAD45719.2019.8942092`.

[2] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. G. Beausoleil, and J. H. Ahn, "Corona: System implications of emerging nanophotonic technology," *SIGARCH Comput. Archit. News*, vol. 36, no. 3, pp. 153–164, Jun. 2008, ISSN: 0163-5964. DOI: `10.1145/1394608.1382135`.

[3] S. Werner, J. Navaridas, and M. Luján, "A survey on optical network-on-chip architectures," *ACM Comput. Surv.*, vol. 50, no. 6, Dec. 2017, ISSN: 0360-0300. DOI: `10.1145/3131346`.

[4] A. Truppel, T.-M. Tseng, and U. Schlichtmann, "Accurate infinite-order crosstalk calculation for optical networks-on-chip," *J. Lightwave Technol.*, vol. 41, no. 1, pp. 4–16, Jan. 2023.

[5] M. Xiao, T.-M. Tseng, and U. Schlichtmann, "Crosstalk-aware automatic topology customization and optimization for wavelength-routed optical nocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5261–5274, 2022. DOI: `10.1109/TCAD.2022.3151247`.

[6] A. Truppel, T.-M. Tseng, and U. Schlichtmann, "Psion 2: Optimizing physical layout of wavelength-routed onocs for laser power reduction," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20, Virtual Event, USA: Association for Computing Machinery, 2020, ISBN: 9781450380263. DOI: `10.1145/3400302.3415655`.

[7] F. Ponce, G. Márquez, and H. Astudillo, "Migrating from monolithic architecture to microservices: A rapid review," in *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, 2019, pp. 1–7. DOI: `10.1109/SCCC49216.2019.8966423`.

[8] A. C. Truppel, "Design automation for wavelength-routed optical networks-on-chip [unpublished doctoral dissertation]," PhD thesis, Technical University Munich, Munich, Germany.

[9] M. Li, T.-M. Tseng, D. Bertozzi, M. Tala, and U. Schlichtmann, "Customtopo: A topology generation method for application-specific wavelength-routed optical nocs," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8. DOI: 10.1145/3240765.3240789.

[10] X. Tan, M. Yang, L. Zhang, Y. Jiang, and J. Yang, "On a scalable, non-blocking optical router for photonic networks-on-chip designs," in *2011 Symposium on Photonics and Optoelectronics (SOPO)*, 2011, pp. 1–4. DOI: 10.1109/SOPO.2011.5780550.

[11] M. Li, T.-M. Tseng, M. Tala, and U. Schlichtmann, "Maximizing the communication parallelism for wavelength-routed optical networks-on-chips," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 109–114. DOI: 10.1109/ASP-DAC47756.2020.9045163.

[12] M. Tala, L. Ramini, and D. Bertozzi, "Exploring communication protocols for optical networks-on-chip based on ring topologies," in *2014 Asia Communications and Photonics Conference (ACP)*, 2014, pp. 1–3.

[13] A. V. Beuningen, L. Ramini, D. Bertozzi, and U. Schlichtmann, "Proton+: A placement and routing tool for 3d optical networks-on-chip with a single optical layer," *J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 4, Dec. 2015, ISSN: 1550-4832. DOI: 10.1145/2830716.

[14] Z. Liu, H. Gu, K. Wang, B. Zhang, and K. Wang, "A novel laser source supply scheme for optical network on chip," *IEEE Access*, vol. 7, pp. 25 872–25 877, 2019. DOI: 10.1109/ACCESS.2019.2899662.

[15] M. Ortín-Obón, M. Tala, L. Ramini, V. Viñals-Yufera, and D. Bertozzi, "Contrasting laser power requirements of wavelength-routed optical noc topologies subject to the floorplanning, placement, and routing constraints of a 3-d-stacked system," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 7, pp. 2081–2094, 2017. DOI: 10.1109/TVLSI.2017.2677779.

[16] B. Little, S. Chu, H. Haus, J. Foresi, and J.-P. Laine, "Microring resonator channel dropping filters," *Journal of Lightwave Technology*, vol. 15, no. 6, pp. 998–1005, 1997. DOI: 10.1109/50.588673.

[17] Z. Zheng, M. Li, T.-M. Tseng, and U. Schlichtmann, "Topro: A topology projector and waveguide router for wavelength-routed optical networks-on-chip," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9. DOI: 10.1109/ICCAD51958.2021.9643451.

[18] Y.-K. Chuang, K.-J. Chen, K.-L. Lin, S.-Y. Fang, B. Li, and U. Schlichtmann, "Planaronoc: Concurrent placement and routing considering crossing minimization for optical networks-on-chip," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18, San Francisco, California: Association for Computing Machinery, 2018, ISBN: 9781450357005. DOI: `10.1145/3195970.3196093`.

[19] Synopsys, *Optsim circuit for single mode network*, Accessed: 2024-07-26, 2024.

[20] Y. Zhang, M. Li, T.-M. Tseng, and U. Schlichtmann, "Open-source interactive design platform for 3d-printed microfluidic devices," *Communications Engineering*, vol. 3, no. 1, p. 71, May 2024, ISSN: 2731-3395. DOI: `10.1038/s44172-024-00217-0`.

[21] B. Bruegge and A. H. Dutoit, *Object Oriented Software Engineering Using UML, Patterns, and Java*, 3rd. Prentice Hall, 2010.