A software language approach for describing and programming photonics hardware

Sébastien d'Herbais de Thun
Promoters: *Prof. dr. ir. Wim Bogaerts, Prof. dr. ir. Dirk Stroobandt*
Ghent University
sebastien.dherbaisdethun@ugent.be

***Abstract*— This research presents a novel way of designing and describing photonic circuits, allowing for faster and more productive design and simulation of photonic circuits. This is achieved using a software language approach, where the photonic circuit is described in a software language, which is then compiled into a photonic circuit.**

***Index terms*—Photonic, Circuit design, Photonic processors, Programmable photonic**

I. Introduction

Integrated photonics is a growing industry that benefits from the high yields and technological maturity of the CMOS industry. However, it has not yet reached the ease of design and simulation that the digital electronics industry has enjoyed for many years. This lack of maturity in the tools and design flows is a hurdle that has led to a slower time to market for photonic integrated circuits. When coupled with the lack of prototyping platforms, it has made integrated photonics an expensive market to enter. In recent years, several teams around the world, including at *Ghent University*, have been working on remediating this issue by developing a prototyping platform called a photonic processor. This processor is a programmable photonic integrated circuit that can prototype photonic circuits and replace photonic ASICs in some applications [1, 2, 3].

This paper will introduce a new hardware description language called PHÔS, which aims at providing a software language approach to photonic circuit design, being compatible with both, the programming of photonic processors and the creation of standalone photonic integrated circuits (PICs).

A. Paper overview

This paper will start by introducing photonic processors, the niche they fill, and their potential applications. Followed by an introduction to the core concepts of the PHÔS programming language and how PHÔS can be used to describe photonic circuits efficiently and productively. Then, this work will present an example written in PHÔS of a 16-QAM 400Gb/s transmitter, showing how the language can describe a complex photonic circuit. Finally, a discussion of future work and the conclusion will close this paper.

II. Photonic processors

Photonic processors, also called photonic FPGAs [4], are reprogrammable PICs designed to be generic enough to express most passive photonic circuits [2]. They are the photonic analogue to digital electronics' FPGAs, with the significant difference being that they represent analog circuits instead of digital ones. Photonic processors are built from gates, seen in Figure 1, arranged in hexagonal grids. It also shows the multiple operating modes of a photonic gate: the bar mode (b) which does not couple the optical signals together, the cross mode (c) which fully couples the optical signals from one waveguide to the other, and partial mode (d) which partially couples the optical signals from one waveguide to the other. The partial mode can split signals, combine them, interfere them, etc.

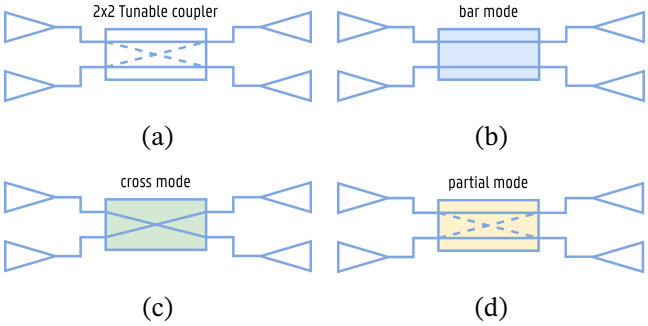
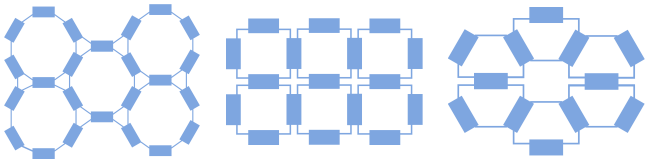


Figure 1: Photonic gate and its states

Photonic gates can be arranged in several ways; the most typical are hexagonal grids, as seen in Figure 2 (a), other configurations such as a square mesh Figure 2 (b), and triangular meshes Figure 2 (c) also exist. However, the hexagonal mesh provides the best routing capabilities. There has been ongoing research into the routing of photonic meshes [5, 6, 7]. However, the placement aspect of photonic processors has not been resolved yet.



(a) (b) (c)

Figure 2: Common mesh topologies in photonic processors

A. Use cases

Besides their use as prototyping platforms, there are situations where photonic processors may be used independently and integrated into a system directly. Photonic processors can be used in many fields, such as telecommunication, sensor, medical, and quantum computing. This section will discuss some of the use cases of photonic processors.

1) *Telecommunication:* Telecommunication is among the most significant driving forces behind the demand for PICs. Therefore, it is no surprise that they are also a significant potential market for photonic processors. Indeed, photonic processors can be used to create transceivers for high-speed communication and route optical signals in data centers and field deployments. This makes them an ideal candidate for fiber deployments and 5G networks. Additionally, their strong capabilities in processing RF signals can be used to create 5G base stations for beamforming and signal processing.

2) *Sensors:* Fiber sensors have been used in many industries, including gas and oil, aerospace, and the medical sector. They are used to measure temperature, pressure, strain, and other physical quantities. Photonic processors can be used to create the base stations for these sensors, being responsible for signal production, processing, and data analysis.

3) *Medical:* The medical industry has been using optical fibers for a long time, and photonic processors can be used as processing units for these fibers. They can also be included in the RF processing chains of equipment like MRI machines.

All of these factors combine place photonic processors in a unique way where they can be integrated into many existing technologies and fields, making them a promising technology.

III. PHÔS

This paper will present the PHÔS programming language, beginning with the reasoning behind its creation, then its core concepts, and finally, its syntax and semantics.

A. Motivation

Many programming languages exist, but none meet the unique requirements of describing photonic circuits. They all lack one of the desirable features for photonic design. Indeed, photonic designs have some unique challenges due to the very nature of light:

- Waveguides can carry two modes, one in either direction
- The light can have a broad spectrum
- Light is phase-sensitive
- Signals modulated on the light are delay sensitive

Additionally, photonic circuits have some unique requirements:

- Devices should behave ideally
- The design should be reconfigurable, tunable, and programmable
- The design should be platform independent

These challenges and requirements make it difficult for existing languages to describe photonic circuits intuitively and efficiently. Therefore, a new language is needed to describe photonic circuits, this language, PHÔS, is presented in this paper.

B. Core concepts

PHÔS is a language based on traditional imperative programming, with features from functional and dataflow programming languages. It is a strongly typed language with a type system inspired by *Rust*'s and a syntax inspired by *Rust*'s, *Python*'s, and *Elixir*'s. PHÔS separates hardware synthesis into three stages: compilation, evaluation, and synthesis. During compilation, the user's code is turned into a bytecode representing their code, which contains information about the program's structure, types, functions, and signal flow. The bytecode is executed during evaluation, and the program's output is computed. Parts of the program that depend on tunable values are collected for further processing. The output of the evaluation stage is a graph representing the program's signal flow. This graph is then used to synthesize the program into a photonic circuit. Some of the core concepts will be discussed in the following few sections.

1) *Constraints:* Constraints are placed on values and signals to inform PHÔS of what values or content is expected at any point in the program. These constraints can then be used for validation of the design and to optimize the design. Con-

straints are also used when simulating the circuit quickly and efficiently simulate the circuit.

2) *Tunable values:* Like any other variables or argument, tunable values are declared in the source code. Only when the user instantiates the module are the tunable values configured; this occurs during the evaluation stage. When this happens, as the user's program is being evaluated, all operations that cannot be performed due to missing values are collected into stacks for further processing.

3) *Reconfigurability through branching:* PHÔS handles the reconfigurability of the circuit it produces through branching in its source code; when tunable values are present, these branches represent the boundaries of reconfigurability regions. PHÔS discards the branches that are not needed based on the constraints of the tunable values. These constraints allow PHÔS to produce a reconfigurable and tunable circuit while avoiding unnecessary work of unreachable states.

4) *Intrinsic operations:* PHÔS decomposes the user's program during evaluation into a series of intrinsic operations. These operations are the unit operations performed on the signal in the circuit. These operations are then used to synthesize the circuit.

C. Syntax and semantics

As previously mentioned, PHÔS follows a syntax familiar to many programmers, as it takes inspiration from *Python* and other *C*-like languages. This section will present the syntax and semantics of PHÔS. In Listing 1, the classic "Hello, world!" program is presented in PHÔS. This program is composed of a single function printing the classic message.

```
fn hello_world() {
    print("Hello, world!");
}
```

Listing 1: "Hello, world!" in PHÔS.

1) *Synthesizable blocks:* PHÔS makes a semantic difference between functions and synthesizable blocks. Synthesizable blocks are similar to functions but can take in and return signals. This distinction encourages the user to separate the concerns between their parameters' computation and the circuit's signal flow they are trying to build. In Listing 2, such a synthesizable block performs filtering on an input signal. It also shows some of the unique features of PHÔS, such as using the pipe operator (`|>`) to chain operations and the ability to express SI units directly in the code.

```
syn my_circuit(input: optical) -> optical {
    optical |> filter(1550 nm, bandwidth = 10 GHz)
}
```

Listing 2: Synthesizable block in PHÔS.

2) *Constraints:*

Constraints are expressed in PHÔS using decorators on input arguments and output values. This syntax may be changed in the future, but as of the writing of this paper, it is the syntax used. In Listing 3, the `@power` decorator represents that the signal has specific power content.

```
syn my_circuit(
    @power(0 dBm) input: optical
) -> optical {
    optical |> filter(1550 nm, bandwidth = 10 GHz)
}
```

Listing 3: Synthesizable block with constraints in PHÔS.

IV. 16-QAM 400Gb/s transmitter

After this short introduction to the PHÔS language, this section will present a 16-QAM 400Gb/s transmitter designed using PHÔS. This example is based on the work by *Talkhooncheh, Arian Hashemi, et al.* [8]. The code of this example can be found in Listing 4, showing the circuit as taking four electrical inputs, which are the four binary sources that will be modulated. It also takes an optical signal, which is the source laser. It then splits the laser source into four signals, one for each binary source. These signals are then modulated using amplitude modulation. Their phase is then constrained to be within 90° of each other. Finally, the four signals are merged back into a single signal.

```
syn coherent_transmitter(
    input: optical,
    (a, b, c, d): (
        electrical,
        electrical,
        electrical,
        electrical
    ),
) -> optical {
    input
        |> split((1.0, 1.0, 0.5, 0.5))
        |> zip((a, c, b, d))
        |> modulate(type = Modulation::Amplitude)
        |> constrain(d_phase = 90°)
        |> merge()
}
```

Listing 4: 16-QAM 400Gb/s transmitter designed using PHÔS.

This circuit can be simulated using the simulator that is built into PHÔS, called the constraint-solver. It uses the constraints and intrinsic operations of the user's circuit to quickly simulate the circuit. In this case, on a modern *AMD* CPU, the circuit was simulated for 10ns with a timestep of 10ps in only 750ms. The result of this simulation is the constellation shown in Figure 3, showing the circuit indeed performs as expected.

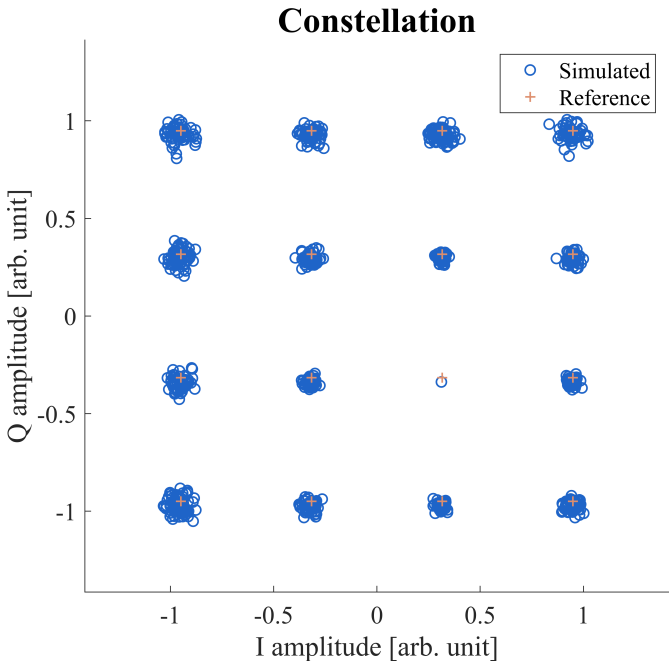


Figure 3: Constellation diagram of the 16-QAM 400Gb/s transmitter.

V. Conclusion

This paper demonstrated a novel way of describing photonic circuits using code. Moreover, while the language is not fully implemented yet, it shows promising results, can easily express complex systems, and performs fast yet accurate simulations.

A. *Future work*

More work is required to complete the PHÔS ecosystem, starting with the complete implementation of the language, research into provers for the constraint system, and the development of place-and-route algorithms for the circuit synthesis.

References

[1] W. Bogaerts, M. Fiers, M. Sivilotti, and P. Dumon, "The IPKISS photonic design framework," in *Opt. Fiber Communication Conf.*, Anaheim, California, 2016, p. 1, doi: 10.1364/OFC.2016.W1E.1. Accessed: Mar. 10, 2023. [Online]. Available: https://opg.optica.org/abstract.cfm?URI=OFC-2016-W1E.1

[2] W. Bogaerts, and A. Rahim, "Programmable Photonics: An Opportunity for an Accessible Large-Volume PIC Ecosystem," *IEEE J. Sel. Topics Quantum Electronics*, vol. 26, no. 5, pp. 1–17, Sep. 2020, doi: 10.1109/JSTQE.2020.2982980. Accessed: Mar. 10, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9049105/

[3] W. Bogaerts, and L. Chrostowski, "Silicon Photonics Circuit Design: Methods, Tools and Challenges," *Laser & Photon. Reviews*, vol. 12, no. 4, p. 1700237, Apr. 2018, doi: 10.1002/lpor.201700237. Accessed: Mar. 27, 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/lpor.201700237

[4] D. Pérez-López, A. López, P. DasMahapatra, and J. Capmany, "Multipurpose self-configuration of programmable photonic circuits," *Nature Commun.*, vol. 11, no. 1, p. 6359, Dec. 2020, doi: 10.1038/s41467-020-19608-w. Accessed: Mar. 10, 2023. [Online]. Available: https://www.nature.com/articles/s41467-020-19608-w

[5] F. V. Kerchove, X. Chen, D. Colle, W. Bogaerts, and M. Pickavet, "Adapting Routing Algorithms to Programmable Photonic Circuits," in *2022 Eur. Conf. Opt. Communication (Ecoc)*, Sep. 2022, pp. 1–4.

[6] Z. Gao, X. Chen, et al., "Automatic Realization of Light Processing Functions for Programmable Photonics," in *2022 IEEE Photon. Conf. (Ipc)*, Nov. 2022, pp. 1–2, doi: 10.1109/IPC53466.2022.9975757.

[7] F. V. Kerchove, X. Chen, et al., "An Automated Router with Optical Resource Adaptation," *J. Lightw. Technol.*, pp. 1–13, 2023, doi: 10.1109/JLT.2023.3275385.

[8] A. H. Talkhooncheh, A. Zilkie, G. Yu, R. Shafiiha, and A. Emami, "A 200Gb/s QAM-16 Silicon Photonic Transmitter with 4 Binary-Driven EAMs in An MZI Structure," in *2023 Opt. Fiber Commun. Conf. Exhib. (Ofc)*, Mar. 2023, pp. 1–3, doi: 10.1364/OFC.2023.M1E.6.