GHENT
UNIVERSITY

FACULTY OF ENGINEERING
AND ARCHITECTURE

Department of Information Technology
Photonics Research Group

A software language approach for describing and programming photonics hardware

Master's thesis defence - Sébastien d'Herbais de Thun - 29th of June 2023
Promoters: Prof. dr. ir. Wim Bogaerts, Prof. dr. ir. Dirk Stroobandt

GHENT
UNIVERSITY

About this presentation

- Introduction
- Elevator pitch
- Programmatic description: an overview
- Example: 16-QAM modulator
- Example: Lattice filter
- Conclusion
- Future work

GHENT UNIVERSITY

To code, or not to code

- Not everybody is a programmer

To code, or not to code

- Not everybody is a programmer **and that's okay!**
  - Code sections will be kept **short**
  - The language is **familiar**
  - Code will be **explained**
  - Code is shown in **boxes**

Python

```python
print('Hello, world!')
```

PHÔS

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

To code, or not to code

- Not everybody is a programmer **and that's okay!**
  - Code sections will be kept **short**
  - The language is **familiar**
  - Code will be **explained**
  - Code is shown in **boxes**
- Code is **non-exhaustive**

Python

```python
print('Hello, world!')
```

PHÔS

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

To code, or not to code

- Not everybody is a programmer **and that's okay!**
  - Code sections will be kept **short**
  - The language is **familiar**
  - Code will be **explained**
  - Code is shown in **boxes**
- Code is **non-exhaustive**
- Code is **not optimized**

Python

```python
print('Hello, world!')
```

PHÔS

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

GHENT UNIVERSITY

To code, or not to code

- Not everybody is a programmer **and that's okay!**
  - Code sections will be kept **short**
  - The language is **familiar**
  - Code will be **explained**
  - Code is shown in **boxes**
- Code is **non-exhaustive**
- Code is **not optimized**
- Code is **illustrative**

Python

```python
print('Hello, world!')
```

PHÔS

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

The elevator pitch

GHENT
UNIVERSITY

Levels of abstraction

- Currently low

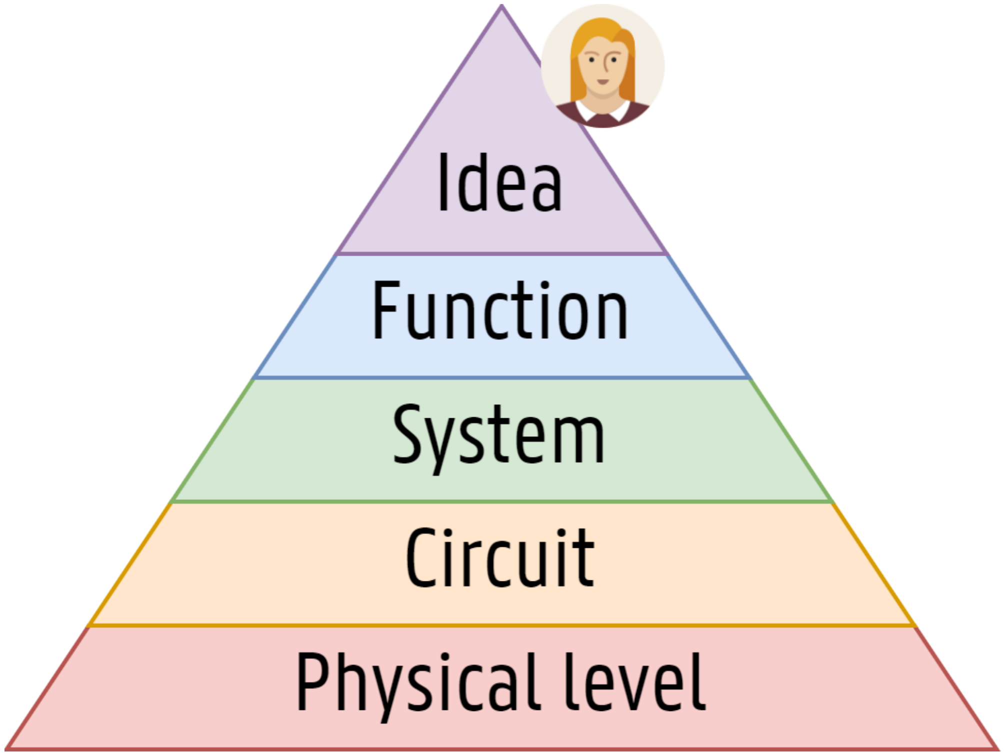

Figure 2 | Levels of abstraction in photonic circuit design.

Levels of abstraction

- Currently low
- We want to go higher

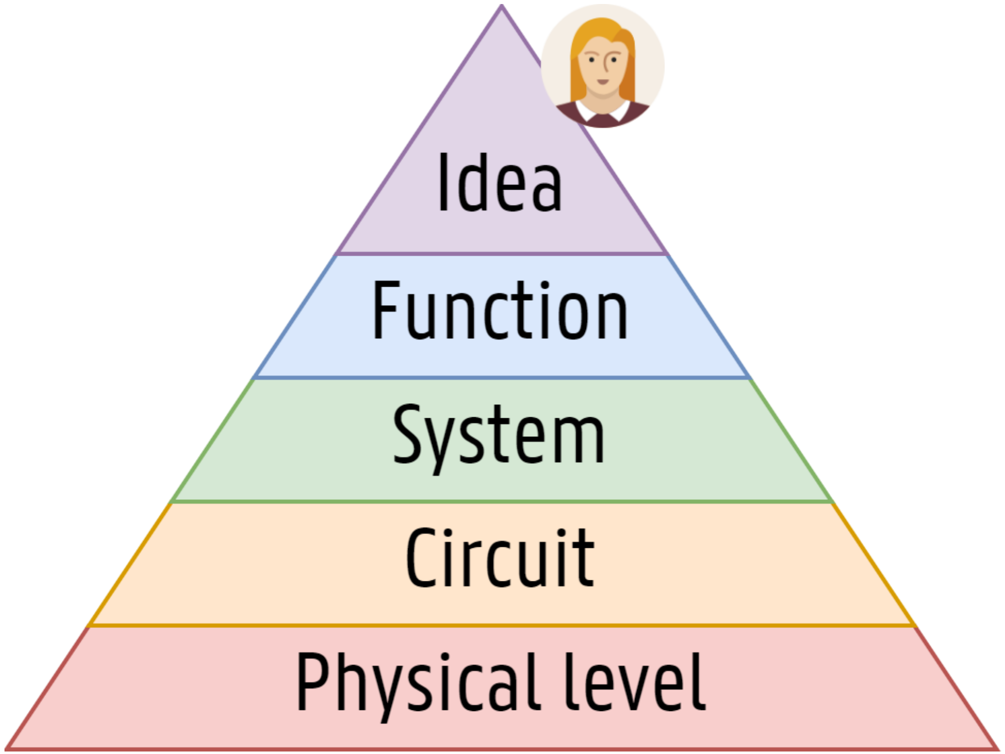

FIGURE 2 | Levels of abstraction in photonic circuit design.

Levels of abstraction

- Currently low
- We want to go higher
- We want to go **much** higher

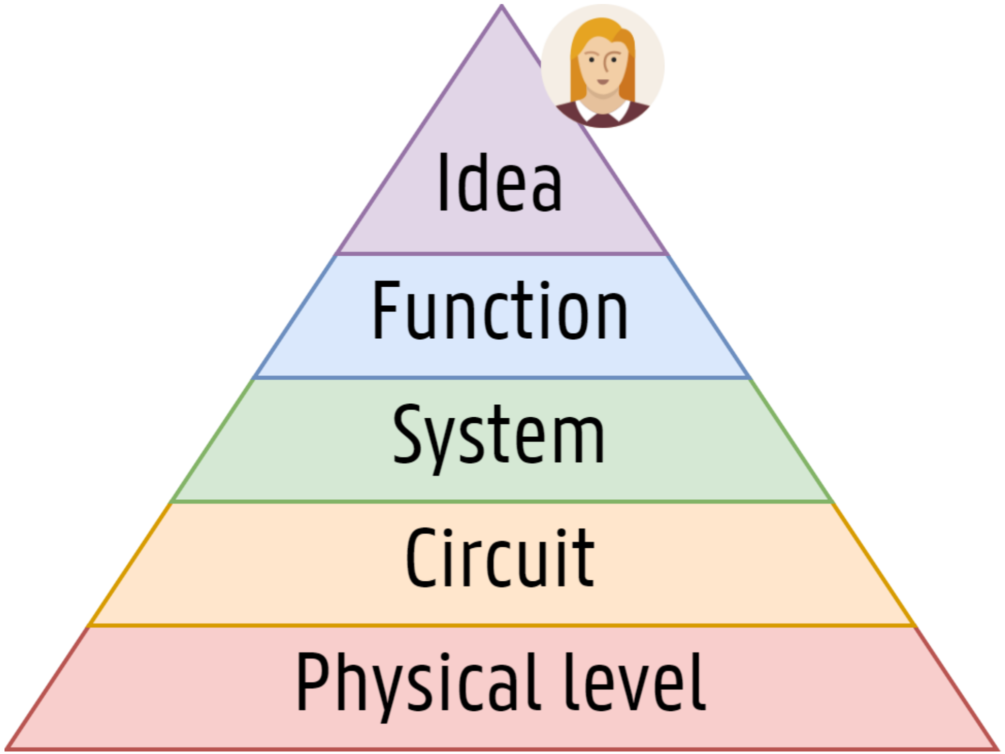

FIGURE 2 | Levels of abstraction in photonic circuit design.

GHENT UNIVERSITY

Levels of abstraction

- Currently low
- We want to go higher
- We want to go **much** higher
- We need to build abstractions
  - Components (parametric)

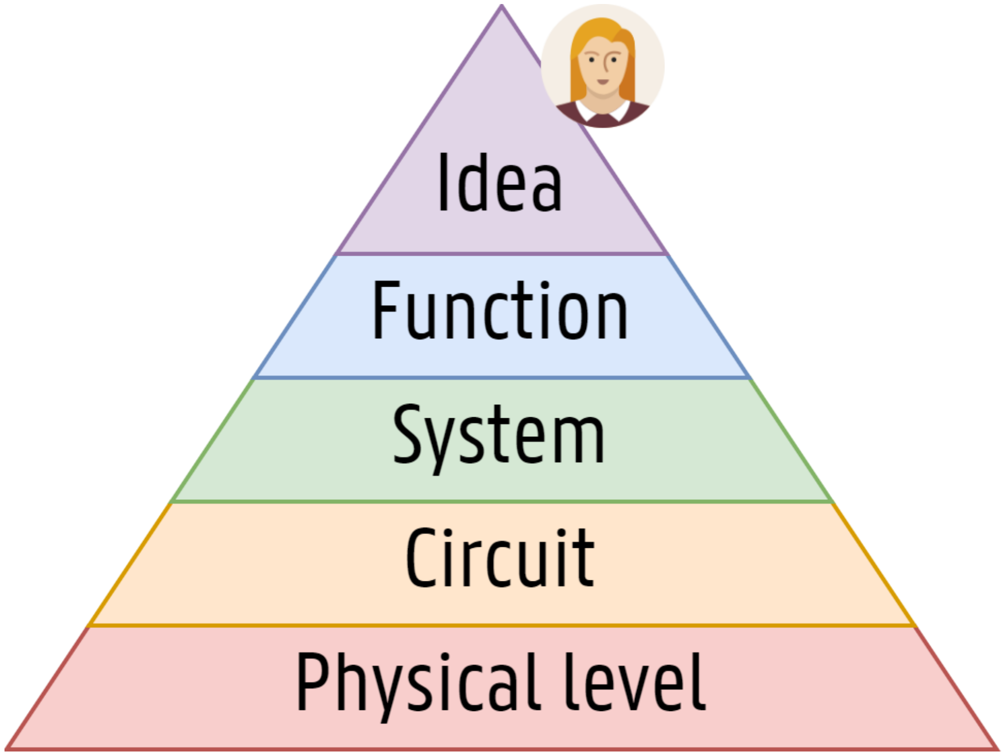

Figure 2 | Levels of abstraction in photonic circuit design.

Levels of abstraction

- Currently low
- We want to go higher
- We want to go **much** higher
- We need to build abstractions
  - Components (parametric)
  - Signal flow graphs

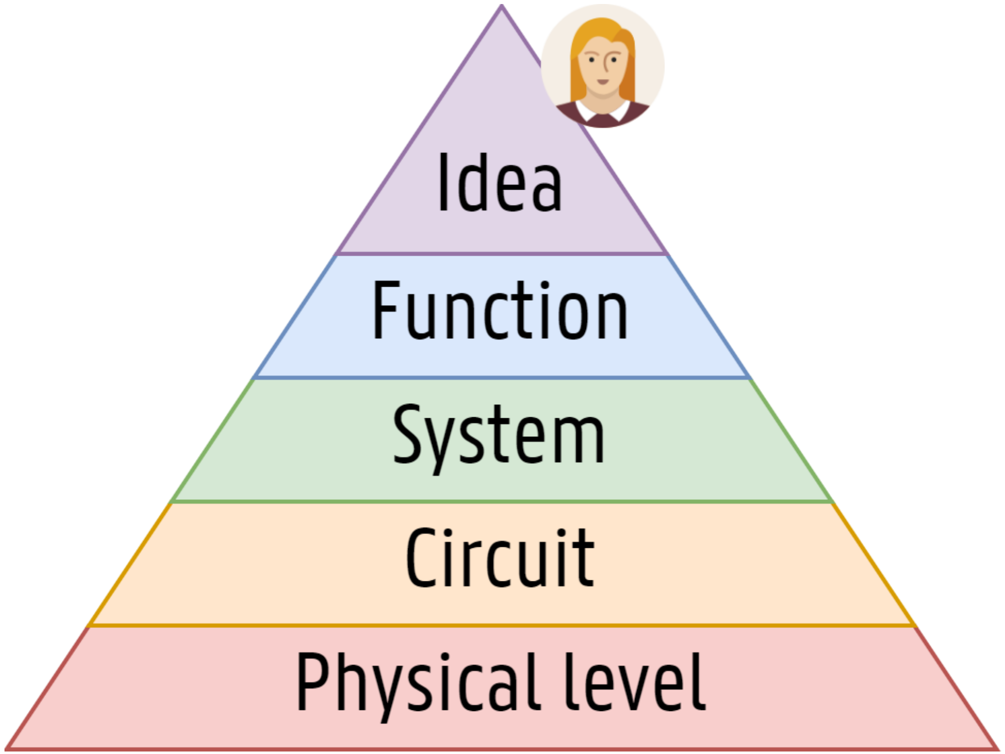

FIGURE 2 | Levels of abstraction in photonic circuit design.

GHENT UNIVERSITY

Levels of abstraction

- Currently low
- We want to go higher
- We want to go **much** higher
- We need to build abstractions
  - Components (parametric)
  - Signal flow graphs
  - ???

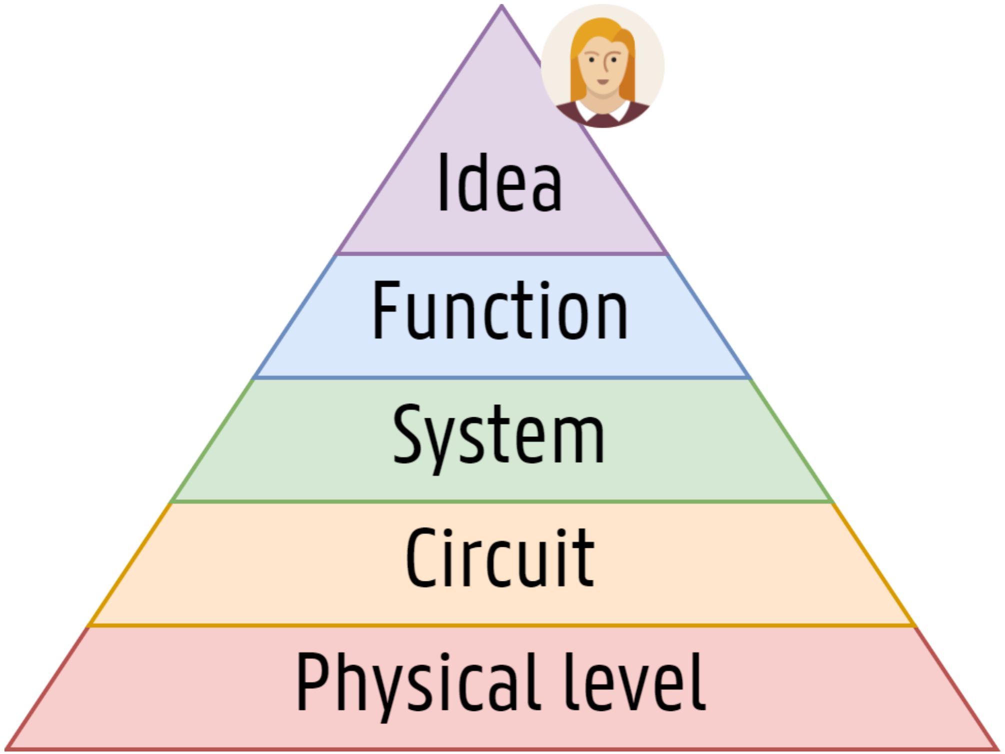

FIGURE 2 | Levels of abstraction in photonic circuit design.

GHENT UNIVERSITY

Introducing PHÔS

- PHÔS is a **domain-specific language**

GHENT UNIVERSITY

Introducing PHÔS

- PHÔS is a **domain-specific language**
- PHÔS describes **photonic circuits**

Introducing PHÔS

- PHÔS is a **domain-specific language**
- PHÔS describes **photonic circuits**
- PHÔS is **declarative**

Introducing PHÔS

- PHÔS is a **domain-specific language**
- PHÔS describes **photonic circuits**
- PHÔS is **declarative**
- PHÔS is **parametric**

GHENT UNIVERSITY

Introducing PHÔS

- PHÔS is a **domain-specific language**
- PHÔS describes **photonic circuits**
- PHÔS is **declarative**
- PHÔS is **parametric**
- PHÔS is **expressive**

- PHÔS is the **function** and **system** levels
  - Filter synthesis
  - Signal flow graph generation
  - Component instantiation
  - Reconfigurability & tunability
  - Optimization

Introducing PHÔS

- PHÔS is a **domain-specific language**
- PHÔS describes **photonic circuits**
- PHÔS is **declarative**
- PHÔS is **parametric**
- PHÔS is **expressive**
- PHÔS is **extensible**

- PHÔS is the **function** and **system** levels
  - Filter synthesis
  - Signal flow graph generation
  - Component instantiation
  - Reconfigurability & tunability
  - Optimization
- PHÔS is **not** at the component level
  - ~~Component design~~
  - ~~Component simulation~~
  - ~~Component optimization~~

Programmatic description: an overview

GHENT UNIVERSITY

Translation of intent

- How do we tell the computer what we want?
- What do we want the computer to do for us?
- How does the computer do it?

Translation of intent

- How do we tell the computer what we want? **Programming!**
- What do we want the computer to do for us?
- How does the computer do it?

Translation of intent

- How do we tell the computer what we want?
- What do we want the computer to do for us?
- How does the computer do it?

Translation of intent

- How do we tell the computer what we want?
- What do we want the computer to do for us? **As much as possible!**
- How does the computer do it?

Translation of intent

- How do we tell the computer what we want?
- What do we want the computer to do for us?
- How does the computer do it? **Compilation, Evaluation, and Synthesis!**

GHENT UNIVERSITY

Why programming?

- Scaling circuits is **really** hard
- Scaling code is **really** easy

Why programming?

- Scaling circuits is **really** hard
- Circuits are **inflexible**

- Scaling code is **really** easy
- Code is **flexible**

Why programming?

- Scaling circuits is **really** hard
- Circuits are **inflexible**
- Circuits are not **reusable**

- Scaling code is **really** easy
- Code is **flexible**
- Code is easily **reusable**

Why programming?

- Scaling circuits is **really** hard
- Circuits are **inflexible**
- Circuits are not **reusable**
- Circuits are not **expressive**

- Scaling code is **really** easy
- Code is **flexible**
- Code is easily **reusable**
- Code is **expressive**

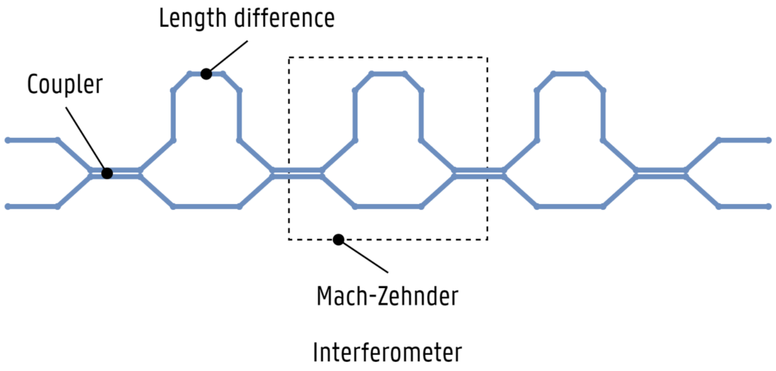

FIGURE 1 | A lattice filter circuit.

PHÔS

```
filter_kind_coefficients(filter_kind)
  |> fold((a, b), |acc, (coeff, phase)| {
    acc |> coupler(coeff)
        |> constrain(d_phase = phase)
  })
```

LISTING 1 | A lattice filter as code.

Yes, but why a new language?

- Existing languages **do not** works for photonics
  - Hardware description languages: ~~VHDL~~, ~~MyHDL~~
  - High-level synthesis languages: ~~SystemC~~
  - Analog modeling languages: ~~Verilog-AMS~~, ~~SPICE~~
  - Traditional programming languages: ~~Python~~, ~~Rust~~

Yes, but why a new language?

- Existing languages **do not** works for photonics
  - Hardware description languages: ~~VHDL~~, ~~MyHDL~~
  - High-level synthesis languages: ~~SystemC~~
  - Analog modeling languages: ~~Verilog-AMS~~, ~~SPICE~~
  - Traditional programming languages: ~~Python~~, ~~Rust~~
- Libraries are **not expressive** enough

Yes, but why a new language?

- Existing languages **do not** works for photonics
  - Hardware description languages: ~~VHDL~~, ~~MyHDL~~
  - High-level synthesis languages: ~~SystemC~~
  - Analog modeling languages: ~~Verilog-AMS~~, ~~SPICE~~
  - Traditional programming languages: ~~Python~~, ~~Rust~~
- Libraries are **not expressive** enough
- Why? **Because photonics is different**
  - ~~Sequential~~ Continuous
  - ~~Digital~~ Analog

Yes, but why a new language?

- Existing languages **do not** works for photonics
  - Hardware description languages: ~~VHDL~~, ~~MyHDL~~
  - High-level synthesis languages: ~~SystemC~~
  - Analog modeling languages: ~~Verilog-AMS~~, ~~SPICE~~
  - Traditional programming languages: ~~Python~~, ~~Rust~~
- Libraries are **not expressive** enough
- Why? **Because photonics is different**
  - ~~Sequential~~ Continuous
  - ~~Digital~~ Analog
- We need a **domain-specific language**

GHENT UNIVERSITY

What do we want the computer to do for us?

- **Ideal behaviour**: feedback loops, calibration

What do we want the computer to do for us?

- **Ideal behaviour**: feedback loops, calibration
- **Simulation**: simulator, interface with existing ones

What do we want the computer to do for us?

- **Ideal behaviour**: feedback loops, calibration
- **Simulation**: simulator, interface with existing ones
- **Platform independence**: process, foundry, processor architecture

What do we want the computer to do for us?

- **Ideal behaviour**: feedback loops, calibration
- **Simulation**: simulator, interface with existing ones
- **Platform independence**: process, foundry, processor architecture
- **Visualization**: signal flow graphs, circuit diagrams

What do we want the computer to do for us?

- **Ideal behaviour**: feedback loops, calibration
- **Simulation**: simulator, interface with existing ones
- **Platform independence**: process, foundry, processor architecture
- **Visualization**: signal flow graphs, circuit diagrams
- **Reconfigurability**: reconfigurability through branching

What do we want the computer to do for us?

- **Ideal behaviour**: feedback loops, calibration
- **Simulation**: simulator, interface with existing ones
- **Platform independence**: process, foundry, processor architecture
- **Visualization**: signal flow graphs, circuit diagrams
- **Reconfigurability**: reconfigurability through branching
- **Tunability**: implicit tunability

What do we want the computer to do for us?

- **Ideal behaviour**: feedback loops, calibration
- **Simulation**: simulator, interface with existing ones
- **Platform independence**: process, foundry, processor architecture
- **Visualization**: signal flow graphs, circuit diagrams
- **Reconfigurability**: reconfigurability through branching
- **Tunability**: implicit tunability
- **Programmability**: hardware abstraction layer (HAL)

Reconfigurability and Tunability

PHÔS

```
syn my_circuit(
    input: optical,
    gain: Gain
) -> optical {
    if gain > 0 dB {
        input |> amplifier(gain)
    } else {
        input
    }
}
```

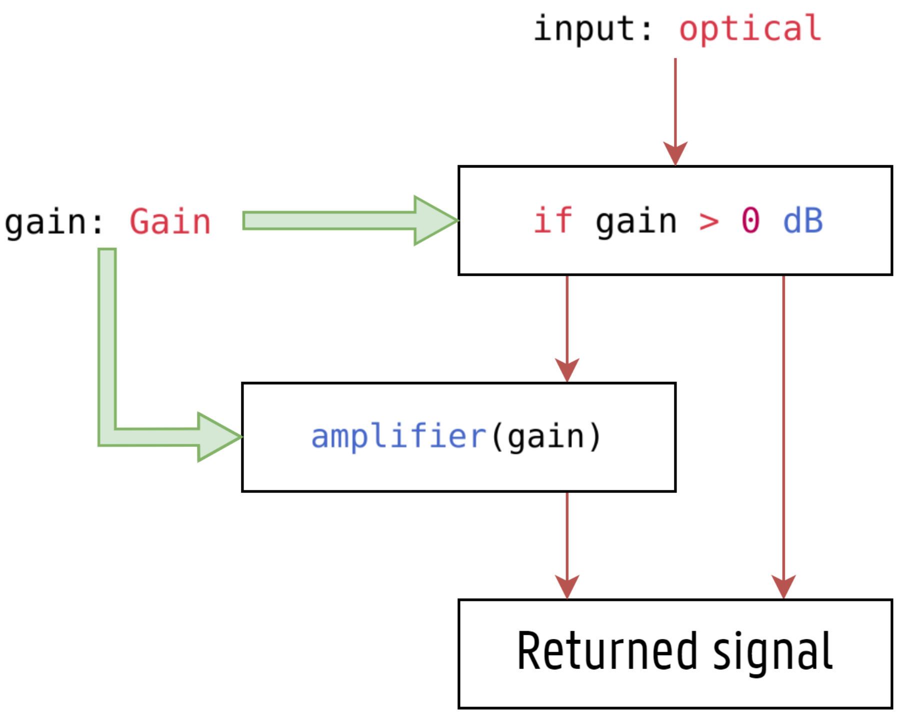

FIGURE 2 Signal flow diagram of `my_circuit`, showing the tunable value impacting reconfigurability.

Tying it all together

- Express **constraints** on the signals and values

Tying it all together

- Express **constraints** on the signals and values
- Used for **verification** and **optimization**

Tying it all together

- Express **constraints** on the signals and values
- Used for **verification** and **optimization**
- Used to **reduce** reconfigurability space

Tying it all together

- Express **constraints** on the signals and values
- Used for **verification** and **optimization**
- Used to **reduce** reconfigurability space
- Used to **simulate** the circuit

Tying it all together

- Express **constraints** on the signals and values
- Used for **verification** and **optimization**
- Used to **reduce** reconfigurability space
- Used to **simulate** the circuit

```
syn amplifier(
    @power(max(0 dBm - gain))
    input: optical,

    @max(10 dB)
    gain: Gain,
) -> @power(input + gain) optical {
    ...
}
```

What is a circuit made of?

- **Filters**

What is a circuit made of?

- **Filters**
- **Gain** and **loss** elements

What is a circuit made of?

- **Filters**
- **Gain** and **loss** elements
- **Modulators** and **detectors**

What is a circuit made of?

- **Filters**
- **Gain** and **loss** elements
- **Modulators** and **detectors**
- **Splitters**, **combiners**, and **couplers**

What is a circuit made of?

- **Filters**
- **Gain** and **loss** elements
- **Modulators** and **detectors**
- **Splitters**, **combiners**, and **couplers**
- **Switches**

What is a circuit made of?

- **Filters**
- **Gain** and **loss** elements
- **Modulators** and **detectors**
- **Splitters**, **combiners**, and **couplers**
- **Switches**
- **Phase shifters** and **delay lines**

What is a circuit made of?

- **Filters**
- **Gain** and **loss** elements
- **Modulators** and **detectors**
- **Splitters**, **combiners**, and **couplers**
- **Switches**
- **Phase shifters** and **delay lines**
- **Sources**, **sinks**, and **empty** signal

What is a circuit made of?

- **Filters**
- **Gain** and **loss** elements
- **Modulators** and **detectors**
- **Splitters**, **combiners**, and **couplers**
- **Switches**
- **Phase shifters** and **delay lines**
- **Sources**, **sinks**, and **empty** signal
- Together, these form the **intrinsic operations**

What is a circuit made of?

- **Filters**
- **Gain** and **loss** elements
- **Modulators** and **detectors**
- **Splitters**, **combiners**, and **couplers**
- **Switches**
- **Phase shifters** and **delay lines**
- **Sources**, **sinks**, and **empty** signal
- Together, these form the **intrinsic operations**
- **Circuits** are made of **intrinsic operations**

What is a circuit made of?

- **Filters**
- **Gain** and **loss** elements
- **Modulators** and **detectors**
- **Splitters**, **combiners**, and **couplers**
- **Switches**
- **Phase shifters** and **delay lines**
- **Sources**, **sinks**, and **empty** signal
- Together, these form the **intrinsic operations**
- **Circuits** are made of **intrinsic operations**

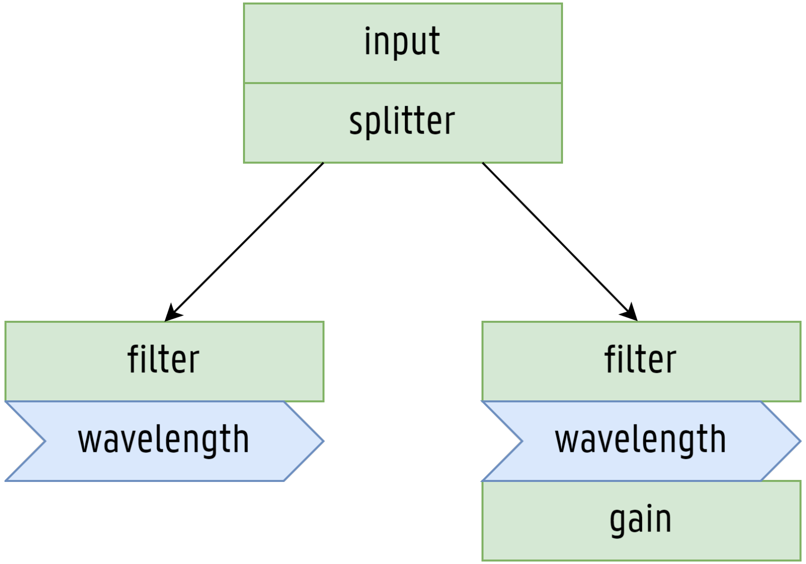

FIGURE 12 | A lattice filter circuit.

Overview

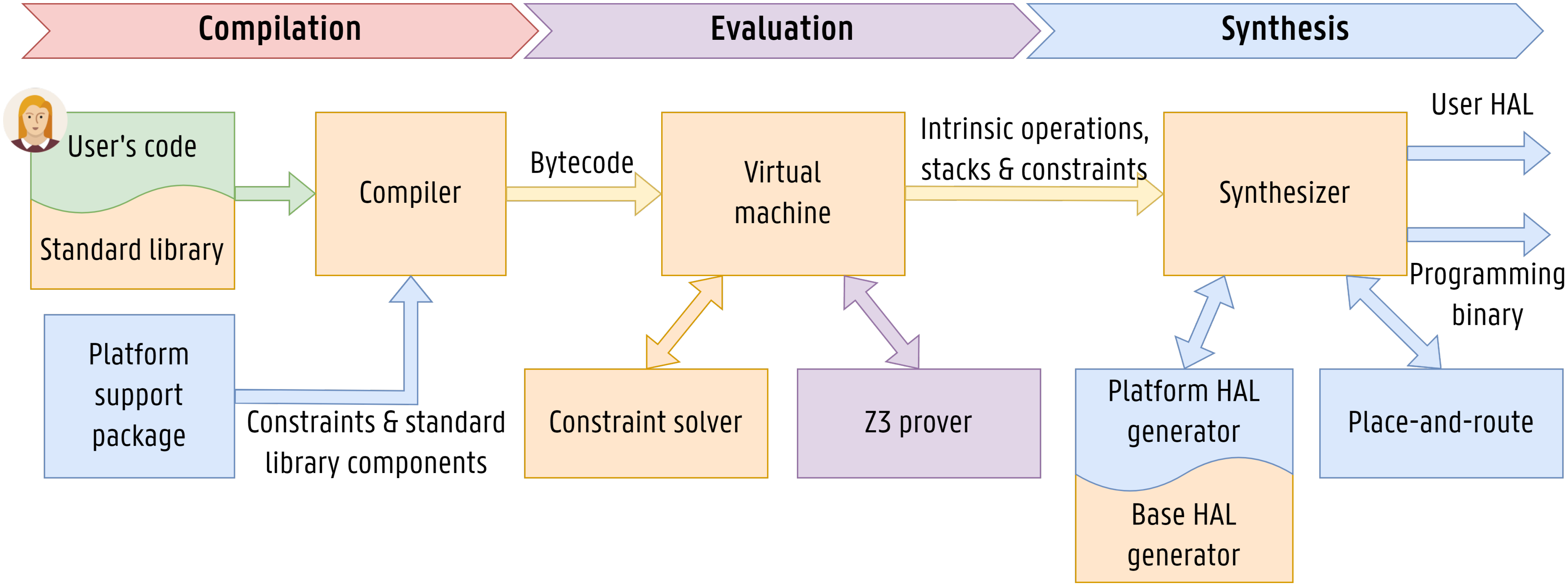

FIGURE 3 Synthesis stages in PHÔS: compilation, evaluation, and synthesis. Shows each step and the corresponding output. The colours describe the responsibility of maintaining each element.

Examples

GHENT UNIVERSITY

16-QAM 400 Gb/s modulator

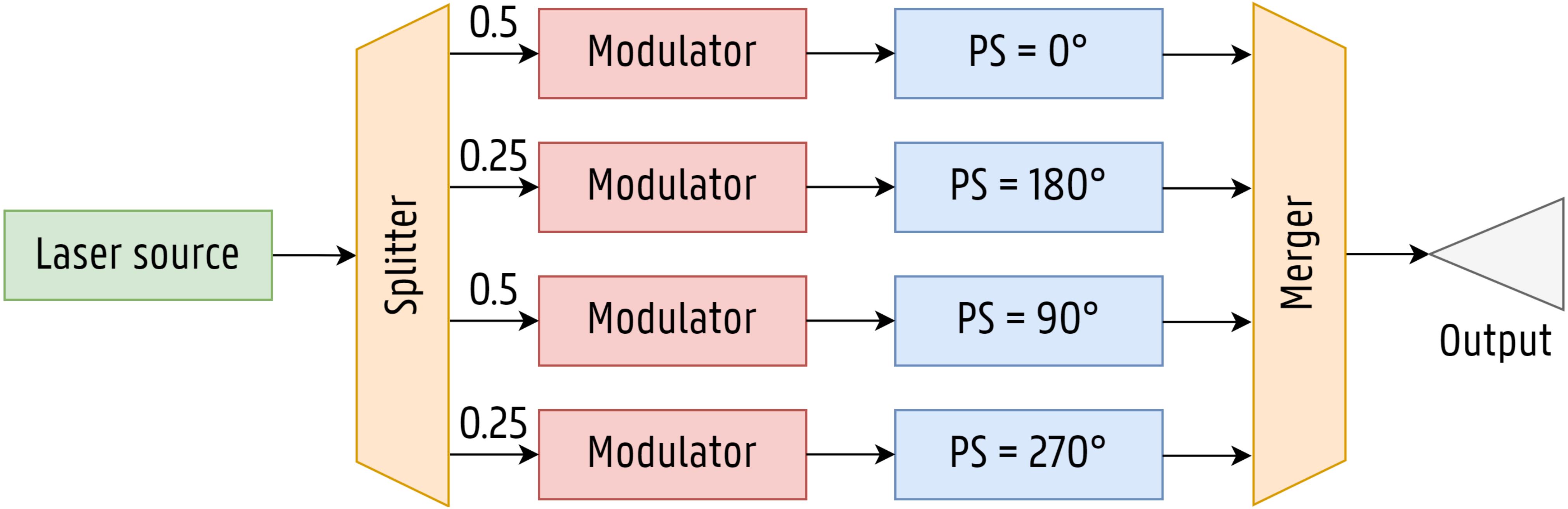

FIGURE 4 | 16-QAM modulator circuit diagram.

16-QAM 400 Gb/s modulator (cont.)

PHÔS

```
syn coherent_transmitter(
        input: optical,
        [a, b, c, d]: [electrical; 4],
) -> optical {
        input
            |> split((1.0, 1.0, 0.5, 0.5))
            |> zip((a, c, b, d))
            |> modulate(Modulation::Amplitude)
            |> constrain(d_phase = 90°)
            |> merge()
}
```

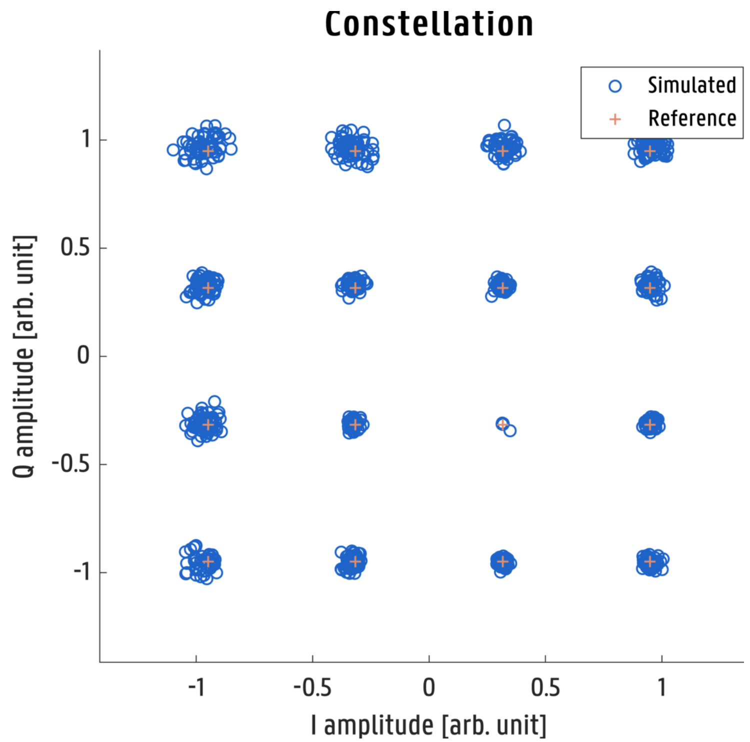

FIGURE 5 | QAM constellation diagram of the modulated output.

GHENT UNIVERSITY

Lattice filter

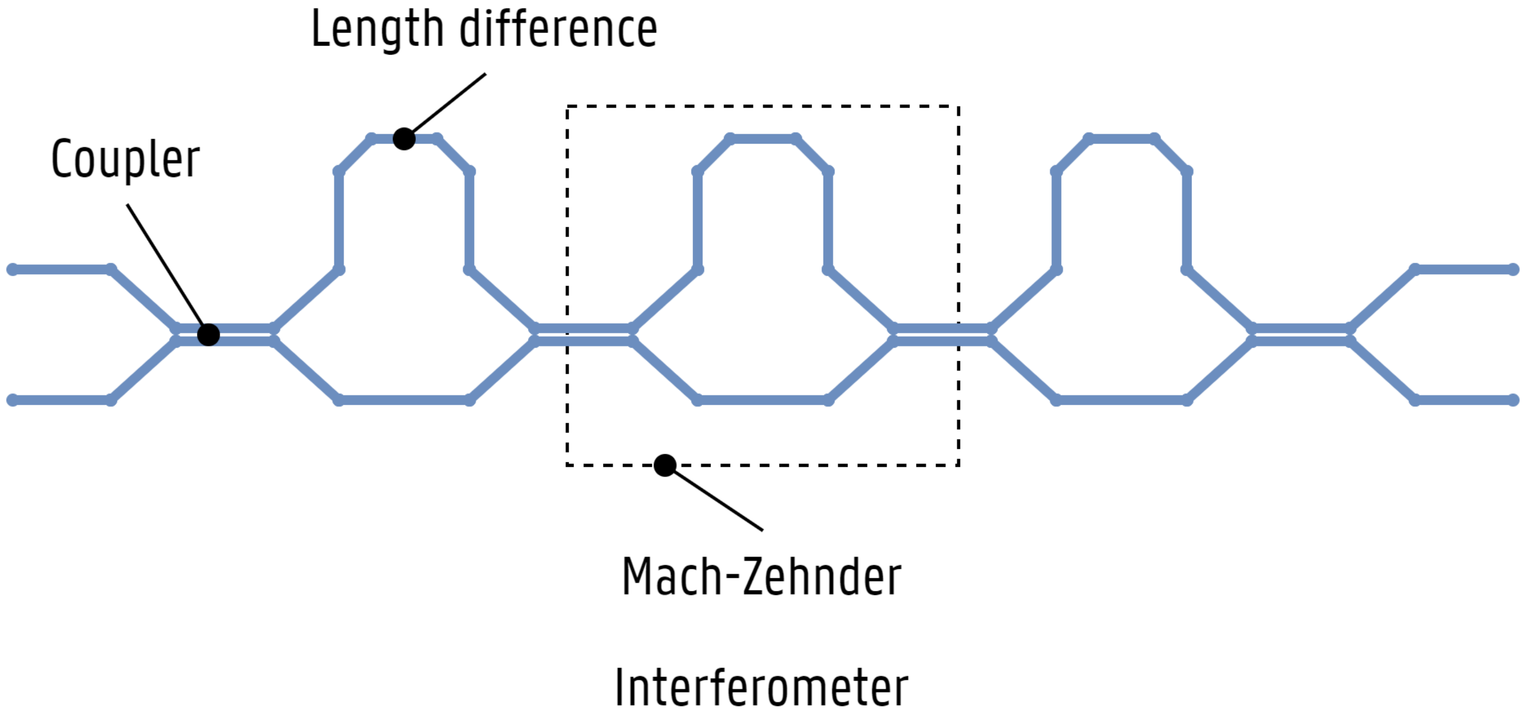

Figure 6 | Lattice filter circuit diagram.

GHENT UNIVERSITY

Lattice filter (cont.)

PHÔS

```
syn lattice_filter(
    a: optical,
    b: optical,
    filter_kind: FilterKind
) -> (optical, optical) {
    filter_kind_coefficients(filter_kind)
      |> fold((a, b), |acc, (coeff, phase)| {
        acc |> coupler(coeff)
            |> constrain(d_phase = phase)
      })
}
```

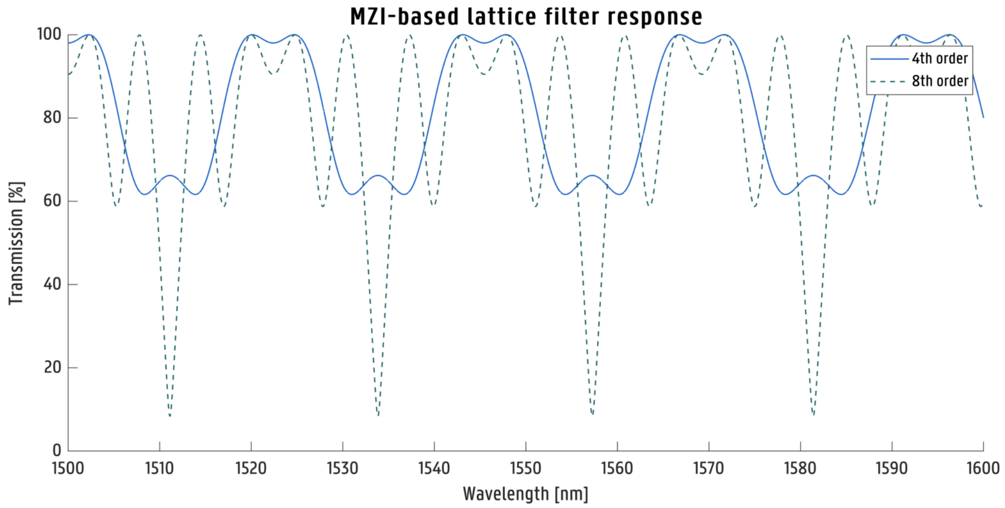

FIGURE 6 | Lattice filter frequency response.

Conclusion

GHENT UNIVERSITY

Future works

- Implementing PHÔS fully
- Co-simulation with digital and analog circuits
- Place-and-route
- Language improvements
- Advanced constraint inference

Key takeaways

- Novel programmatic way of describing photonics:
  - Expressive
  - Flexible
  - Reusable
  - Programmable
  - Opens the way to VLSI for photonics

Key takeaways

- Novel programmatic way of describing photonics:
  - Expressive
  - Flexible
  - Reusable
  - Programmable
  - Opens the way to VLSI for photonics
- Novel constraint system for photonics:
  - Optimization
  - Verification
  - Simulation

THANK YOU FOR LISTENING

Sébastien d'Herbais de Thun

Student

Universiteit Gent
@ugent
@ugent
Ghent University
Sébastien d'Herbais de Thun

PHOTONICS RESEARCH GROUP

E sebastien.dherbaisdethun@ugent.be
M +32 (0) 473 12 86 57

www.ugent.be

Sources