Circuit validation in Nazca Design for mask layout

M. Passoni[1] and R.G. Broeke[1]

[1]*Bright Photonics B.V., Horsten 1, 5612 AX Eindhoven, the Netherlands*

We present a connectivity validator as well as circuit simulation capability for photonic IC design in Nazca Design, an open-source PIC design framework in Python3. The validator, Pathfinder, and linear simulation perform layout vs schematic validation at the connectivity and functional level. The circuit simulation runs directly on a netlist, which also generates GDS layout and, therefore, avoids going back and forward between a layout and circuit simulator tools. With accurate compact models for the building blocks, linear simulation will predict device performance after fabrication, while also confirming the netlist underpinning the GDS layout to be manufactured.

Introduction

Optimizing Photonic Integrated Circuit design flows has been a hot topic in the photonics community for quite some years [1]. Nevertheless, an ideal photonic design workflow, e.g. modelled on the EDA approach typical for analog electronics, is far from developed. Two often emerging problems are, firstly, the lack of tools able to handle commonplace features of EDA, such as schematic-driven-layout and layout-versus-schematic (LVS) verification [2], and secondly, the lack of libraries of well characterized and verified building blocks. In this paper we will discuss how Nazca Design can help to bring the photonic community closer to such mature design flow. In particular, we discuss the insertion of compact models into Nazca and two modules for layout-versus-schematic validation, i.e. path tracing and linear circuit simulation.

Nazca Netlist

At its core, each Nazca Design layout is a graph with nodes and edges, referred to as “the Nazca tree”. More generally, nodes can represent different kinds of objects, of which the most relevant for the user are cells, instances and pins. Edges are connections between nodes which carry information about the relation between the connected objects, e.g. layout position, distance or optical paths. For added flexibility, multiple edges carrying different information can exist between the same two nodes simultaneously. Almost every operation in Nazca is a manipulation of the Nazca tree. For example, putting elements in a layout automatically builds connections between pins based on geometrical proximity of pins and pin properties, like the type of waveguide they represent. Nazca’s flexibility is ultimately derived from the flexibility of its underlying graph and data architecture.

Once the graph is defined, different views can be defined for different use cases. Also exporting the layout in GDS format is fundamentally just a view on the Nazca graph. In this framework, defining additional views for connectivity checks, circuit simulation or layout-versus-schematic validation is quite straightforward.

However, these views need information beyond the pure geometrical data embedded within the Nazca tree for mask layout. In particular, the building blocks (cells) used in the layout need to have information about their internal connectivity, or functional behavior, stored in a compact model. This is solved within Nazca by again leveraging the netlist concept and compact models provided via foundry PDKs or PDK extensions. In practice, the compact model of a building block is stored as connections between the

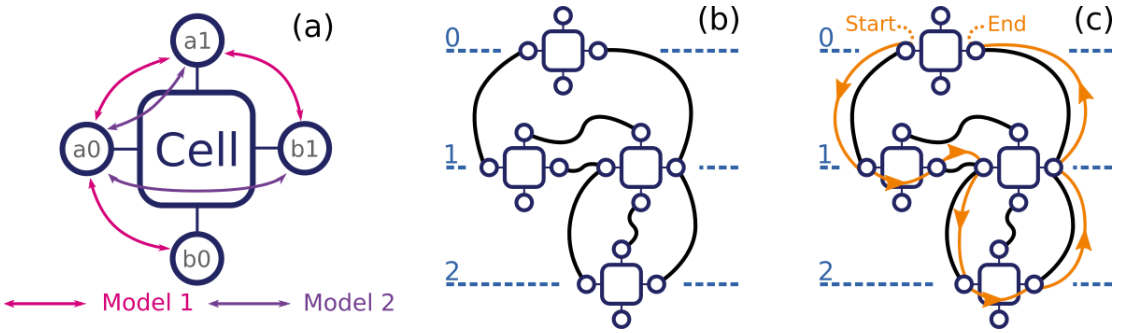


Figure 1: (a) Representation of a Nazca cell with pins a0, etc., and two compact models shown as connections between pins. (b) Schematic of the Nazca tree showing connectivity between various cells and cell hierarchy. (c) The pathfinder navigates a path through the structure from b in yellow.

nodes representing its pins (see Figure 1a). Since the connection information can take the form of any (Python) object, this format is intrinsically modular and easy to extend. Moreover, the fact that multiple edges can exist between two nodes allows the cell to have multiple compact models defined at the same time, each one carrying either information about a different physical aspect, e.g. optical, electrical or thermal behavior, or about the same physical aspect with different levels of precision.

In this paper we present two examples of netlist verification views of the Nazca tree. The first approach consists of tracing paths through the layout to find and verify connectivity. The second one extracts the compact models and circuit topology from the graph to perform a linear simulation describing the circuit behavior.

Pathfinder module

The pathfinder is a Nazca module which allows for the tracing of paths in a layout. For the purpose of this paper, a path in a layout is a sequence of connected cells representing a signal propagating through the circuit, e.g light path or electrical connection. The pathfinder can start from a cell or instance pin and navigate the Nazca tree and the compact models of the cells. It collects all paths starting from the specified pin, and is able to navigate the full hierarchy of the Nazca tree by using connectivity information stored in the compact model of the cell or by drilling down the tree if no compact model is found within the cell (Figure 1c). The utility of pathfinder is twofold. Firstly, it can verify that inputs are connected to the desired outputs, automating a process that would otherwise consist of a very tedious and unreliable manual verification step. Secondly, it can return all the compact models in the path, and extract information from them. For example, if the extracted compact model contains info about the geometrical length of

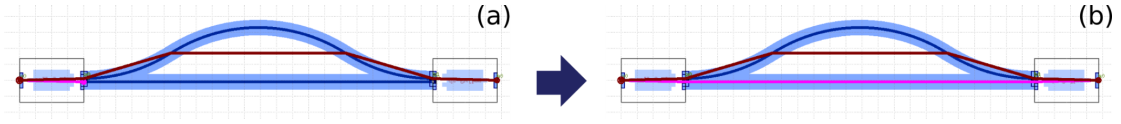


Figure 2: MZI with (a) and without (b) connection error. Blue shapes represent waveguides and trenches, the boxes MMI blocks, and the brown and pink lines represent two paths on top of the layout using straight lines. As can be seen, when the connection error is present the (pink) pathfinder is not going through. The example is created using the demofab PDK provided with Nazca.

the components, the pathfinder can be used to get the total length of the path. Other useful compact models that could be defined are loss (for power budget calculation), optical length (for phase matching), and time delay (for matching of signal timing). Others could be added if needed. These pathfinder use cases are discussed below by example. Figure 2a shows the layout of a simple asymmetric MZI (Mach-Zehnder interferometer), which has been purposefully designed with a mistake, namely a gap of 1 um between the bottom waveguide and the second MMI. In this example, the pathfinder starts to trace from the input pin of the first MMI. Two paths can be observed: one through the spiral and one through the straight waveguide. The second one clearly shows broken connectivity, as the bottom path is not arriving at the output of the second MMI, while the top one does.

In addition to making connectivity errors more evident to a visual inspection of the GDS, the pathfinder module also provides a way to automate connectivity checks. Indeed, leveraging the ability of the pathfinder to return the full list of visited cells, the designer can implement ad hoc connectivity checks directly in the script that generates the layout. Moreover, the full collection of paths can be logged and stored for future connectivity reference.

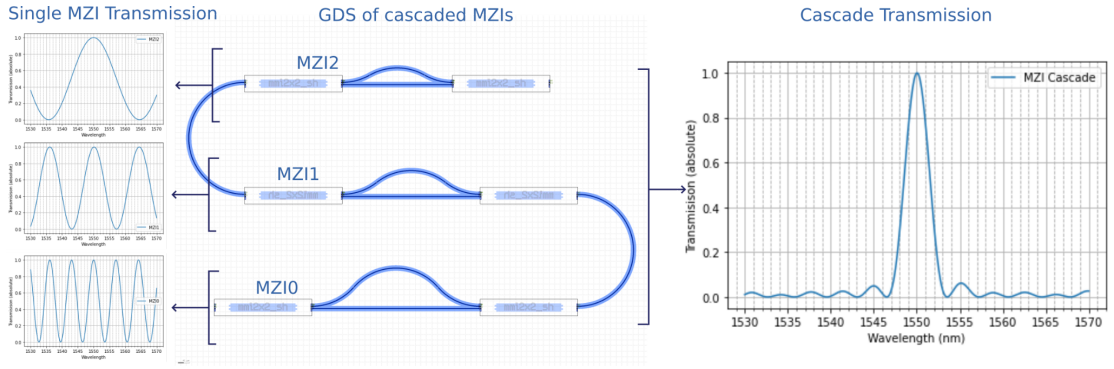
Once an error is identified and corrected (Figure 2b), the pathfinder can be employed to extract useful data for design validation. As an example, the lengths of the two arms of the MZI are measured: 594.663 and 619.517 μm, respectively. Being able to reliably extract this kind of information directly from the layout can be a great help to the designer as part of layout validation.

Linear circuit simulation

The Nazca tree and the compact models provide a solid base, which can be extended well beyond the pathfinder approach. Here, we demonstrate that with a Nazca module for linear circuit simulation running directly on the Nazca tree. Although linear circuit simulations have limitations, it has been clearly proven that they are a valuable tool to quickly assess the basic functionality of any photonic integrated circuit [3]. In parallel we have developed a similar tool based on the scattering matrix method, which obtains the full response of the circuit to any linear excitation by gradually joining the scattering matrices of all the building blocks involved. The tool takes the hierarchical nature of a circuit into account from the start and builds the scattering matrix of blocks starting from the compact models stored in Nazca. Thus, the designer can, with minimal extra code (see snippet in Figure 3) get the full linear response of the circuit using Nazca as netlist builder, resulting in seamless integration with layout and PDKs.

A typical use case for this kind of simulation is presented in Figure 3. There, we want to realize a filter using a cascade of three Mach-Zehnder interferometers. In order to have a narrow transmission window, the easiest way is just to tune the MZIs to have FSRs (free spectral range) that are multiples of each other. This task is less trivial than it seems, because depending on the MZI geometry and waveguide optical characteristics, analytical formulas for calculating the shape, and/or refractive indices the arms may not be available. The final verification of the proper tuning of the MZI would be left as tedious “homework” for the designer. Fortunately, this is solved by having access to a linear circuit simulator that runs directly on the same graph used to build the layout. In

this particular case, getting the transmission spectra of the single MZI and of the cascade is enough to exclude errors in the design phase.




```
cascade = create_mzi_cascade()  # create Nazca cell as usual
sol = nd.get_solver(cascade)  # get solver object form the cell
mod = sol.solve(wl=np.linspace(1.53, 1.57, 501))  # create scattering matrix
data = sol.get_data("a0", "b0")  # get Pandas dataframe with transmission from pin a0 to b0
```

Figure 3: Example of simulation spectra extracted from a Nazca layout. The analyzed structure is a cascade of MZI (center). Both the transmission spectra through the bar channel of the single MZIs (left) and through the full cascade (right) are reported. The code snippet to obtain transmission spectra from the Nazca cell, here stored in the variable 'cascade', is shown at the bottom.

Conclusion

We showed how we can leverage the flexibility of Nazca Design to extend it with layout-versus-schematics validation with minimal additional coding effort from the designer. In particular, we discussed how compact models for building blocks can be embedded in the Nazca graph and how different views on said graph can be used to obtain different kinds of information directly from the layout script. We provided path tracing and linear circuit simulation examples. These tools, coupled with high quality compact models, are a critical asset in the hands of the designer, automating the layout validation process. It reduces the probability of error and increases the predictability of the performance, saving cost and time by reducing the development cycles.

This paper is supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 824980, project InPulse.

References

[1] Bogaerts, Wim, and Lukas Chrostowski. "Silicon photonics circuit design: methods, tools and challenges." *Laser & Photonics Reviews* 12.4, 1700237, 2018.

[2] Khan, Muhammad Umar, et al. "Photonic integrated circuit design in a foundry+ fabless ecosystem." *IEEE Journal of Selected Topics in Quantum Electronics* 25.5, 1-14, 2019.

[3] Ploeg, Sequoia, Hyrum Gunther, and Ryan M. Camacho. "Simphony: An open-source photonic integrated circuit simulation framework." *Computing in Science & Engineering* 23.1, 65-74, 2020.

[4] Leijtens, X. J. M., P. Le Lourec, and M. K. Smit. "S-matrix oriented CAD-tool for simulating complex integrated optical circuits." *IEEE Journal of Selected Topics in Quantum Electronics* 2.2, 257-262, 1996.