Erasmus+

PSRS

Erasmus Mundus Joint Master degree
Photonics for Security Reliability and Safety

Master *Photonics for Security Reliability and Safety* (PSRS)

UNIVERSITÉ DE LYON

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

UNIVERSITY OF EASTERN FINLAND

UPEC UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE
Connaissance - Action

POLITECNICO DI TORINO

SILICON PHOTONICS CHIP FOR TELECOM APPLICATIONS

Master Thesis Report

Presented by

Alberto Otero Casado

and defended at

University Jean Monnet

22nd August 2023

Academic Supervisor: Dr. Carlo Ricciardi, Politecnico di Torino

Host Supervisor: Dr. Paolo Bardella, Politecnico di Torino

Jury Committee: Dr. Carlo Ricciardi, Politecnico di Torino

Dr. Matthieu Roussey, University of Eastern Finland

Abstract

In recent years, telecommunication networks have experienced a drastic increase in traffic, which calls for the implementation of novel techniques for switching. With this aim, the design of a state-of-the-art 1 x 3 Wavelength Selective Switch using Mach-Zehnder Interferometers and Contra Directional Couplers is presented. The proposed device operates in the S+C+L bands, using 8 channels per band. To accomplish the device implementation, Silicon Photonics technologies will be employed. In the process towards its full realization, first, a novel architecture for the design of the physical layout will be developed. Within it, the band and channel treating of the signal are performed at specific physical blocks. The next step involves implementing the design using the OptoDesigner script language, producing the components that make up the device and placing connections among them. In summary, the Wavelength Selective Switch presented proposes an innovative approach and showcases the potential of advanced switching techniques for addressing the demands of modern telecommunication networks.

Table of contents

Table of figures

Acknowledgements

I am very grateful to the PSRS consortium and their professors for the valuable knowledge and the incredible opportunity that this master represents. I am also very grateful to professor Paolo Bardella for his priceless support and all the skills transmitted to me.

I would like as well to express my gratitude to my classmates during these two years, for the good times, experiences and all the valuable insights that I have learned in their company. Special thanks to Francisco Matos, Héctor Jair Morales, Fernando Engels and Md Imtiaz Sultan.

Tambien gracias a mi padre y a mi hermano por su apoyo desde siempre. Y por último muchas gracias a mi madre por su apoyo, cariño y comprensión durante toda mi vida y especialmente en estos últimos dos años de mi etapa educativa.

1 Motivation

The goal of this thesis is the design of the realization mask (GDSII) of a complete silicon photonics switch for new telecommunication networks. These switches present broad applications in networks and optical communications, high-throughput computing, and similar concepts may be also used in more diverse areas such as sensing. Key requirements are low power consumption, wide bandwidth, and reduced footprint [1,2]. Indeed, the amount of data passing through telecommunication networks has dramatically increased over the past few years, leading to the need for improved systems to handle all this information. To do so, bandwidth scalability plays a fundamental role and the throughput must increase: in this scenario, silicon photonics (SiPh) is an excellent candidate, due to its suitability the complementary metal-oxide-semiconductor (CMOS) and its low power consumption [2,3,4,5]. Furthermore, optical switches avoid the need to perform optical/electrical and electrical/optical conversions, which expand problems such as the reduction of the bandwidth, an increased demand for energy, further delays, and higher costs [3,6].

There are several technologies for the fabrication of waveguides and passive and active components for Photonic Integrated Circuits (PICs), which includes materials such as Silicon, silicon compounds such as Silicon Nitride (SiN) and Silicon Dioxide (SiO_2), and III-V semiconductors. In any case, Silicon is arguably the most prominent due to its lower cost, the wide availability and the high quality of foundries working with the technology. [7,8,9]. Furthermore, by using this technology there is also enough availability of Multi-Project Wafer (MPW) run options, which prove important for research purposes and small-scale productions. Some of these are located in Europe, and can be easily reached for the purpose of this project, such as CEA-LETI, ePIXfab and Europractice [9,10].

2 State-of-the-art

2.1 Silicon Photonics

Over the last decades, SiPh has emerged as a promising technology to develop PICs using CMOS processes [3]. It comprises the “generation, routing, modulation, processing and detection of light” [11]. It is being developed to satisfy the needs of the telecom industry, due to its high throughput, bandwidth, low cost, and the existence of already developed CMOS processes that are compatible with SiPh [2,3,10]. Firstly, SiPh was used for sensing applications but by the late 2000s, it started to be commercialized as integrated circuits (IC). In the following decade, new applications were developed in the form of PICs [10,12]. This has been possible due to a great development of the technology behind it, which has allowed a stronger integration of components over time and a higher integration level, as shown in Figure 1 [7,13].

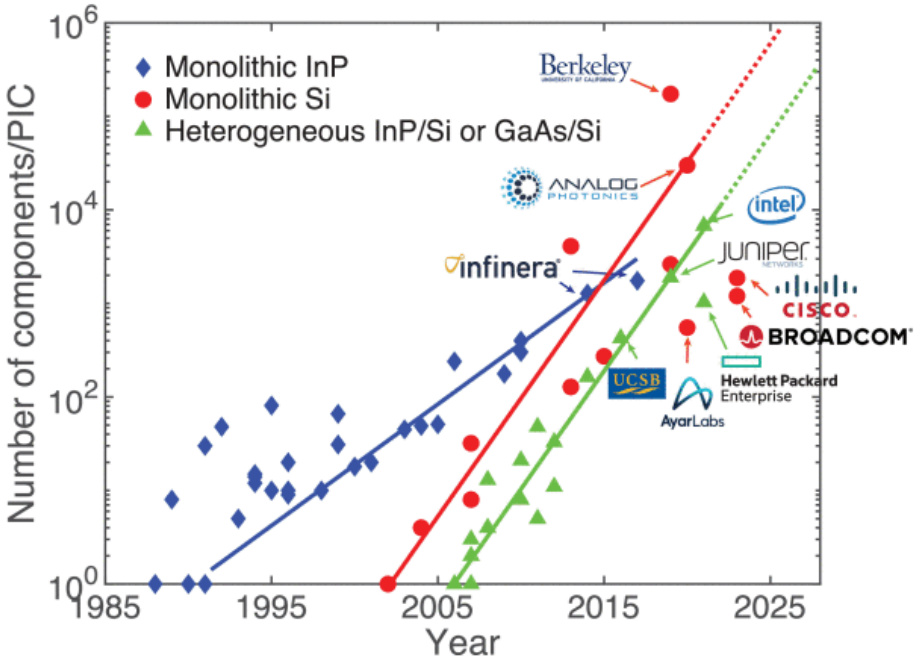


***Figure 1**. Evolution of the number of components within PICs [7].*

Although photonics integration presents a promising future, there are still some disadvantages when compared to microelectronic integration. The building blocks (BB) used in PDKs are still much larger than the microelectronic ones, and the optical power amplifiers consume more power than transistors, even if it is less than what consumed by microelectronic operational amplifiers [14].

There are several materials that can be used alone to develop PICs: Silicon (Si), Indium Phosphide (InP) or Gallium Arsenide (GaAs) are examples of this, although not the only possibilities. Another possibility consists of integrating different material technologies into single PICs (hybrid and heterogeneous integration). In any case, the use of SiPh provides a lower cost and reduces waveguide loss, which may change from 0.1 dB/cm in Si to more than 1 dB/cm for GaAs and InP. Thus, by using Si the need of power can be decreased [1,7,8,13]. It is also important to notice that the use of SiPh may be restrained to certain wavelengths, between 1.1 µm to 4 µm; in comparison to other materials such as SiN that possess a transparency window from 400 nm to 4 µm [11,15].

2.2 Silicon Photonics Platform

Due to its compatibility with CMOS fabrication technologies, the development of SiPh chips is easier and cheaper with respect to other technologies [2,13]. SiPh development is normally based on Silicon-on-Insulator (SOI) technologies, although under a broader definition it includes other technologies such as Silicon Nitride-on-Insulator. This definition of SiPh relies on every material that is able to make use of the existing CMOS methodologies [10,16].

There are two distinctive types of open-access SOI technologies, namely sub micrometer SOI and thick SOI. These two are classified according to the thickness of their guiding layer, belonging to thick SOI if it is larger than 1 µm and to sub micrometer SOI if it is in the range of 220 nm to 500 nm. The latter is nowadays the most commonly produced flavor of SiPh, and it offers the possibility for small bendings, up to 5 µm radius [10,17].

For SOI technologies there is currently an investigation on whether laser sources can be embedded within the PICs. To do so, it has been proposed to perform selective epitaxial growth of a III-V cavity in the SOI substrate. Another possibility consists of adding laser sources by using heterogeneous integration [16,18,19].

Since CMOS technologies are well-established within the semiconductor fabrication industry, there are several foundries and Integrated Device Manufacturers (IDM) that already produce SiPh PICs, and also different MPW providers. The latter are of special importance for academic purposes, because of a good compromise between price, small-scale production, and adaptability of the design. On these, the fabrication mask and the cost are shared among different clients, which all have a reduced portion of the wafer. The chips provided are normally sized between 10 mm^2 to 50 mm^2, although some companies such as LioniX offer larger areas [11,20]. Their price is of about 1000$/$mm^2$ However, the waiting times to develop chips using MPW runs may be of about 3 to 12 months, which adds difficulty for short-timed projects [10,12,17].

2.3 Design Process and Process Design Kits

To use the SiPh platforms Process Design Kits (PDKs) are given from the semiconductor fabrication facilities (fabs) to users. These contain the blocks to be used and information about the process followed, including as well the capabilities to verify the design and circuit simulations [12,21]. Nowadays there are no libraries used as a standard for the SiPh industry, and thus each fab presents its own solutions that may differ from one to another [10,21]. In recent times, there has been an effort between semiconductor fabs and design tool developers to develop libraries and Process Design Kits (PDKs) associated with their respective open-access technologies. By doing so, the accuracy and reliability of PICs is enhanced due to the possibility for developers to simulate and refine the performance of PIC designs prior to fabrication [21].

Two different flows for performing the design of the PICs are commonly used nowadays. The first is traditionally used and consists on designing the chip based on creating first the physical components based on a certain geometry for the materials. Hence, it will be referred to as the physical component design process flow. The obtained geometry will be optimized and later, an electromagnetic modelling technique will be applied to know the behavior of the component. After this it is possible to either define this component as a BB within a PDK or to merge several of these components and to perform an optical simulation for the response between the ports of the full design. If the second approach is used, once it is done, the full

design can be defined as a BB of a PDK. Following it, the circuit layout is generated by connecting the BBs hierarchically, which consist of a cell rather than a specific geometry. Finally, this generated layout follows verification, where the layout is checked in search of possible errors and violations of the design rules given by the fab [21,22]. A schematic of the steps followed by the physical component design process flow can be seen in Figure 2.

The second is an emerging process flow which is based on electronic design automation. Hence, it will be referred to as the Electronic Design Automation (EDA) process flow. It is based on using the already built BBs of a PDK to form a logical circuit, in which the components are connected to the others forming a specific topology. After forming it, a circuit simulation is completed, where the behavior of the circuit is characterized for all the elements. Now, if the simulation is satisfactory, we may proceed to create the circuit layout from the schematic, by hierarchically attaching the BBs. Now, the packaging is applied to the layout, which may change the properties of the circuit. Last, a verification process is applied, checking the physical layout properties, whether its behavior is correct and if it follows what described on the schematic [21]. For better comprehension, the steps followed by the electronic design automation process flow can be seen in Figure 3 as a schematic.

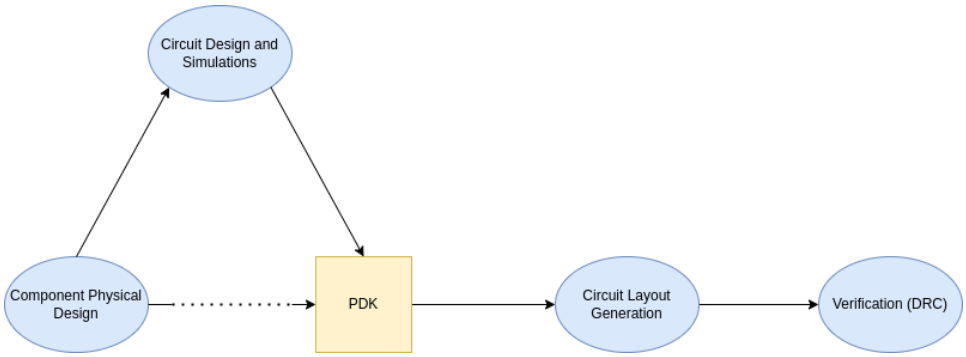


***Figure 2**. Physical component design process flow.*

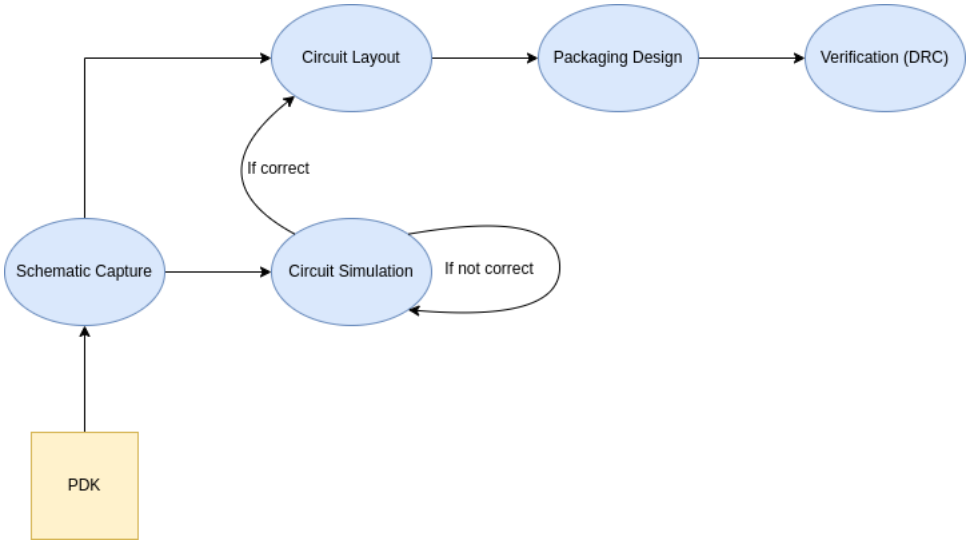


***Figure 3**. Electronic design automation process flow.*

2.4 Silicon Photonic Switches

Switches are devices capable of establishing desired connections from certain inputs to certain outputs. For the case of optical switches, the basic blocks most commonly used to design them are micro-ring resonators (MRR), Mach-Zehnder interferometers (MZI), a combination of both MMRs and MZIs, and micro-electro mechanical systems (MEMS) waveguide couplers [1,3].

The MRR is a resonant device, which offers a wavelength selective filtering feature. In its simplest configuration, it is made by placing very closely a ring-shaped and two linear waveguides [1,2]. The MZI consists of an interferometer which splits the input signal into two different paths to later recombine them. By controlling the phase difference between the two branches we are able to perform the switching [1]. Last, MEMS are specific devices made of waveguides that are subject to mechanical forces, being able to change its geometry [2,9]. MRRs present a more promising approach than MZIs because of their smaller footprint, and lower power consumption, but they also present disadvantages such as a narrow bandwidth and temperature susceptibility. The combination of MRR and MZI is still being studied, but currently it still presents a narrow bandwidth and large insertion loss [1].

Within the SiPh switches we may distinguish four diverse approaches for design. We may develop the switch utilizing the thermo-optic effect, use the electro-optic effect, employ phase-changing materials (PCM), or make use of MEMS [1,6].

2.4.1 Thermo-optic effect

The thermo-optical (TO) switches shift the phase of the waves travelling through a waveguide due to its heating. To do so, a resistive heater is placed along the waveguide, raising its temperature [6,23]. The phase delay experimented by any material can be described by Eq. (1):

$$\Delta\phi(\lambda, T) = \frac{2\pi}{\lambda}\Delta n(T)\Gamma L \quad (1)$$

where λ corresponds to the wavelength of the signal, $\Delta n(T)$ is the temperature-dependent refractive index change of the material, Γ is the optical confinement factor of the waveguide, and L is the length of the heated section of the waveguide [6].

2.4.2 Electro-optic effect

The electro-optic (EO) switches change their phase shift due to the carrier dispersion effect. By applying currents to intrinsic Si, a positive-intrinsic-negative junction (PIN) is formed, and the carrier dispersion effect is induced. This one will change the refractive index along the PIN junction [1,6]. Another type of EO switches may be based on the Kerr, Franz-Keldysh and Pockels effect. The first and second ones are negligible in comparison to the carrier dispersion in Silicon, while the Pockels effect is absent, so only the carrier dispersion effect will be studied for Si [3,24].

2.4.3 Micro-electromechanical systems

MEMS function by moving or deforming the waveguides, so that there is a change of phase at the end of the waveguide. It can be performed by applying a compressive or tensile force to the material, which may induce absorption and a change in the refractive index. It can also be performed by stretching or elongating, which in fact has the effect of a compressive or tensile force, adding a shift of phase due to a new relative position of the output port of the waveguide. Finally, it can also be performed by the displacement of a slab of the material, which changes the field mode in the waveguide, generating a field profile change [2 ,9].

2.4.4 Phase Changing Materials

A different approach consists of adding new material to the silicon-integrated optical switch, which presents huge changes of their optical properties when an external stimulus is applied. The most commonly used PCMs in Si switches are vanadium dioxide (VO2) and $Ge_2Sb_2Te_5$ (GST). [1]

2.4.5 Comparison of the approaches

The different approaches presented also have different characteristics when applied. Between these distinctive characteristics, there are certain key parameters that help to choose the approach for the tuning method, such as the power consumption, the footprint of the device and the speed of tuning. For a better comprehension, Table 1 presents a comparison between distinct approaches to obtain a SiPh switch at usual telecommunication wavelengths [1,9].

	Power Consumption	Footprint	Speed	Limitations
TO	--	+	-	Footprint limited by

				inner characteristics
EO	+	++	++	High optical losses
MEMS	++	+	-	
PCM	++	+	+	bad wavelength selectivity

Table 1. Comparison of different approaches to SiPh switch designing.

The EO method presented refers to the carrier dispersion effect. Its main limitation are the optical losses, which are higher when compared to other methods. It is faster than the others, being the tuning speed placed in the order of the nanoseconds and picoseconds generally [1]. The TO approach presents the highest power consumption, due to the need of heating [23]. Its footprint is limited by the thermo-optic coefficient and the crosstalk between adjacent devices [9]. The MEMs are a good approach in terms of power consumption because they mainly need the supply when a mechanical action is happening. If no mechanical action is happening the consumption is minimal, just caused by current leakages [9]. Regarding PCMs, although they present good overall characteristics in terms of power consumption, footprint, and speed, they cannot be used for wavelength division multiplexing (WDM), due to a short wavelength selectivity [1].

3 Chip Design Methodology and Component Description

3.1 Design process: characteristics and elements

The scope of this project is to design the mask of a Silicon Photonics Chip for Telecommunications, specifically a Wavelength Selective Switch (WSS). To accomplish it, firstly an approximate layout with the connections and the positioning of the single elements will be developed using the PHIDL library from Python, which allows to create layouts as Graphic Design System (GDS) files. The library is particularly useful for that stage, because it simplifies many of the low-level layout details, making easier to focus on the design, rather than in the single components. It also allows for a robust positioning of the elements, even if changes are introduced. Later, to perform the mask realization, the software OptoDesigner from Synopsys will be used, which allows for Photonics Integrated Circuit (PIC) Design using a proprietary programming language. Among the multiple benefits of using this software we find that it allows for wide access to different technologies and there are many supported foundries PDKs, being widely recognized and used in the semiconductor fab industry [26].

The designed chip will need to fulfill specific characteristics. First, it will need to operate in the region comprising the S-band (1460-1530 nm), C-band (1530-1565 nm) and L-band (1565-1625 nm) [27]. The chip will receive in input from a single optical

fiber 24 channels with 100 GHz spacing, 8 in the S-band, 8 in the C-band and 8 in the L-band. Through a proper design and connection of Mach-Zehnder interferometers and directional couplers, the 24 signals will be programmatically routed to 3 output fibers, implementing therefore a non-blocking multi-band fully optical switch. The interferometers and couplers have been designed by the research group in Politecnico di Torino.

To perform the full design of the chip layout given the design specifications, several designs of individual components and groups ("blocks") of components will be done independently. After the design of all of these, they will all be connected to obtain the full mask. Therefore, the full design will be composed of the following components: Fiber couplers to act as the input and outputs of the circuit; electrical control and pads to decide and perform the switching; MZIs which split the signals according to their wavelengths; and CDCs based on Bragg grating, which are used for multiplexing and demultiplexing the signals.

3.2 Fiber couplers

Fiber couplers are needed in order to connect the WSS chip to its input and output ports, as well as testing ports. The reason these are used for fiber-to-chip coupling is because it significantly reduces the losses from fibers to waveguides, delivering the most of the power from one side to the other. For an optimal coupling, the size and geometry of the elements, the fundamental modes supported, the index difference and the optical alignment need all to be taken into account [28]. There is a critical angle that defines the last angle θ_a at which there is optical alignment, which takes into account the index differences between the two different waveguides. This, along with the coupled-mode theory can be calculated and implemented in order to improve the coupling. [29].

3.3 Contra directional Couplers

CDCs will be used in order to filter the signals, as band-pass filters. Using several of them, we are able to obtain all the 8 channels within the 3 used bands (S+C+L optical telecommunication bands).

The CDCs are components formed by two asymmetric waveguides placed closely to each other, allowing coupling between them [30,31]. These two waveguides are specifically grating-assisted couplers, which means that the waveguides exhibit a serrated profile, featuring corrugations along their surface. The period of the profile (pitch) will not necessarily be constant, as it might slightly increase throughout the waveguide. The difference between the last and first pitches is called chirp, and it will not be null on the CDCs used for band filtering. To better understand CDCs, a schematic can be seen in Figure 4. For the CDCs, W_1 and W_2 are the width of the two waveguides, ΔW_1 and ΔW_2 represent the widths for the corrugations of the two waveguides, Λ_1 and Λ_n are the periods of the first and last pitches, $\Delta\Lambda$ is the chirp, G is the gap between the teeth of the two waveguides and L is the length of the CDC [31].

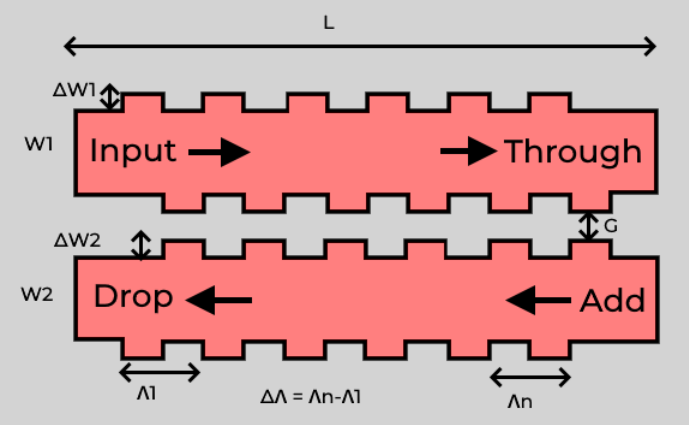


Figure 4. CDC schematic.

CDCs couple light for a selected wavelength from one waveguide to the other, with opposite directions [32]. Unfortunately, symmetrical waveguides cause side lobes, so in order to avoid it a technique called apodization might be used. This technique consists on changing the pitches along the waveguides, to obtain the chirp. By doing so, the effective refractive index and the coupling coefficients are changed, and therefore the wavelengths belonging to other channels are further attenuated. [30,31,32].

To calculate the necessary pitch for the grating we can use Eq. (2) which depends on the central wavelength of the band-pass filter (λ_D). The other terms on this equation are $n_{eff,1}$ and $n_{eff,2}$ which correspond to the effective refractive indices of each waveguide [30].

$$\Lambda = \frac{\lambda_D}{n_{eff,1} + n_{eff,2}} \tag{2}$$

3.4 Mach-Zehnder Interferometers

In this project, the MZIs are used after demultiplexing to split a multichannel input signal into two different signals defined by their wavelength and select one of them. This is possible due to the fact that by varying the optical path difference it is possible to concentrate all the power of a particular optical frequency into one of the outputs [25]. In this case, both the geometrical paths of the MZI are symmetrical, so in order to introduce phase delay, the thermo-optic effect will be applied to a section of one of the paths. This effect is described by Eq. (1) and as we can see, it is dependent on the wavelength.

A diagram illustrating the operation of the MZI is presented in Figure 5. In it, an input beam is split into two different paths. The light passing through both arms will endure different phase shifts, leading to an optical delay ($\Delta\Phi$) between the two arms. At the second beamsplitter, the light reflecting from one path and the one transmitted from the other path will merge and produce interference. There will be

two interferences made by the light from the two paths: one travelling to one detector and another travelling to another detector. The phase delay between the waves going to one detector and the other equals to π radians, so by adjusting $\Delta\Phi$ we may find a value such that the interference is destructive on one side and constructive on the other [33]. Another way to obtain this result is using quantum mechanics. By doing so, it will be seen that for a photon entering the MZI, the probabilities of watching them at the two detectors are respectively the square sine and square cosine of half the phase shift [34].

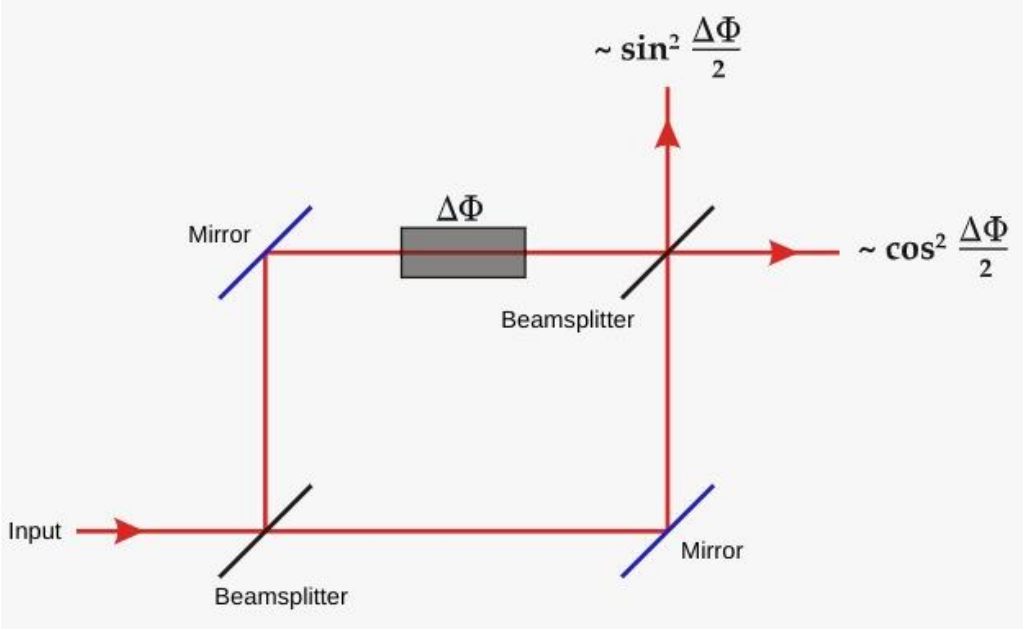


***Figure 5**. Diagram illustrating the operation of a Mach-Zehnder interferometer.*

3.5 Electrical control and pads

Apart from optical elements, there are present electrical circuits, whose function within the chip consists of controlling the switching of the MZIs. To do so, electrical pads will be connected to heaters, being these heaters closely placed to one of the arms of the MZI. When a voltage is applied between the electrical pads, current will flow through the heater, increasing its temperature, and thus, by means of the thermo-optic effect, adding a phase delay between the two arms. By controlling the voltage difference, we may control the MZI in such a way that for a certain wavelength the output power is all concentrated on one output, while being null on the other.

For the heaters there are several materials that might be used, such as Tungsten, silicide, and doped silicon [35].

Two different approaches had been proposed and studied to produce phase shift. In both cases the electrical heater will be placed above the element to which the phase shift is going to be applied. The first approach consists of a parallel line to the

element whose refractive index needs to be changed, with perpendicular arms for the electrical connection, but parallel among them. The second approach consists of a simple line made by the electrical lines and a small heater on top of the element whose phase will be shifted. The electrical lines would enter and exit from opposite sides. Both the cases are represented schematically in Figure 6: (a) and (b) respectively. For the design of the WSS, the first approach will be used since it is in close contact to the optical phase shifting element for longer, thus requiring a smaller current. The second one could be used in other conditions with thick equal to the length of the element in which the phase shift is developed, but due to a high number of elements, it cannot be implemented in this manner for the WSS chip.

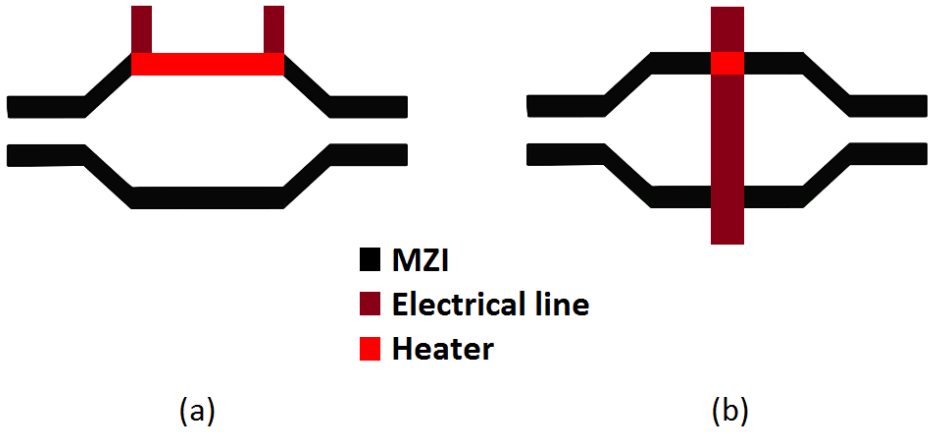


***Figure 6**. Schematic of the electrically controlled heaters applied to a MZI. **(a)** Parallel heater with perpendicular electrical lines layout. **(b)** Perpendicular heater with continuing electrical lines.*

4 Layout of the chip

Back to the design of the chip, it becomes important to find and design the full layout of the integrated circuit, since it contains a substantial number of different elements interconnected. For the design of the layout 2 rules will be followed:

1. The area of the chip should be minimized.
2. The number of waveguide crossings should be minimized.

4.1 Elements of the layout

To compose the full circuit the following elements will be used:

- Mach-Zehnder interferometers (MZIs), which are approximately 500 µm length and 6 µm width.
- Contra directional couplers (CDCs) for the bands, which are approximately 1500 µm length and 6 µm width.
- Contra directional couplers (CDCs) for the channels, which are approximately 900 µm length and 6 µm width.

- Fiber couplers for the input, output, and testing ports, which have exactly 127 µm separation among them. Their size is 35.603 µm x 22.152 µm.

while the elements used for connections are:

- Straight optical fibers to connect the elements, whose width is approximately 0.5 µm.
- Curved optical fibers, whose width is assumed to be the same as the straight ones. The radius of the curves (to the center of the fiber) is fixed to 5 µm to reduce the propagation losses of the optical signal.
- Crossings between waveguides.

Between the inserted components an empty space of 5 µm is introduced in order to reduce coupling effects. Crossings impose an even further distance to minimize optical losses: in this case an empty space is required within a square of 20 µm length centered at the middle point of the crossing.

Also, the elements to be connected (CDCs and MZIs) have specific port positions, which must be considered to design the layout correctly.

4.2 Theoretical block description

The designed chip will be capable of working with 3 bands, each of them containing 8 different channels. Thus, there will be 24 channels in total. The 8 channels within a band must be connected one to another, and the same for the bands. It is important to note that not all channels and bands are the same, containing different elements ones from others.

Apart from the band and channel description, the elements and blocks of elements are assigned distinct functions within the system. Namely, there will be 3 sections performing contrasting functions: one for filtering the bands and channels, by demultiplexing the signals, another for switching between them and lastly a multiplexing one to choose the desired outputs, as shown in Figure 7.

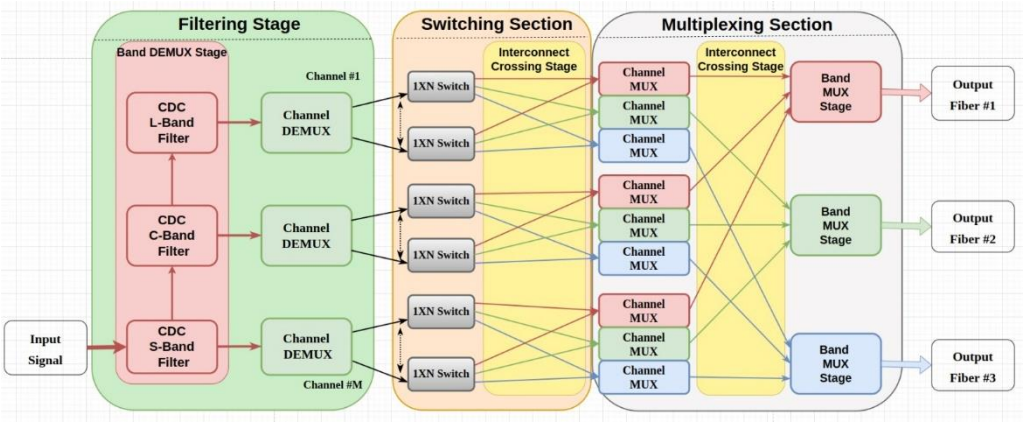


***Figure 7**. Working sections of the WSS chip.*

These 3 different sections and their inner elements are better described in Figure 8, where the elements for the demultiplexing, switching and multiplexing sections and their connections are shown for each channel. These will form networks within the chip that will perform a specific function for the different channels. Since we have 3 different fiber outputs, there will be 24 channel networks for multiplexing and demultiplexing.

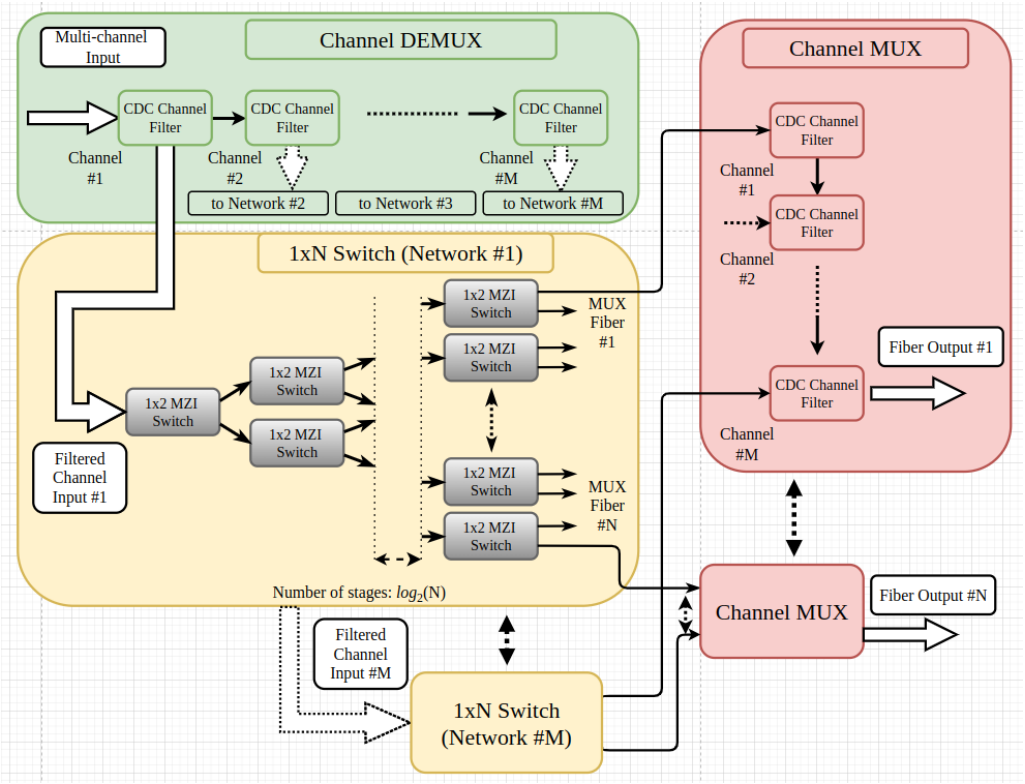


***Figure 8**. Schematic description of the sections of the WSS chip at channel level.*

4.3 Ports of the elements

To connect the elements in OptoDesigner, we need to consider the position of the ports. The CDCs for multiplexing and demultiplexing will present diverse directions of the input and output signals, and the connections to the rest of the schematic will be different as well. The contra directional couplers for demultiplexing (input CDCs) possess an input port (denoted as IN), a drop port (DP) and a through port (TH), being this last one described as the difference between the input and the drop (i.e. $TH_{demux} = IN_{demux} - DP$). On the other hand, the contra directional couplers for multiplexing (output CDCs) have an input, add (denoted as AD) and through ports, being this last one described as the sum of the two other ports (i.e. $TH_{mux} = IN_{mux} + AD$). To represent this, descriptive schematics of the ports of both the input and output CDCs are shown in Figure 9.

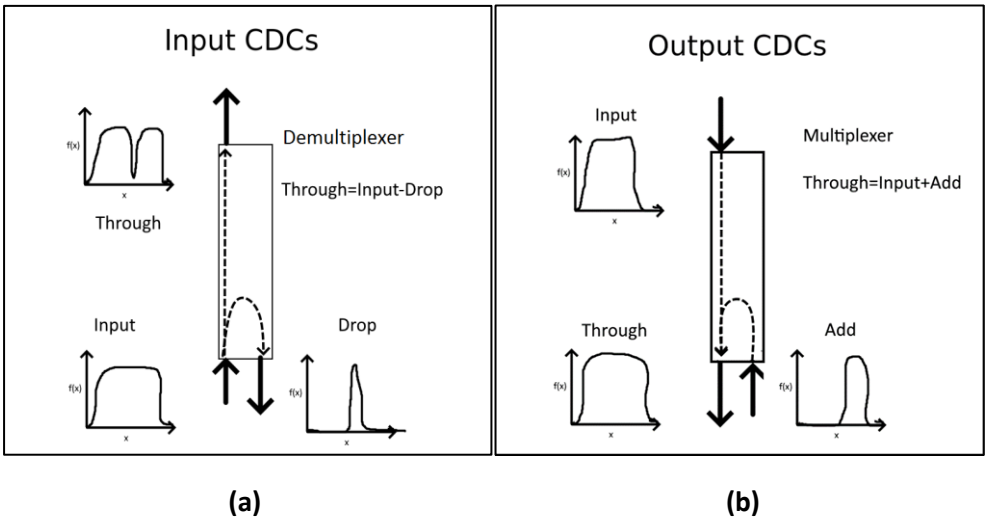


***Figure 9**. Ports of the CDCs for **(a)** demultiplexing and **(b)** multiplexing.*

The MZIs used for switching also present a specific port layout. There are 4 different ports in each of these, and they are given the same names as the ones for the CDCs. For this specific project, 3 of them will be used, namely the input port (IN), the through port (TH) and the drop Port (DP). These two output port are complimentary, and again the input signal is divided between the two ports (i.e. $IN = TH + DP$). The schematic describing the ports and its positions within a single MZI is given in Figure 10. Since the power at the output might be controlled by changing the phase shift, it will be tuned accordingly so that the full input power emerges only in the desired output port. This makes the choice of the output ports arbitrary, since different tunings might give the same output if the ports are reversed.

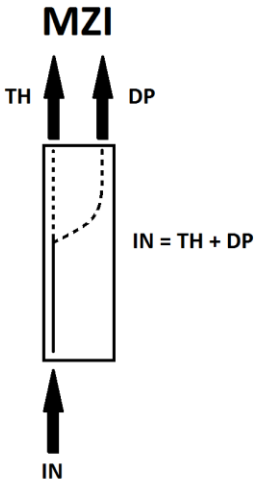


***Figure 10**. Port layout schematic for a single MZI.*

In this project, MZIs come in pairs of 2, to perform switching between 3 different channels, by cascading one MZI to the other. As it was previously said, the decision made to assign the ports is arbitrary, and in this case it will be chosen that the DP port of the first MZI connects to the IN port of the second MZI. If we define these two cascaded MZIs as a single block for switching, it will present an input port to a channel demultiplexer (denoted as IN), three other port connections to 3 different channel multiplexers (denoted as P1, P2 and P3) and an interconnection between the two MZIs (denoted as IC1 and IC2). These connections of the ports of the single MZIs within the block and the ones as a full block are shown in Figure 11.

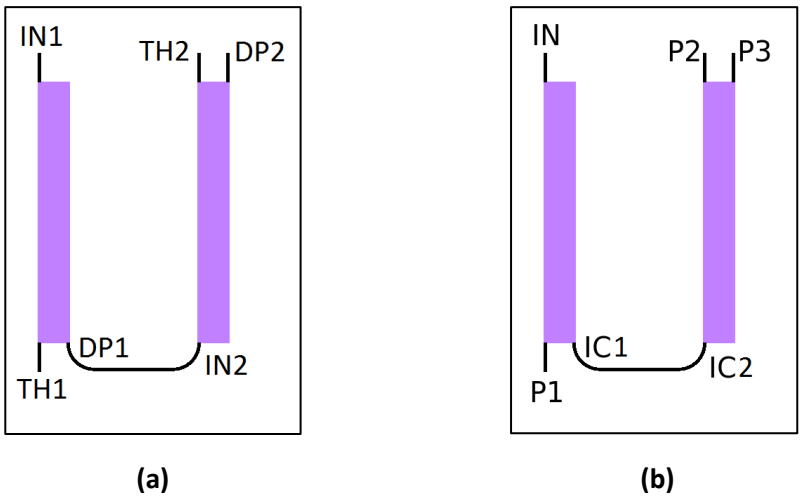


***Figure 11**. Ports of the cascaded MZIs **(a)** as the ports of each MZI and **(b)** as the ports of the single block.*

4.4 Connections between the elements

Now that all the elements of the chip and its ports have been defined, we may start to describe the connections and the ports that are used to link all the different components. To connect all these components, we must look at how the structures connect. Based on the description made on the 4.2 section, it can be seen that the input port is first connected to the S-Band CDC filter, which connects in cascade to the C-Band and L-Band filters, as seen in Figure 7. To do so, and since these 3 are demultiplexers, they are connected through the input port to the previous band CDC while the previous is connected using the TH port (For the S-Band CDC, the Input port connects to the Input port of the full WSS chip). These 3 band CDCs are respectively connected each to a channel CDC demultiplexer using their DP port (Linked to the Input port of the first channel CDC). Meanwhile, these channel CDCs are connected in cascade to another seven of them. The connection ports are IN when linked to the previous one and TH when connected to the next one.

Then, for the switching section, each one of the channel CDCs is connected to a block of cascaded MZIs through its DP port, while the port used by the MZIs pair is the IN one.

Now for the multiplexing section, all the three outputs of the MZI pairs (P1, P2 and

P3) are connected to identical but independent networks. Basically, the first pair of MZIs belonging to channel #1 is connected to the input port of a channel CDC multiplexer, which is in turn connected in cascade to the IN ports of another six of them by using the TH port. Each one of these will also be connected to the output of the next pair of MZIs through their AD port. After including these 3 channel multiplexing networks, it becomes necessary to merge them. To do so, 2 band CDC multiplexers will be used. The TH port of the last channel CDC multiplexer within the network attached to the L-Band demultiplexer will present a connection to the IN port of the first band demultiplexer. Similarly, the TH port of the last channel demultiplexer linked to the middle band multiplexer will connect to the AD port of this first band multiplexer. Also, its TH port is connected to the input of the second band multiplexer, and the TH port of the branch linked to the first CDC band demultiplexer connects to the AD port. Finally, the TH port of this last multiplexer goes to an output port of the WSS chip.
Because of the difficulty of the connections, a schematic showing the port diagram of the different elements is present in Figure 12. To improve the readability and understandability, the networks connected to P2 and P3 for the different pairs of MZIs are not present. In any case, they present the exact same connections and ports as the one for P1, these three networks being independent between them.

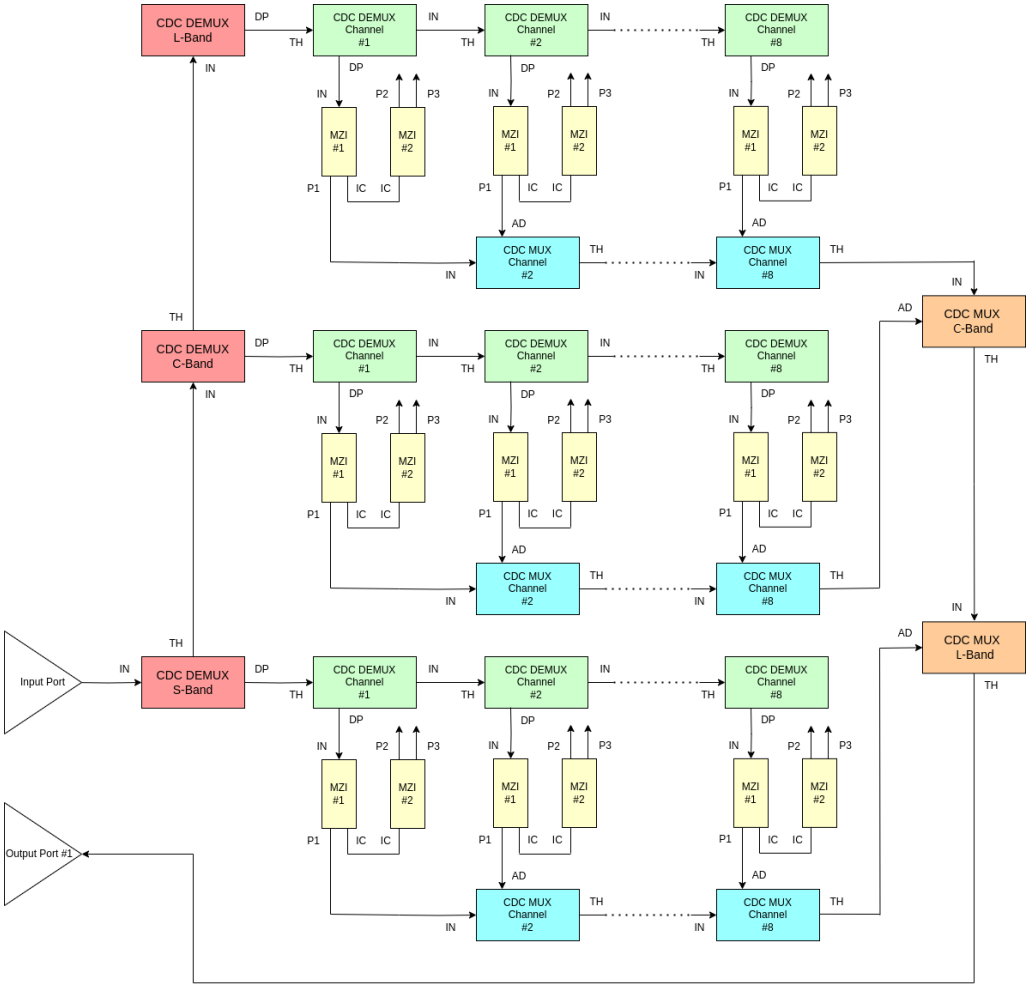


***Figure 12**. Port schematic of the elements of the WSS chip (with only one multiplexing network linked to the output port P1).*

4.5 Approach for designing the layout of the WSS device

Now that the approximate measures of all the optical elements and the connections between them have been defined, a specific approach is proposed towards designing the chip, following the rules defined earlier at the beginning of this section. By following this approach, an approximation of the final layout will be presented. The reason behind this approximation is that the components that make up the WSS chip may slightly vary depending on the characteristics of the chip. Therefore, by creating an approximation of this layout, it can be later tuned with minimum changes to perform its functions, while still presenting the same shape. To represent the approximate layout, the PHIDL library from Python is used. The full code and the link to this library can be found on Appendix A. Using it, the single components will be placed and connected to each other, with the only need of knowledge of the relative port positions among them within the components. Furthermore, its user-friendliness and its ability to rapidly create and manipulate geometries make it a good choice to start approaching the layout design.

The connection of the different elements can be made by connecting smaller networks to one another. In this work, firstly the channel networks will be designed, containing each the demultiplexing, the switching and the multiplexing part of a single channel. Eight of these will be connected to make a band network and three of these band networks plus its connections to the ports will be finally linked to create the full layout of the WSS chip.

In order to minimize the area, and to ensure similar component properties with respect to growing issues on the Si substrate, it is proposed to place all the CDCs and MZIs along the same direction (it will be said to be vertically because of the relative position within the GDS file), with one channel CDC on top of a MZI, which is also on top of another channel CDC, summing 2300 μm plus the connections and necessary spaces. This will be the measure restricting height. Now, for the case of the band CDCs, they will be placed alone, one after another (horizontally) because of their huge length (1500 μm). This way we can combine horizontally the band CDCs and the channel blocks to complete the full layout. Using this type of layout, the chip size from top to bottom is of 2449.5 μm and 1980.25 μm from right to left, so the total area needed to build the chip would be 4.85 mm^2 approximately.

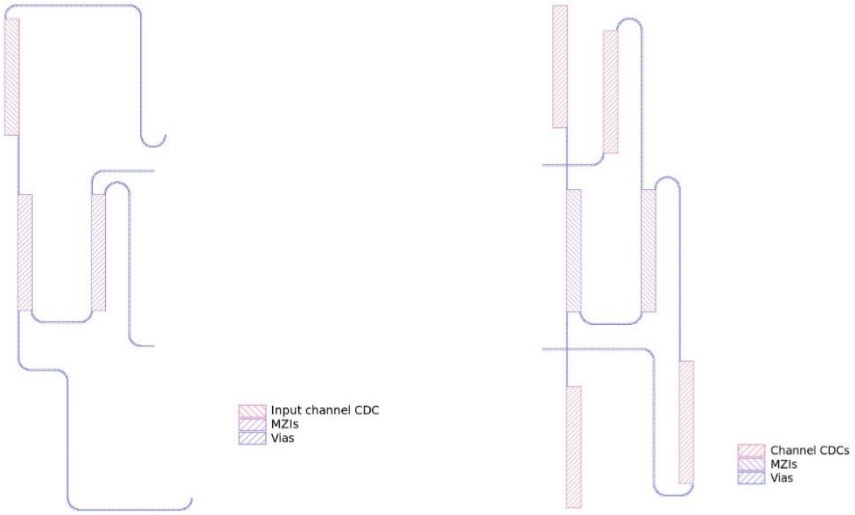
For the representation of the layout, CDCs and Mach-Zehnder interferometers are replaced by simplified shapes (rectangles). Moreover, two different full designs have been created: one with the real measures of all the elements and a simplified version with smaller measures for the elements. In this last version, the height of the channel CDCs and MZIs has been reduced to 50 μm to make it easier to see the relative position of the elements when compared to the others, and allow for a debug of the connections. Band CDCs have been set to 170 μm to keep a good balance between aspect ratio relation to the channel CDCs and its correct position without any further connections. Last, the separation distance between the ports has been set to 30 μm. It is important to notice that the simplified version will be represented for the visualization of the different smaller structures that compose the full WSS chip, due to its nature, which greatly simplifies visualization.

Due to the complexity of the design, smaller blocks will be developed, and they will finally be merged in order to obtain the full WSS design. There will be 3 diverse blocks, representing the filtering and switching of the single channels and that of the bands.

4.5.1 Channel multiplexer/demultiplexer filters and routing

The channel blocks are physical components which handle the signal contained in a single channel. It performs the demultiplexing, switching and multiplexing for any single channel. Of these blocks, we must distinguish two distinct types. The differences between them are the presence of multiplexers after the switching stage, as shown in Figure 13. The channel block without multiplexers is used for the first channel of a band, since the signal that will be passed to the next channel block is the only one at that point, and thus there is no need for multiplexing signals. This block is simpler and due to this simplicity, it does not require the use of crossings. On the

other hand, the second one will need them due to a more complex layout. It presents output band CDCs (multiplexers) due to the possibility of having to merge two different signals, and thus it is present for all the channels within a band except for the first one.



(a) **(b)**

***Figure 13**. Channel blocks used for the design of the chip (simplified layout). **(a)** Leftmost channel block without output channel CDCs and its connections to the next channel. **(b)** Channel block with 3 output channel CDCs.*

4.5.2 Band multiplexer/demultiplexer filters

The band block, is the analogous of the channel block, but with respect to the bands. It performs the demultiplexing and multiplexing of the single bands. Regarding the physical layout, it contains 8 channel blocks, which have on their left side the input band CDC, while on its right there are the 3 output band CDCs. The 8 channel blocks are not the same, being the first on the left the same as the channel block shown in Figure 13(a) is present, while the rest of them are as the one shown in Figure 13(b). This first one has its input channel CDC connected to the input band CDC on its left, while the output CDCs of the last channel block are connected to the output band CDCs to its right.

It is important to notice that again there are two diverse types of connections groups: one present on the left and on the middle, which contains the output band CDCs for multiplexing; and another to the right which does not contain these CDCs. These two different band blocks can be seen in Figure 14 and Figure 15.

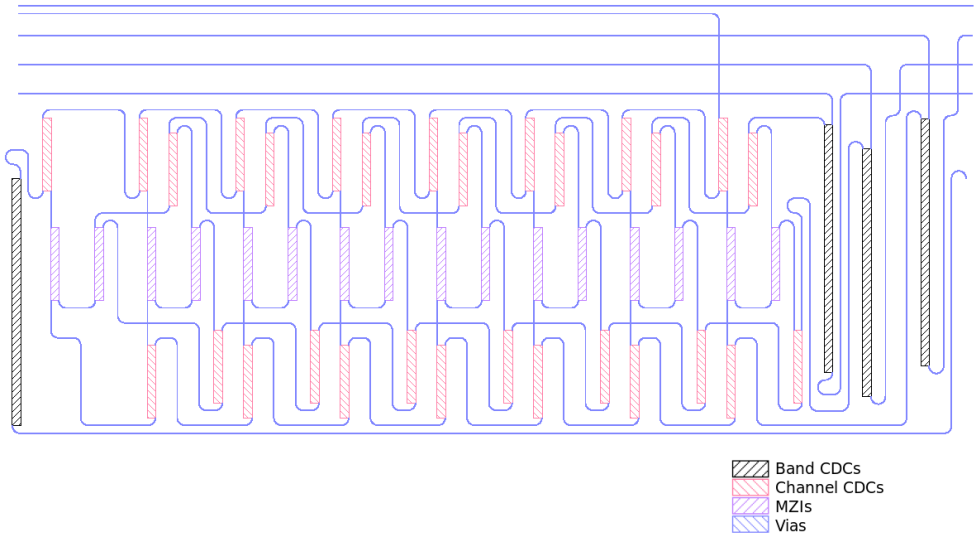


***Figure 14**. Band block present on the middle of the WSS chip (simplified layout).*

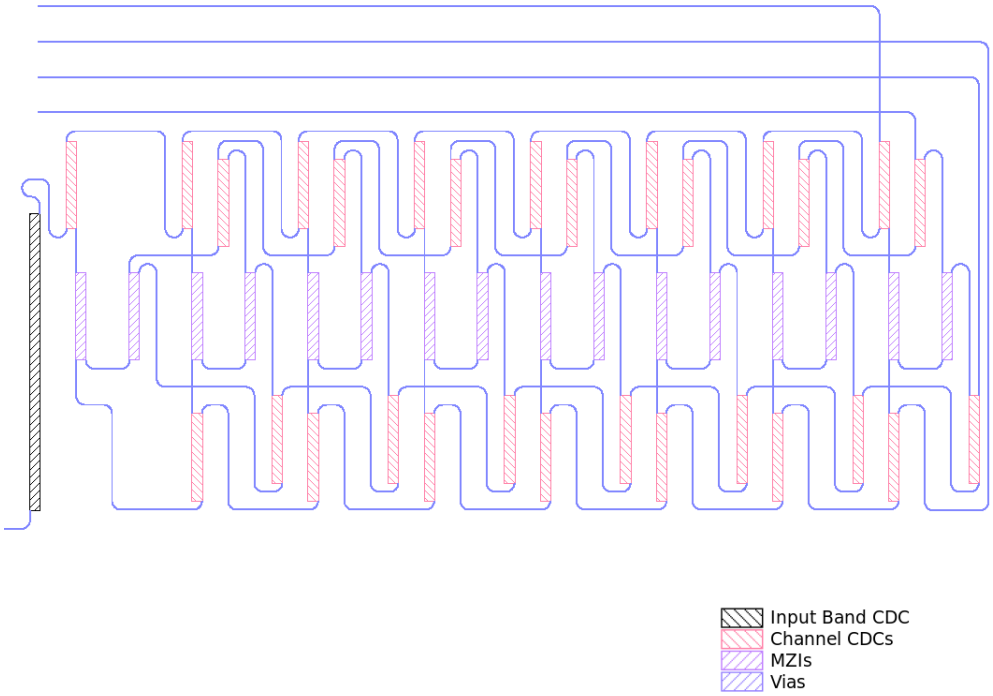


***Figure 15**. Band block present on the right of the WSS chip (simplified layout).*

4.5.3 Complete WSS layout

Finally, by combining the 3 band blocks we are able to obtain the layout of the chip. Nevertheless, the input and output ports still need to be added. They will be placed to the left side of the block, and will have a distance of 127 µm, which is imposed because of the multichannel fiber-to-chip coupling used for packaging in the industry

[4]. The simplified layout and the circuit with correct sizes are shown in Figure 16 and Figure 17 respectively.

Apart from the previously mentioned elements, there are specific terminal ports for testing. These have been added to test the proper operation of each band. There are 4 testing ports: one for the last input channel CDC of each band and one for the last input band CDC. The terminal ports are connected to the TH port of each respective input CDC, which as stated previously, equals to the sum of the input port minus the drop port. At this port of the input channel CDCs we are measuring, given that they are placed at the end of the band, the signal must be zero. This is due to the fact that at this point all the channels within the band must have been already routed. Similarly, for the input band CDC, the ideal result would be zero, because the other 2 bands should have been routed at the other band demultiplexers, while the one for this band must be routed to the DP port.

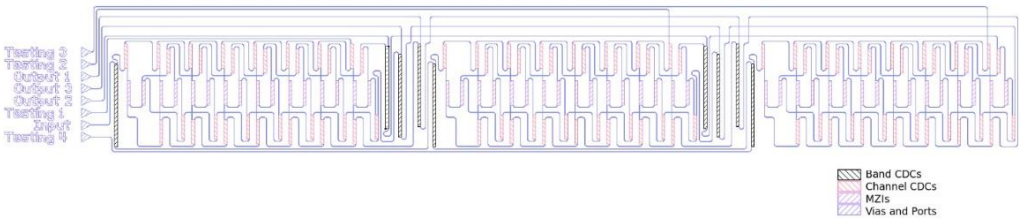


***Figure 16**. Simplified layout of the WSS device.*

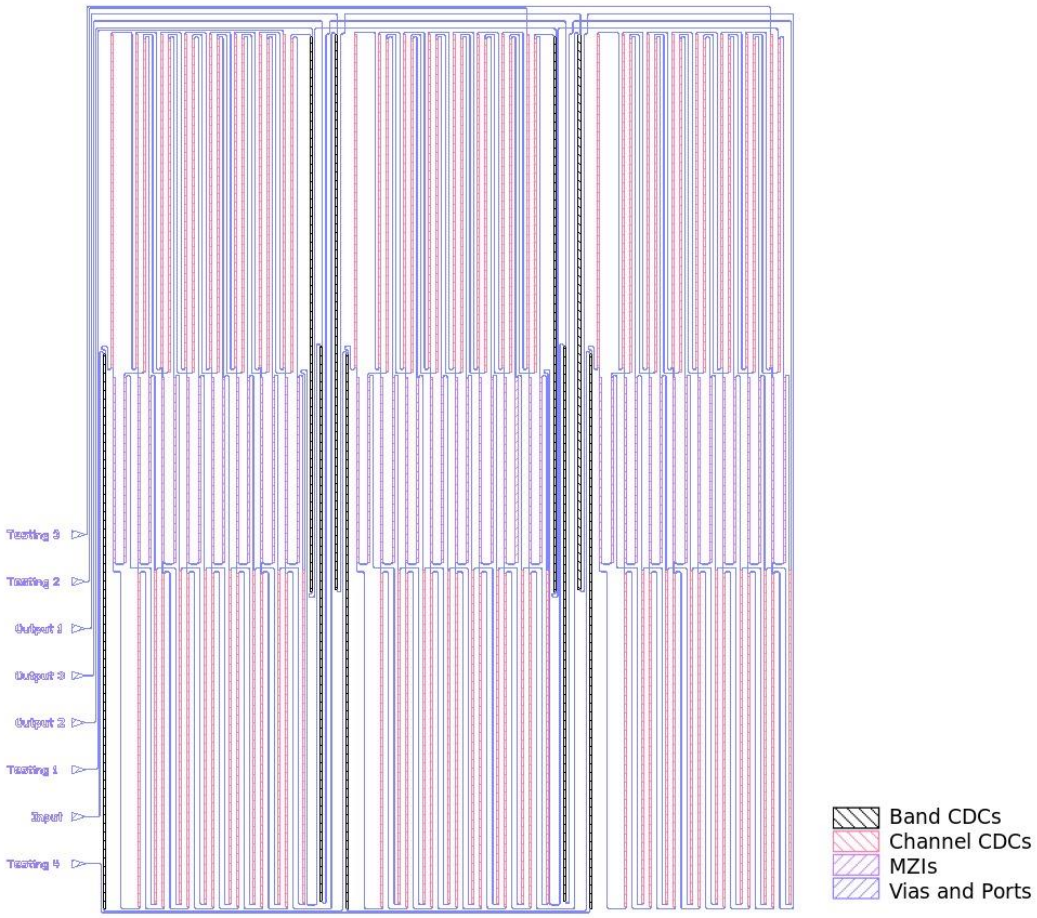


***Figure 17**. True layout of the WSS device.*

5 Design of the WSS components with OptoDesigner

5.1 Waveguides design

The waveguides are used to link optically the different components in the layout. They are critical, since there are hundreds of them, and they are present in different shapes within the chip, such as S-bends, semicircles and crossings. Their geometry has been computed and chosen by simulations. After them, it has been decided the usage of SiPh waveguides whose width is W = 0.55 µm and height H = 0.22 µm, which are standard values for C-band SiPh waveguides. When the width of the waveguides is required to change, ideal tapered waveguides will be assumed.

5.2 CDC design

5.2.1 CDC parameters

The CDCs are necessary for filtering the incoming 3 bands and their respective channels. There will be 96 of them with two different purposes: filtering the band and filtering the channel within a band. Depending on which band and channel they are filtering, their geometrical specifications may differ. Furthermore, there will be a major design difference between channel and band CDCs: in the latter, chirp will be present, meaning that the pitch is not continuous throughout the device, while the former does not present chirp.

The different characteristics defining the CDCs for the WSS chip can be seen on Figure 4, and are summarized as the following:

- Width of each of the waveguides (W_1 and W_2)
- Width of the corrugations for each waveguide (ΔW_1)
- Gap length between the two waveguides (G)
- Length of the corrugated waveguides (L)
- Pitch (Λ)
- Chirp ($\Delta\Lambda$)

For the development within the WSS chip, we must design those using different parameters for each of the CDCs, by applying the Coupled-Mode Theory (CMT). The obtained parameters are presented in Table 2 below, which contains the different values applied to each of the band couplers, while Table 3 contains the values applied to the channel couplers.

	S-band	C-band	L-band
Λ (nm)	275	293	302
$\Delta\Lambda$ (nm)	20	9	18
L (mm)	1.5	0.8	1.4
W_1 (nm)	570		
ΔW_1 (nm)	100		
W_2 (nm)	430		

ΔW_2 (nm)	60
G (nm)	100

Table 2. Parameters used for the design of the band CDCs.

	S-band	C-band	L-band
Λ_1 (nm)	284.2	298.9	312.6
Λ_2 (nm)	284	298.7	312.4
Λ_3 (nm)	283.8	298.5	312.1
Λ_4 (nm)	283.6	298.2	311.9
Λ_5 (nm)	283.4	298	311.6
Λ_6 (nm)	283.2	297.8	311.4
Λ_7 (nm)	283	297.6	311
Λ_8 (nm)	282.8	297.3	310.9
L (mm)	0.94		
W_1 (nm)	600		
ΔW_1 (nm)	20		
W_2 (nm)	400		
ΔW_2 (nm)	40		
G (nm)	300		

Table 3. Parameters used for the design of the channel CDCs.

5.2.2 Implementing the CDC as a building block

To implement the CDCs, a function layout will be developed which returns the CDC as a BB, given its geometrical characteristics as input. The function defining the BB that implements a CDC can be found in Appendix D. This function can be used for both band and channel CDCs; and contains specific new parameters which may be useful for different projects. For instance, it allows to define a functor, which will be used to modify the width of the corrugations throughout the CDC, multiplying the specified value by the functor. It also permits the developer to define the duty-cycle of the device. It is important to notice that the created CDC will not have exactly the length specified as input, due to the combination of pitch and chirp, which already define for a certain number of cycles the length of the CDC. As a rule, the CDCs are considerably larger than their periods, and may include thousands of periods. Therefore, this change of length can be considered negligible regarding the functioning of the CDC coupler, since it is always less than the last pitch.

It is important to note that the CDC needs to be boxed in order to be placed in the WSS chip. This boxing process involves adjusting the position of the CDC and adding connective waveguides so that there is a smooth transition to external components, thus placing the elements in a box. To develop it as a function layout, 3 new parameters will be added, which allow the connections of the CDC as a block within the WSS chip. These 3 are the width of the box (W_b), in the opposite direction of the coupling length; the width of the waveguides outside the box (W_o), to which the component is connected; and the length of the connecting waveguides (L_b),

measured in the same axis as the coupling length. By making use of these, two S-bend waveguides are created, connecting the CDC with the outer waveguides and permitting a gradual convergence between different widths of the waveguides within and outside the device. As a consequence of the addition of these S-bends, now the length of the full device will be equal to the length of the corrugated waveguides plus two times the lengths of the S-bends (i.e. $2L_b + L$.) The two CDC corrugated waveguides and the S-bends present at the layout will have by default the OptoDesigner Demofab mask cross-section (MCS) "*mcsSOI_DEEP*" applied. A different MCS may be used if it is explicitly specified at the input parameters.

An example of the implementation of a CDC, prior to the boxing process and using OptoDesigner is given in Figure 18, where the input parameters are the ones given for the second trial of the CDC in the code in Appendix D, and presented in Table 4. By using the chosen input parameters of this trial, the geometrical shape of the CDC and the differences between its subcomponents might be seen. Now, to present the boxed element, the same CDC block is used for Figure 19, which can be consulted on the first trial in of the CDC in the code in Appendix D. All the input parameters used for the design of the physical CDC BB are again listed in Table 4, while the parameters used for boxing are presented in Table 5.

Functor	W_1	W_2	ΔW_1	ΔW_2	Λ	Duty-cycle	L	$\Delta\Lambda$	G
"functor1" $f(x) = 1$	0.57 μm	0.43 μm	0.1 μm	0.06 μm	0.275 μm	50 %	15 μm	0.1 μm	0.05 μm

Table 4. Input parameters used for the CDC implementation trials in Appendix D.

W_b	W_o	L_b
6 μm	0.5 μm	5 μm

Table 5. Input parameters used for 2nd CDC implementation trial (boxed) in Appendix D.

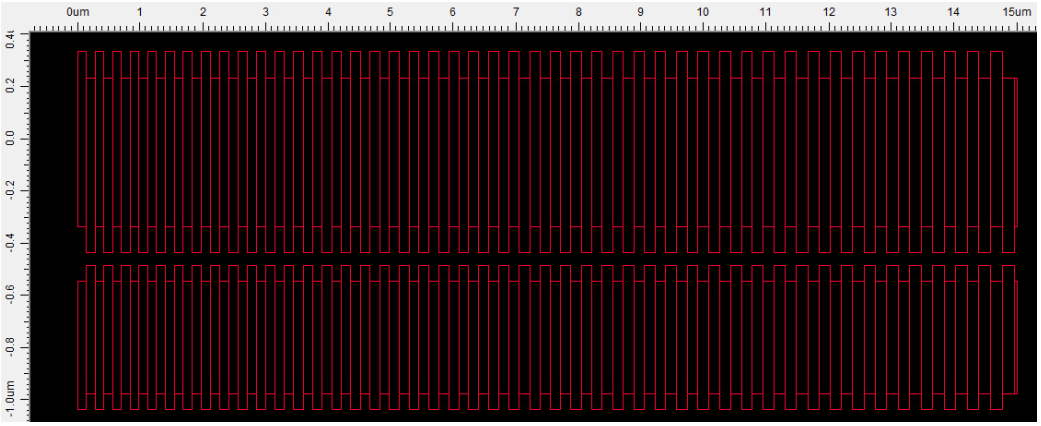


***Figure 18**. CDC BB implementation, with input parameters defined in Table 4, prior to boxing.*

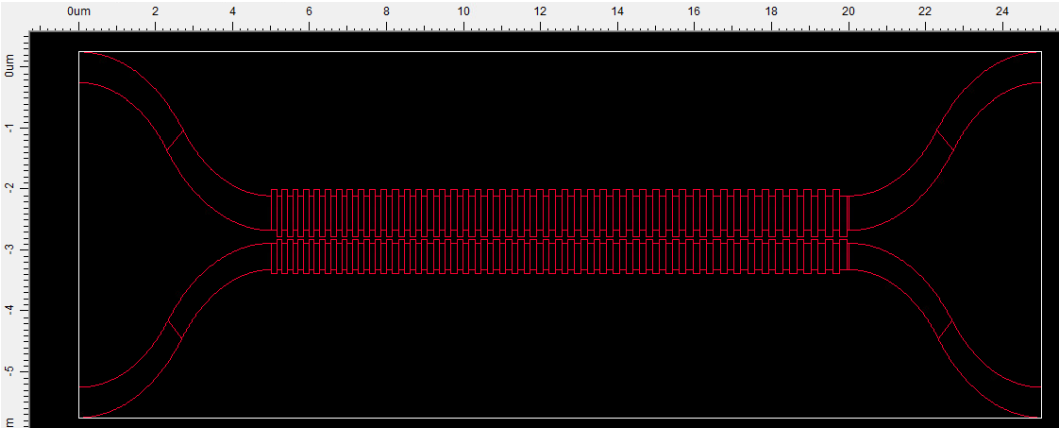


***Figure 19**. CDC BB implementation, with input parameters defined in Table 4 and Table 5, boxed.*

As it might be seen in the image, the CDC presents four different ports, corresponding to each of the S-bend terminals. Of these, three will be used simultaneously for each single CDC. Basically, the input port (IN) will be connected to the first waveguide (Top-Left in Figure 19) while the TH one will be connected to the other end of this waveguide. The other two remaining ports, AD and DP will present the same relative position with respect to the other two ports, as shown in Figure 9. The definition of the ports in OptoDesigner will be done using the abbreviated names, so in order to establish connections to the device, the abbreviations need to be used. It is important to notice that even if the input and output CDCs are different, they can be both implemented by using previously described BB in OptoDesigner, just by choosing correctly the ports to be used. Furthermore, the CDC element can be rotated and mirrored because of OptoDesigner language features, thus allowing for any type of placement that we may need while designing larger elements.
Examples of the filtering properties of the CDC filters used for the band and channels selections are presented in Figure 20.

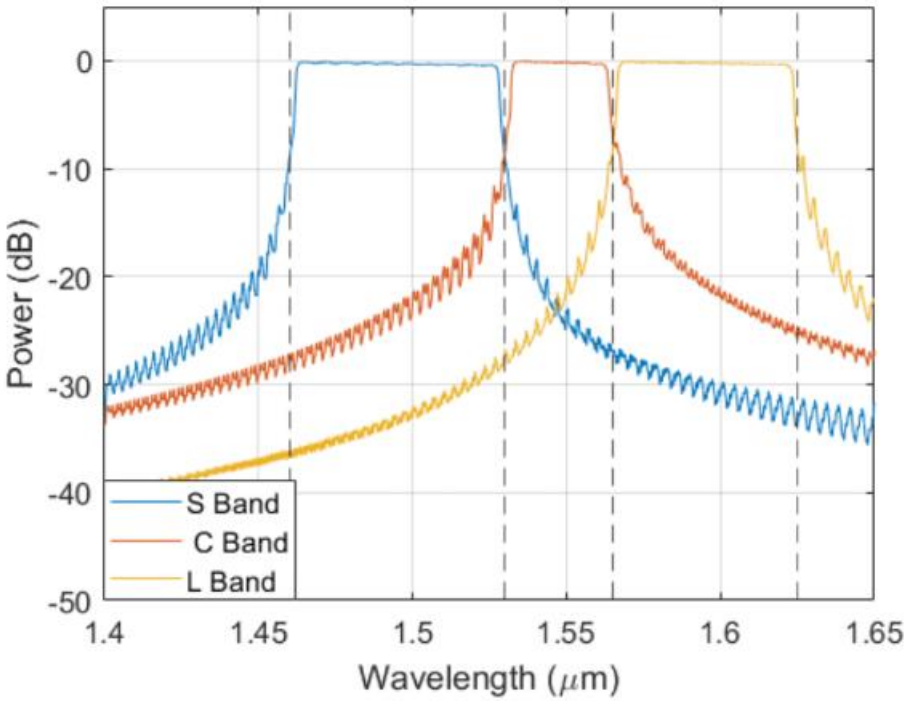


***Figure 20**. Example of CDC responses for the S, C and L bands.*

5.3 MZI design

The MZI constitutes the pillar for switching in this project. There will be 48 of them, and their specific geometrical characteristics will be different depending on the band in which they are operating, due to the necessity of switching between different wavelengths. The geometrical characteristics which define the MZI can be seen in Figure 21, and are summarized as the following:

- Width of the waveguides (W)
- Gap length between the two waveguides (G)
- Length of the phase shift control section (L_P)
- Length of the coupling section (L_C)
- Length of the interphase section (L_I)
- Height between the phase shift control and coupling sections (H)

It becomes important to note that the length of the interphase section does not equal its exact length, but rather the measure in the same axis as the lengths of the other two sections.

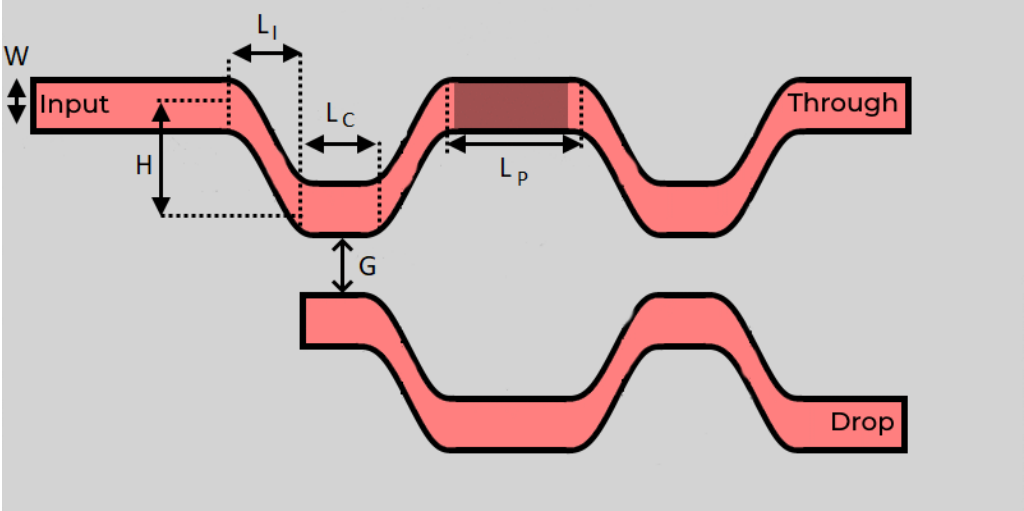


Figure 21. *Schematic of the geometrical characteristics of the MZI.*

For the development within the WSS chip, we must design the values of the different parameters for each of the MZIs, based on CMT. Table 6 below contains the different values applied for each parameter.

W	G	L_P	L_C	L_I	H
0.5 µm	0.3 µm	100 µm	16 µm	20 µm	0.5 µm

Table 6. Parameters used for the design of the MZIs.

5.3.1 Phase shift and thermal control of the MZI

To control the delay introduced by the MZIs it becomes necessary to insert electrical connections for tuning, so that a phase delay is introduced in one of the arms of the MZI. This delay will be produced at the phase shift control section, with respect to the other waveguide. For the purpose of this project, the phase shift control will be implemented by means of thermo-optic phase shift. To do so, a heating line will be introduced. This component will present two electrical ports: at the beginning and at the end of the electrical line, which are used to introduce current and thus perform the tuning. It is composed of three different sections: 2 electrical lines for connection outside the MZI and a heater placed above the arm of the MZI in which the phase shift is induced. The electrical lines and the heater will be made using different materials. For the latter, a heater mask will be applied, whereas the former will have a metallic mask applied. An example of how the heating line is fully implemented on the MZI device can be seen in Figure 24.

5.3.2 Implementing the MZI as a building block

To implement the device, a function layout will be developed, which allows to introduce the geometrical characteristics and return the device as a BB. This function will take all the design parameters that have been explained at the beginning of section 5.3, plus new parameters regarding the boxing process.

The function layout will generate a MZI, composed of two coupling, a phase shift region and three interphase sections. It will start at the coupling region and end after an interphase section. This function layout will again be developed including the boxing process in order to be used in the WSS chip. Therefore, 4 new input parameters will be included: the width (W_b) and length (L) of the box; the width of the waveguides outside the box (W_o), so that the difference can be gradually overcame; and the width of the electrical connections (W_e).

Examples of the implementation of a MZI BB, prior to boxing, and after boxing using OptoDesigner are given in Figure 22 and Figure 23 respectively. These correspond to the first and second trial of the MZI in the code at Appendix D. Their values are presented in Table 7, where the MZI device parameters are in orange cells, whereas the boxing ones are in cyan cells.

W	G	L_P	L_C	L_I	H	W_b	W_o	L	W_e
1 µm	0.3 µm	30 µm	5 µm	8 µm	0.5 µm	6 µm	0.5 µm	100 µm	5 µm

Table 7. Parameters used for the MZI trials. In orange MZI device parameters. In cyan, boxing parameters.

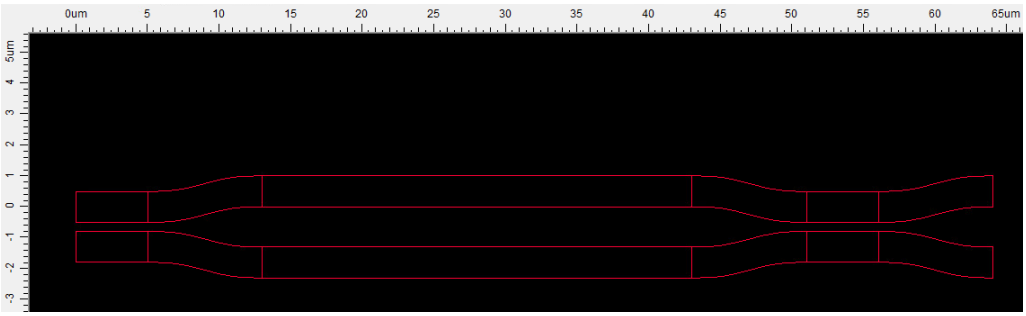


***Figure 22**. MZI BB implementationI, with input parameters defined in Table 7, prior to boxing.*

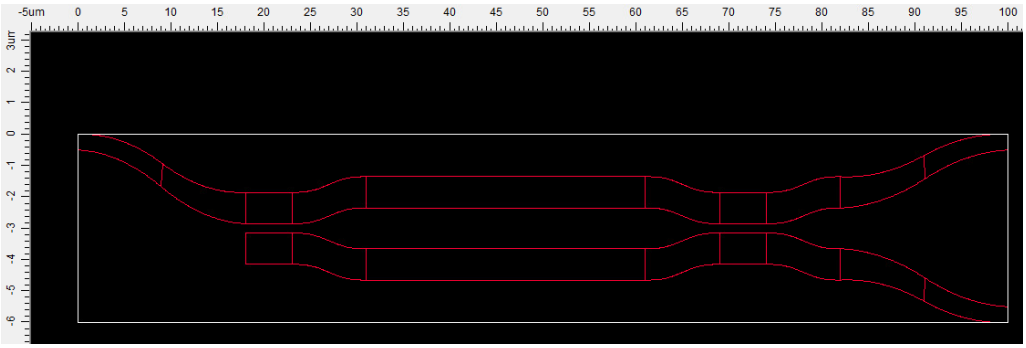


***Figure 23**. MZI BB implementation, with input parameters defined in Table 7, after boxing.*

As it can be seen in Figure 23, the boxed device possesses 3 different ports, which correspond respectively to the ones defined in Figure 10. For the purpose of this trial,

the IN port corresponds to the top-left one; the TH port is connected to the other end of the same waveguide; and the DP port is the one on the waveguide with no connections on the other end (lack of an S-bend guide to connect it outside). In order to connect the elements to the MZI in OptoDesigner, the abbreviations need to be used. All the optical components for connections and the ones for the MZI device itself will again be designed by default using the Demofab MCS "*mcsSOI_DEEP*". A different MCS may be used if it is explicitly specified at the input parameters.

After boxing the MZI, the heating line can be introduced at another function layout, which takes into account the width introduced for boxing in order to adjust the size of the electrical line. This introduced electrical line will range from being just above the phase control section to positioning at the border of the box, permitting the connection of outer electrical lines to the heating section. It is important to notice that each section of the period of the electrical line will be made by a different type of material. These are respectively a heater, implemented by the Demofab MCS "*mcsVia_metal*"; and a metallic line, implemented by the Demofab MCS "*mcsHEATER_SOI*". For the electrical line there will be two ports at the end and the beginning of the line in order for current to flow, with a metallic surface. To refer to them and make connections in OptoDesigner, the abbreviations "*VO*" and "VI", referring to the electrical ports need to be used. The size of these two may be chosen by defining a value at the input parameters. By doing so, the metallic line will also change its width. For a better comprehension, Figure 24 is presented, whose code may be found on the third trial of the MZI trials implementation section, in Appendix D.

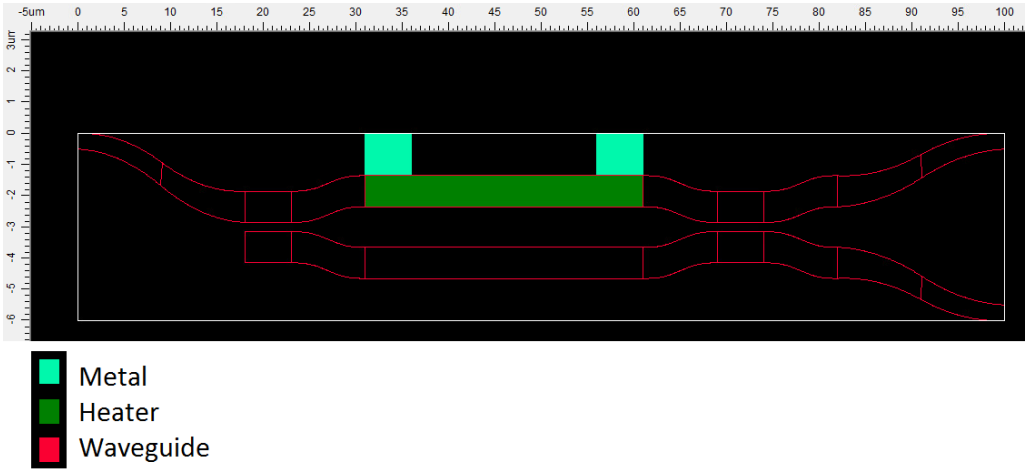


***Figure 24**. MZI BB implementation, with input parameters defined in Table 7, boxed and with heating included.*

6 WSS full design with OptoDesigner

A schematic of the layout of the device was presented in Figure 17. This layout shows the approach towards element positioning in the chip, and it will be followed for the development of the switch in OptoDesigner. Again, for simplicity, the design will be completed by creating smaller objects, which will be finally merged together to form the full WSS chip. First, the channel blocks will be created, forming 8 of them a band block. Later, 3 of these band blocks will be merged, and finally ports and the connections to the elements will be added. Once all the optical elements are placed, the electrical connections will be developed, in order to allow the routing of the different channels.

In order to implement, construct and represent the blocks and the full WSS chip in OptoDesigner, 2 pseudo-elements will be developed to act as MZIs and CDCs. The reason behind the introduction of these elements is the high computational power and time required to develop these, along with easier representation due to simpler shapes. Therefore, these will be used in the different stages for developing the WSS. Only after the completion of the development of the full WSS chip, the real elements will be used to represent the real full design.

6.1 Pseudo-elements

6.1.1 Pseudo-MZI

The real MZI after the boxing process and with the presence of the heating lines presents a shape as seen in Figure 24. This element contains 3 different MCS, with diverse shapes and sizes. This fact and the relative thinness of the device when compared to the full layout adds difficulty to distinguish the element while representing the full layout. Thus, the pseudo-MZI will be introduced.

To develop the pseudo-MZI, a function layout will be created, which mirrors the definition and the input parameters of the real MZI layout. Basically, this function will create a square which simulates the box, along with two other rectangles within it, representing the waveguides of the MZI, whose widths coincide with those. The larger rectangle occupies the full length of the box, and acts as the waveguide connected to IN and TH ports at its ends. To distinguish between them, a long trapezoid has been developed at the IN port. On the other side of the box there is a rectangle, representing the waveguide which contains the DR port. This one is half the length of the other waveguide. Also, two rectangles will be created using the MCS "*mcsVia_metal*", which represent the electrical ports, and are placed at the same position as these. To facilitate a better understanding, an example of the use of a pseudo-MZI is presented in Figure 25. The code to develop a pseudo-MZI BB can be consulted in the trials section of the pseudo-MZI, in Appendix D.

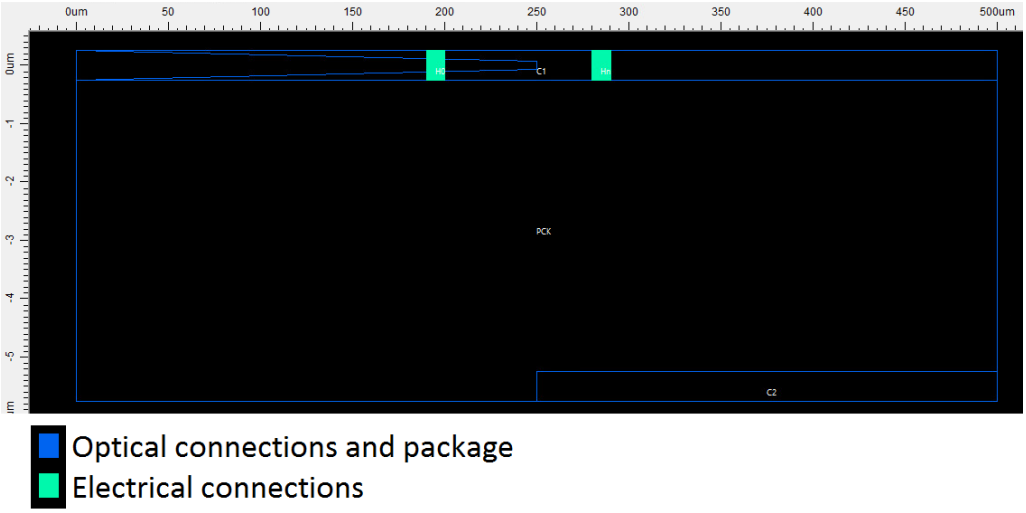


***Figure 25**. Example of a pseudo-MZI.*

6.1.2 Pseudo-CDC

The real CDCs are complex devices, which are made by the union of different smaller components in OptoDesigner. For this specific project, a single CDC may have to implement and merge up to 25000 elements, which is computationally expensive. Therefore, to reduce the time for developing the pseudo-MZI has been created. Furthermore, the representation in the following layout will be also simpler, since it presents a rectangular shape.

The pseudo-CDC function layout mirrors the real CDC one, being able to take its exact same input parameters, having the same definition as the real one, and producing a component with the same boxing. Again, there are 2 rectangular rectangles that represent the waveguides, and their widths equal that of the waveguides to which the CDC is linked. To distinguish the port layout, once more, a long trapezoid figure has been introduced at the IN port. On the other side there will be the TH port, and the other waveguide will possess the DP and the AD ports, being this last one located on the opposite corner of the IN port. For a better comprehension, Figure 26 implements a pseudo-CDC BB in OptoDesigner. Also, the code for developing this figure can be consulted in Appendix D, at the trials section of the pseudo-CDC

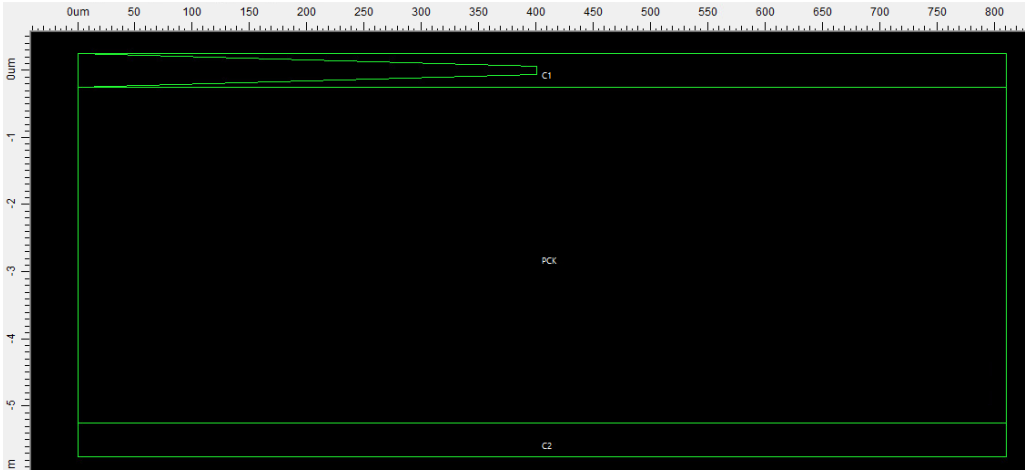


***Figure 26**. Example of a pseudo-CDC.*

6.2 Channel blocks

The channel blocks are the simplest blocks containing several optical components linked. Twenty-four of them will be used to compose the WSS chip, and they implement all the necessary components to treat a single channel. In OptoDesigner, there are three types of these, which are defined by the channel that is being treated within the band. These are the initial channel blocks, the middle ones and the final one.

The initial channel block is the simplest one, and it is formed by an input channel CDC (demultiplexer) connected to 2 MZIs in cascade for switching. In it, the connections to the next channel block are also developed. Basically, the CDC connects through its DP port to the first MZI via its IN port. The 2 MZIs cascaded work as a 3-output switch block, whose ports are defined in Figure 11. For simplicity, the definition of the ports given the MZIs as a block will be used. P1 (TH1), P2 (TH2) and P3 (DP2) will connect in this case to the 3 different output channel CDCs in the next channel. For a better comprehension, Figure 27 implements the initial channel block of the S-Band. Also, the code implementing it can be consulted in Appendix E, at the first trial of the channel and band blocks implementation, which generates this specific type.

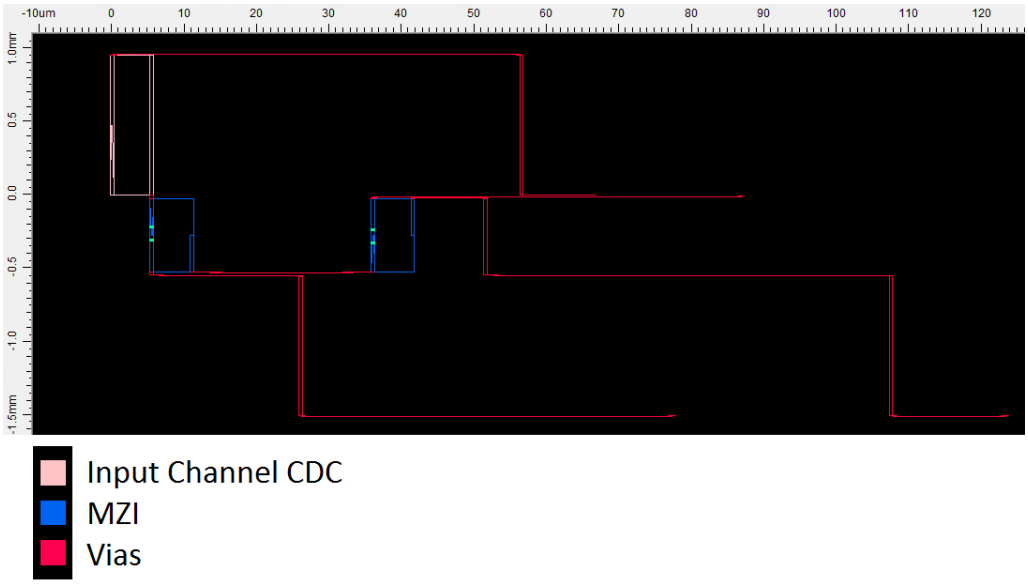


***Figure 27**. Initial channel block (of the first band block).*

The middle channel block is a bit more complex than the initial one, since it contains all the elements of the initial one, plus 3 output band CDCs. These are connected to the P1, P2 and P3 ports of the MZIs via their AD ports. As in the previous block, the connections to the next channel block are also included, but this time the output channel CDCs are the one connecting to it. In all the cases, these will connect to the following channel via its TH port, and will receive the signal from the preceding channel via their IN port. For a better comprehension, Figure 28 implements a middle channel block of the S-Band. Also, the code implementing it can be consulted in Appendix E, at the second trial of the channel and band blocks implementation section, which generates this specific type.

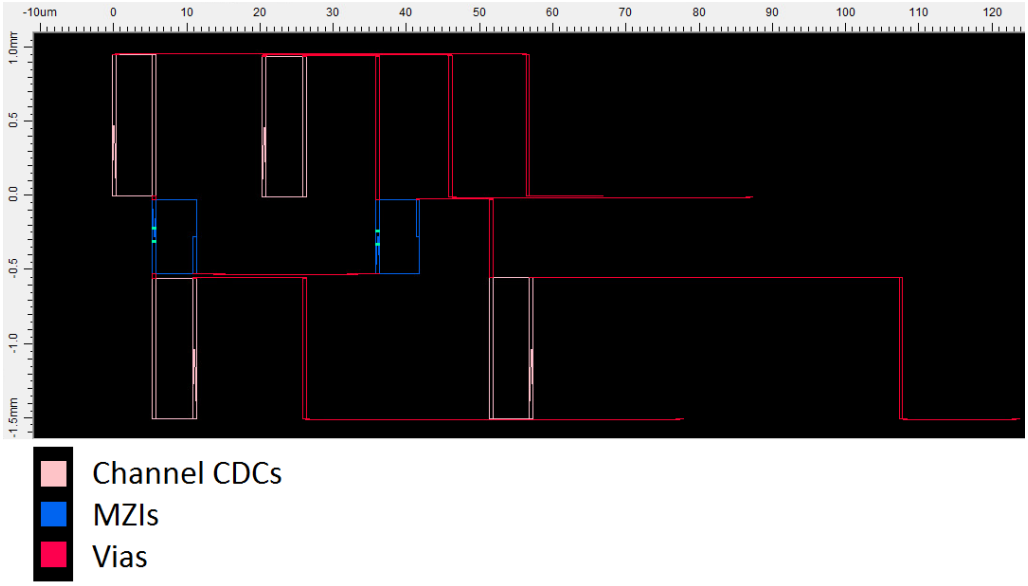


***Figure 28**. Middle channel block (of the first band block).*

Last, the final channel block presents a very similar layout to the middle one, presenting the same elements. The only difference resides on the connections to the next channel block, which are not present because of its position as the last one. For a better comprehension, Figure 29 implements the final channel block of the S-Band. Also, the code used to create this block is located in Appendix E, at the third trial of the channel and band blocks implementation section, which generates this specific type.

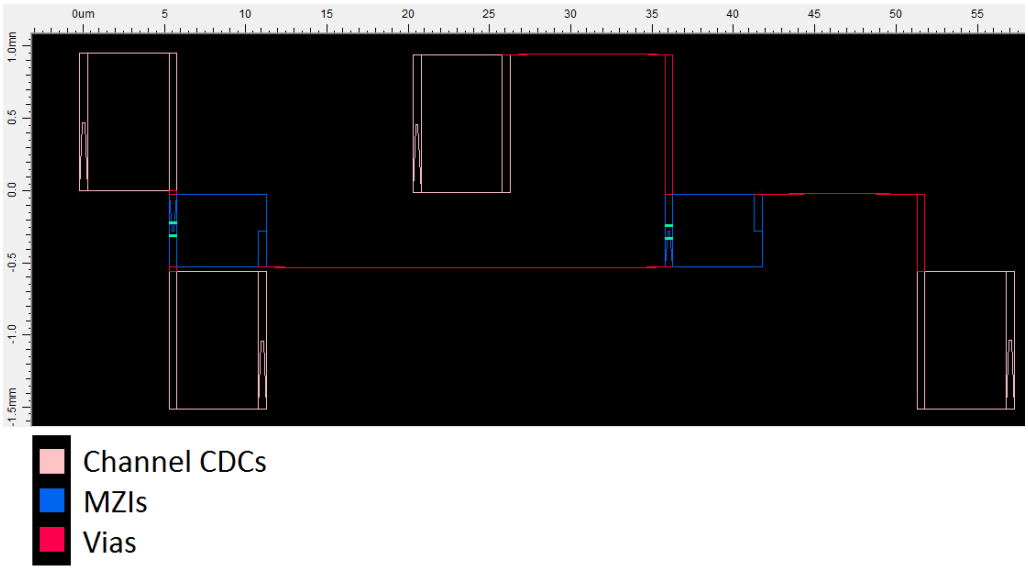


***Figure 29**. Final channel block (of the first band block).*

6.3 Band blocks

Band blocks are more complex than the previously introduced channel blocks, and in fact, they are formed by 8 independent channel blocks. Of these, the first one is of initial type, the last one of final type and the rest between them are of middle type. Three of these band blocks will be used to compose the WSS chip, each implementing all the necessary components to handle a single band. Analogously to the channel blocks, three types of these will be designed in OptoDesigner. These are the initial band block, the middle band block and the final band block.

The initial band block is formed by an input band CDC connected to the input CDC of the first channel block. To do so, the input band CDC connects through its DP port to the IN port of the input channel CDC. Now, all the 8 channels blocks will be linked together, and finally at the last channel there will be three connections to the three output band CDC, used for multiplexing the bands. These will merge the obtained signals with that of the other two bands. The connections between the output channel CDCs and the output band CDCs are done via the TH port of the former and the AD port of the latter. Connections to the next band block are also done via the output band CDCs, which are linked via their IN ports to the TH of the output band CDCs of the following band. For a better comprehension, Figure 30 implements the initial band block. Also, the code implementing it can be consulted in Appendix E, at the fourth trial of the channel and band blocks implementation section, which generates this specific type.

It is important to notice that the elements within the initial band block will all be defined to work for the S-band, with the exception of the of the output band CDC, which works for the L-band.

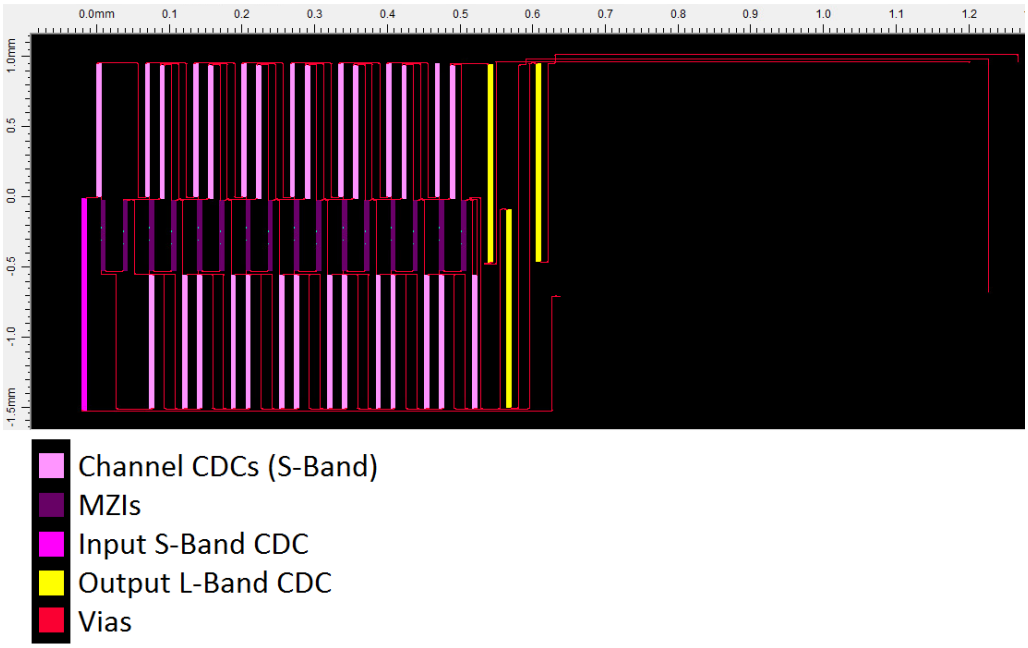


***Figure 30**. Initial band block.*

The middle band block is formed by the same type of components as the initial one, but there are two main differences. The first on resides on the band at which the components are operating, being in this case the C-band for all the different components. The other difference lies on the connections to the following band, which are attached within the block. Now, the output band CDCs are connected to the output channel CDCs of the last channel, the former through their IN port and the latter through their TH port. To visualize it, Figure 31 implements the middle band block. Also, the code implementing it can be consulted in Appendix E, at the fifth trial of the channel and band blocks implementation section, which generates this specific type.

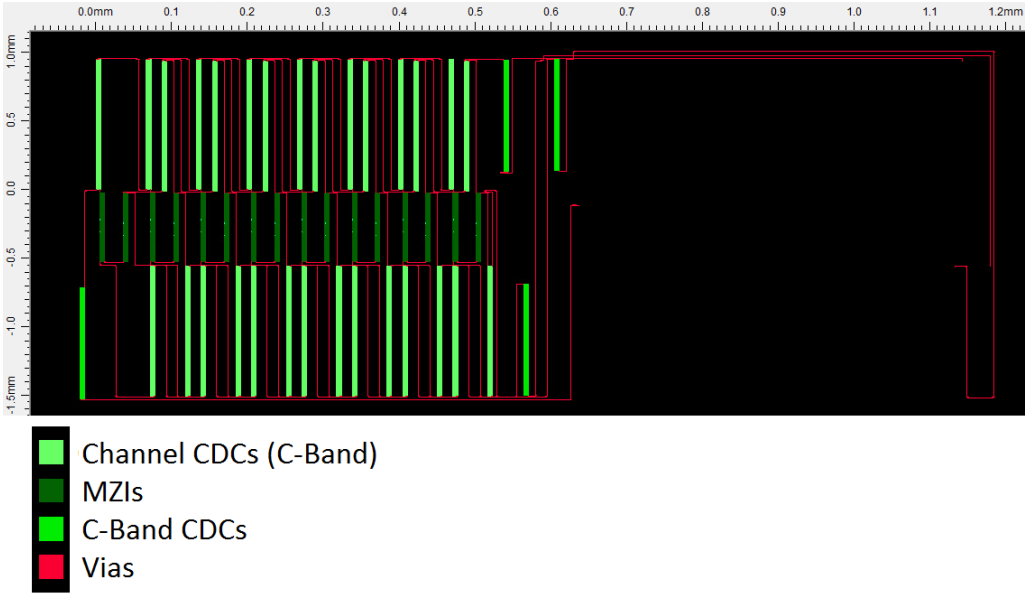


***Figure 31**. Middle band block.*

The final band block, unlike the other two, does not present output band CDCs, since the signal has been demultiplexed for the other 2 bands already. All the elements operate within the L-band. To visualize it, Figure 32 shows the final band block. Also, the code implementing it can be consulted in Appendix E, at the sixth trial of the channel and band blocks implementation section, which generates this specific type.

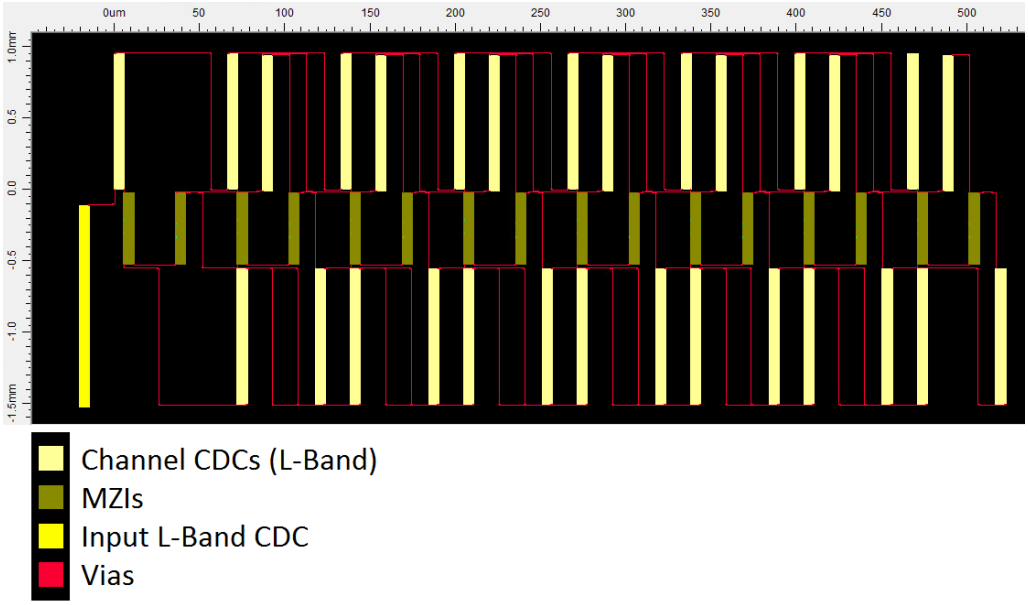


***Figure 32**. Final band block.*

6.4 Connections to terminal ports

Now that the 3 band blocks are created and connect to each other, the terminal ports, which connect the WSS device with the outside, can be set within the layout. To do so the Demofab component *demofabFiberCoupler* will be used. The terminal ports will be placed close to the initial band block in the chip, on the opposite side of the final band block location. They will be situated at the bottom, on the side in which the IN port of the input band CDCs are located. The distance between the ports is 127 µm, with first another 127 of spacing at the bottom, from the very last waveguide, which is connecting to Testing #4 port. In any case and since they occupy less than half of the chip height, they can be recognized by their relative position to the corner of the device. To summarize the information relative to the terminal ports and their relative positions, Table 8 is presented.

Port Name	Position on device	Function	Direction	Connected to
Input	2	Receives the input signal	Inward	Input S-Band CDC (IN port)
Output #1	6	Transmits routed output signal	Outward	Output L-Band CDC #1 (TH port)
Output #2	4	Transmits routed output signal	Outward	Output L-Band CDC #2 (TH port)
Output #3	5	Transmits routed output signal	Outward	Output L-Band CDC #3 (TH port)
Testing #1	3	Tests behavior at the end of the S-Band	Outward	Input channel CDC #8 (S-Band) (TH port)
Testing #2	7	Tests behavior at the end of the C-Band	Outward	Input channel CDC #8 (C-Band) (TH port)
Testing #3	8	Tests behavior at the end of the L-Band	Outward	Input channel CDC #8 (L-Band)
Testing #4	1	Tests behavior at the beginning of the L-Band	Outward	Input L-Band CDC (TH port)

Table 8. Terminal ports specifications. The number in the "Position on Device" column indicate the position starting from the components closer to the input corner.

6.5 WSS optical layout

Once all the different components of the 3 band blocks have been connected to the desired terminal ports, we obtain the full optical layout of the WSS chip. Within it, as previously stated, the different elements will work for different bands. To visualize the optical layout, Figure 33 is shown, where the connections between the different band blocks and to the terminal ports can be seen. The code implementing it can be consulted in Appendix E, in the first trial of the WSS layout implementation section.

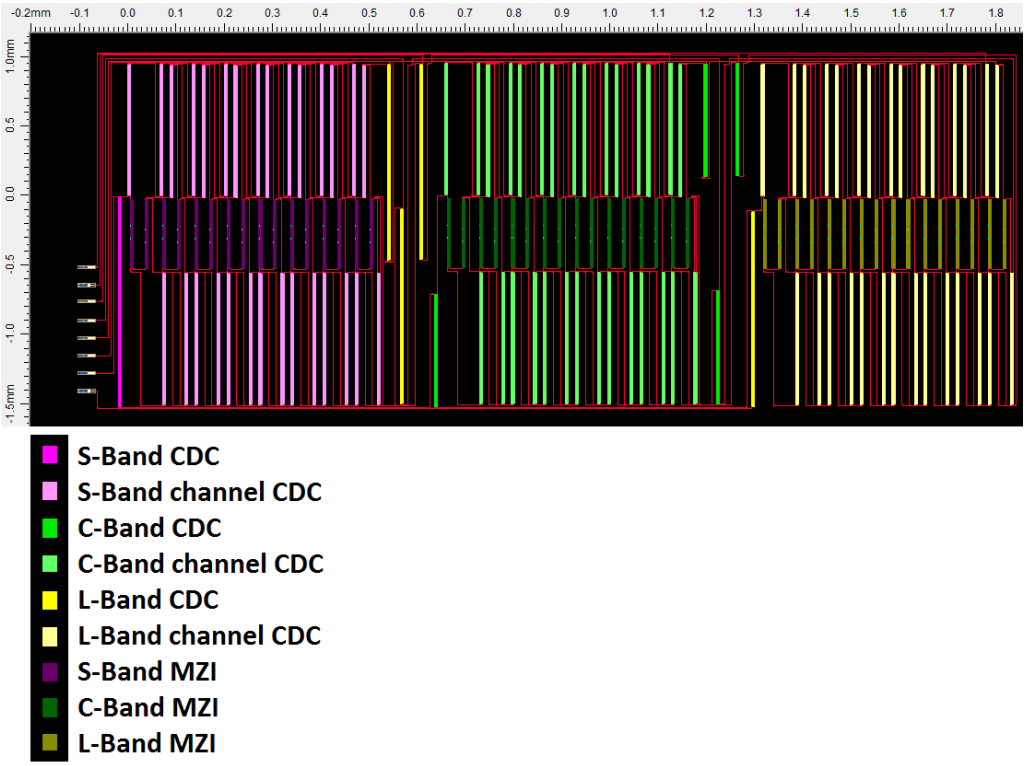


***Figure 33**. WSS chip optical layout.*

6.6 Electrical connections for switching

Once the optical layout has been designed, the last step towards the full mask realization can be performed. This consists on placing the electrical connections that allow for the switching of MZIs. To do so, two different elements will be used: electrical pads and electrical vias.

6.6.1 Electrical pads

In order to perform the switching, it is needed an interface that allows the switching signal to be transmitted to the WSS. For this purpose, electrical pads are introduced, which will be used as electrical ports. In OptoDesigner there are a few components already designed for this, but since we are designing following the Demofab rules and components, the elements defined by the function *demofabDCPad_bidir* will be used

as electrical pads. The size of these is 100 x 100 µm, and there is a bounding box of 220 x 220 µm in which no other elements can be placed.

There will be 49 of them, one for each MZI plus one used as a common ground line. They all will be placed on the opposite side of the optical port, with 7 rows and 7 columns, in order to fit these and their respective connections. There will be a spacing of 100 µm between the pads within a column, and a spacing of 325 µm between columns, which comprises the bounding box and the different electrical connections that will be placed between the columns.

6.6.2 Vias from electrical pads to MZIs

The electrical vias are placed above the optical components. Thus, these can traverse any point regardless of the optical layout that has been designed, with the exception of the MZIs, which contain electrical connections of their own. The width of these will be 10 µm, because it allows to have a current of approximately 10 mA, which is enough to deliver a π phase shift to the MZI. The separation between the electrical lines will be 5 µm. There are two types of connections: The signal lines and the ground line. To the first type, a specific voltage will be applied, which in turn generates a phase shift for the MZI. The ground line, on the other hand, will be used for voltage reference.

The signal lines will be point to point. These will go from the specific pad to its corresponding MZI, first traversing the zone in which the pads are connected to the lower part of the optical layout, just below the MZIs. From there, each line will follow a horizontal path and then turn vertically to reach its MZI. For simplification of the electrical layout, the port will correspond to the lower one of the MZI.
On the other hand, the ground line will be point to multipoint. Basically there will be a common ground line which is connected to the first pad on top. This line will first come closer to the MZIs, and then turn to follow a horizontal path prior to the MZIs position. This line will follow until the last horizontal position of the last MZI is achieved, and meanwhile small vertical lines will connect to this common ground line, at the position of each interferometer. To visualize all the electrical connections, Figure 34 is presented, where these can be seen along with the CDCs and MZIs that compose the layout. The code implementing it can be consulted in Appendix E, in the second trial of the WSS layout implementation section

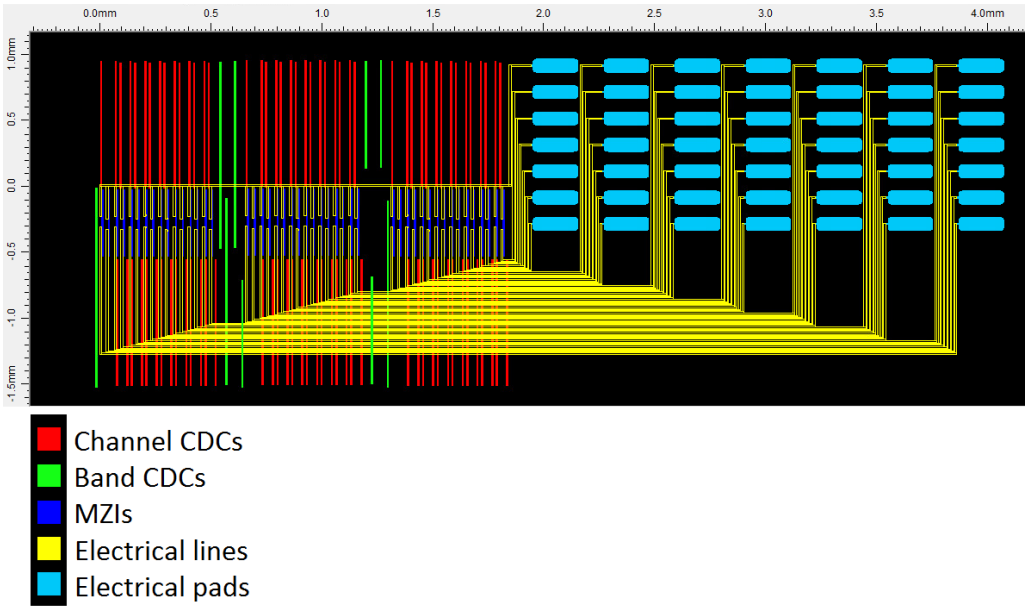


***Figure 34**. WSS chip electrical connections layout.*

Now, to understand the electrical pinout, Table 9 is presented, which summarizes the information relative to all of the electrical pads. It is important to remark that the first MZIs within the channels will either switch the signal to its output band CDC or to the next MZI of that channel. As a consequence, the control of that MZI will only affect the functioning when the first one is controlled to pass the signal to the second one.

Port name	Column position	Row position	Function	Connected to
GND	1	1	Ground	-
S1	1	2	Controls L-Band Channel #8 MZI #2	L-Band Channel #8 MZI #2 (VI port)
S2	1	3	Controls L-Band Channel #8 MZI #1	L-Band Channel #8 MZI #1 (VO port)
S3	1	4	Controls L-Band Channel #7 MZI #2	L-Band Channel #7 MZI #2 (VI port)
S4	1	5	Controls L-Band Channel #7 MZI #1	L-Band Channel #7 MZI #1 (VO port)
S5	1	6	Controls L-Band Channel #6 MZI #2	L-Band Channel #6 MZI #2 (VI port)
S6	1	7	Controls L-Band Channel #6 MZI #1	L-Band Channel #6 MZI #1 (VO port)
S7	2	1	Controls L-Band Channel #5 MZI #2	L-Band Channel #5 MZI #2 (VI port)

S8	2	2	Controls L-Band Channel #5 MZI #1	L-Band Channel #5 MZI #1 (VO port)
S9	2	3	Controls L-Band Channel #4 MZI #2	L-Band Channel #4 MZI #2 (VI port)
S10	2	4	Controls L-Band Channel #4 MZI #1	L-Band Channel #4 MZI #1 (VO port)
S11	2	5	Controls L-Band Channel #3 MZI #2	L-Band Channel #3 MZI #2 (VI port)
S12	2	6	Controls L-Band Channel #3 MZI #1	L-Band Channel #3 MZI #1 (VO port)
S13	2	7	Controls L-Band Channel #2 MZI #2	L-Band Channel #2 MZI #2 (VI port)
S14	3	1	Controls L-Band Channel #2 MZI #1	L-Band Channel #2 MZI #1 (VO port)
S15	3	2	Controls L-Band Channel #1 MZI #2	L-Band Channel #1 MZI #2 (VI port)
S16	3	3	Controls L-Band Channel #1 MZI #1	L-Band Channel #1 MZI #1 (VO port)
S17	3	4	Controls C-Band Channel #8 MZI #2	C-Band Channel #8 MZI #2 (VI port)
S18	3	5	Controls C-Band Channel #8 MZI #1	C-Band Channel #8 MZI #1 (VO port)
S19	3	6	Controls C-Band Channel #7 MZI #2	C-Band Channel #7 MZI #2 (VI port)
S20	3	7	Controls C-Band Channel #7 MZI #1	C-Band Channel #7 MZI #1 (VO port)
S21	4	1	Controls C-Band Channel #6 MZI #2	C-Band Channel #6 MZI #2 (VI port)
S22	4	2	Controls C-Band Channel #6 MZI #1	C-Band Channel #6 MZI #1 (VO port)
S23	4	3	Controls C-Band Channel #5 MZI #2	C-Band Channel #5 MZI #2 (VI port)
S24	4	4	Controls C-Band Channel #5 MZI #1	C-Band Channel #5 MZI #1 (VO port)
S25	4	5	Controls C-Band Channel #4 MZI #2	C-Band Channel #4 MZI #2 (VI port)
S26	4	6	Controls C-Band Channel #4 MZI #1	C-Band Channel #4 MZI #1 (VO port)
S27	4	7	Controls C-Band Channel #3 MZI #2	C-Band Channel #3 MZI #2 (VI port)
S28	5	1	Controls C-Band Channel #3 MZI #1	C-Band Channel #3 MZI #1 (VO port)
S29	5	2	Controls C-Band Channel #2 MZI #2	C-Band Channel #2 MZI #2 (VI port)
S30	5	3	Controls C-Band Channel #2 MZI #1	C-Band Channel #2 MZI #1 (VO port)

S31	5	4	Controls C-Band Channel #1 MZI #2	C-Band Channel #1 MZI #2 (VI port)
S32	5	5	Controls C-Band Channel #1 MZI #1	C-Band Channel #1 MZI #1 (VO port)
S33	5	6	Controls S-Band Channel #8 MZI #2	S-Band Channel #8 MZI #2 (VI port)
S34	5	7	Controls S-Band Channel #8 MZI #1	S-Band Channel #8 MZI #1 (VO port)
S35	6	1	Controls S-Band Channel #7 MZI #2	S-Band Channel #7 MZI #2 (VI port)
S36	6	2	Controls S-Band Channel #7 MZI #1	S-Band Channel #7 MZI #1 (VO port)
S37	6	3	Controls S-Band Channel #6 MZI #2	S-Band Channel #6 MZI #2 (VI port)
S38	6	4	Controls S-Band Channel #6 MZI #1	S-Band Channel #6 MZI #1 (VO port)
S39	6	5	Controls S-Band Channel #5 MZI #2	S-Band Channel #5 MZI #2 (VI port)
S40	6	6	Controls S-Band Channel #5 MZI #1	S-Band Channel #5 MZI #1 (VO port)
S41	6	7	Controls S-Band Channel #4 MZI #2	S-Band Channel #4 MZI #2 (VI port)
S42	7	1	Controls S-Band Channel #4 MZI #1	S-Band Channel #4 MZI #1 (VO port)
S43	7	2	Controls S-Band Channel #3 MZI #2	S-Band Channel #3 MZI #2 (VI port)
S44	7	3	Controls S-Band Channel #3 MZI #1	S-Band Channel #3 MZI #1 (VO port)
S45	7	4	Controls S-Band Channel #2 MZI #2	S-Band Channel #2 MZI #2 (VI port)
S45	7	5	Controls S-Band Channel #2 MZI #1	S-Band Channel #2 MZI #1 (VO port)
S47	7	6	Controls S-Band Channel #1 MZI #2	S-Band Channel #1 MZI #2 (VI port)
S48	7	7	Controls S-Band Channel #1 MZI #1	S-Band Channel #1 MZI #1 (VO port)

Table 9. Electrical pinout of the WSS chip. The numbers indicating the position in “row position” and “column position” go from top to bottom and from right to left.

6.7 Implementation of the full WSS mask

After the design of the optical and electrical layout has been completed, the full layout of the chip can be implemented. The visibility is reduced in this case because of the relative small width of MZIs and CDCs in comparison to the size of the full layout, and the use of the same MCS for the vias, MZIs and CDCs. Both the real full mask layout and the one composed with the pseudo-MZIs and pseudo-CDCs are

presented in Figure 35 and Figure 36 respectively. Now that the full mask layout has been implemented, the last step may be taken, which consists of fabricating the device, for which small changes might still need to be introduced, due to the different Design Checking Rules (DRC) that may be introduced by the foundries.

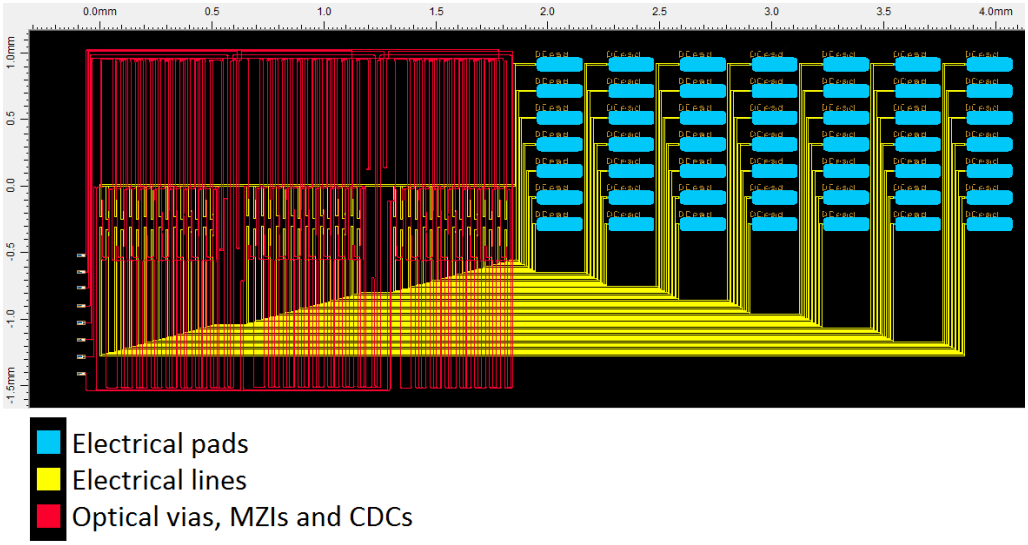


***Figure 35**. Full mask layout of the designed WSS chip.*

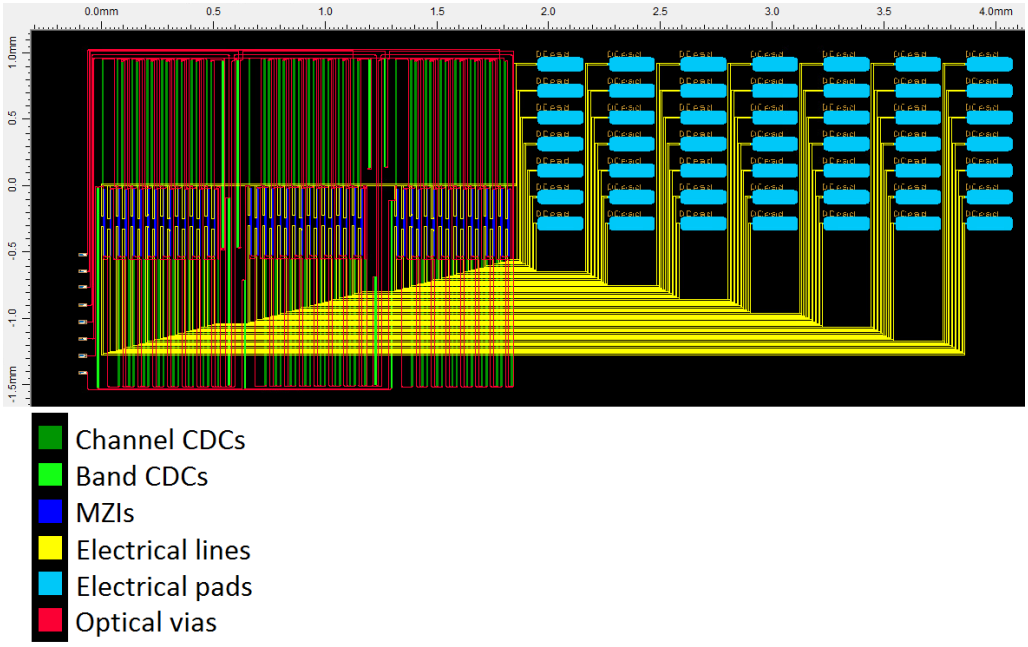


***Figure 36**. Mask layout of the designed WSS chip with pseudo-MZIs and pseudo-CDCs.*

7 Fabrication of the chip

Due to our project's scope, it is necessary to find Multi-Project Wafers (MPW) Providers to build the designed telecom photonics chip. In this section, several options of providers for fabricating the device are showcased. Now, to select the foundry that suits this project the best, it becomes necessary to consider distinct characteristics, such as the price, the production schedule, the technologies offered and the supported software to design the chip. In this specific case the software OptoDesigner and the Process Design Kit (PDK) from Synopsys have been used for the creation of the chip layout, and thus the compatibility with it will be crucial.

In order to compare all the different alternatives, Table 10 contains recompiled different MPW solutions from different providers. The table is not extensive, and it comprises only information publicly available through online sources.

Foundry	Minimum Price	Maximum Price	Minimum Area	Maximum Area	PDK and software
Imec	6400€	165000€	2.5 x 2.5 mm	10.45 x 10.45 mm	Synopsys-based
IHP	3800€/mm^2	8000€/mm^2	10 mm^2	-	Cadence-based
LioniX	9350€	24600€	10 x 10 mm	10 x 20 mm	Synopsys-based
IMB-CNM	7500€	15000€	5 x 5 mm	5 x 10 mm	Synopsys-based
SMART Photonics	-	-	-	-	Based on: Synopsys, Lumerical, Nazda, ANSYS, Luceda
CORNERSTONE	5250£	40250£	5.5 x 4.9 mm	11.47 x 4.9mm	Luceda-based; GDSII format
AIM Photonics	20000$	120000$	25 mm^2	50 mm^2	Synopsys-based
CEA-LETI	-	-	-	-	Synopsys-based
LIGENTEC	-	-	-	-	Based on: ANSYS, Lumerical, Luceda, Synopsys, Siemens; GDS format and KLayout
VTT	7200€	46000€	5 x 4.75mm	20 x 19.5mm	private-owned

					PDK library

Table 10. Comparison between different MPW solutions by providers.

Different MPW solutions have been introduced. Now they will be discussed in the following subsections for further details, and a broker will be also introduced.

7.1 Europractice

Europractice constitutes a consortium of research organizations that provides European industry and academia with a platform to develop electronic circuits and systems [36]. They present discounts for academic and research institutions subscribed to it. Several foundries are presented for MPW runs for photonic projects: AMF, CEA-LETI, CORNERSTONE, GlobalFoundries, IHP, imec, LioniX and Teem Photonics. Each of them presents different prices and different run schedules.

7.2 Imec

Imec is a semiconductor fabrication company with multiple fabrication locations. Their prices range from 6700€ to 11000€ for the tiniest chips, and it can get up to 165000€ for their biggest sized chips. These sizes are comprised between 2.5 x 2.5 mm to 10.45 x 10.45 mm [37]. In any case, the price may be lowered by purchasing through Europractice, up to 6400€ [38]. It is important to remark that the use of Synopsys PDKs is supported.

7.3 IHP

IHP is a non-university research establishment institutionally funded by the German federal and state governments. They provide photonics solutions in the C/O band (i.e., from 1.25 µm to 1.55 µm approximately) along with 0.25 µm CMOS technology and NPN HBTs within the chip. It is also important to remark that it includes both active and passive photonic devices. This aligns with the specific characteristics of the desired chip since it will need to include both an electrical part and a photonics part with both active and passive devices. Regarding the MPW prices, it will range from 3800€/mm^2 to 8000€/mm^2, with a minimum surface of 10mm^2 [39]. For our specific project, the biggest drawback is that OptoDesigner is not supported by this establishment, and instead they provide Cadence-based software support.

7.4 LioniX

LioniX is a Dutch microsystems provider, which specializes in photonic and MEMS devices. They provide MPW photonic chips that are optimized for visible light (400 nm to 700 nm), Near IR (0.85 µm) and C-band (1.55 µm). The latter encompasses a wavelength that suits this project. Their PDKs are available for Synopsys, which also suits it [20]. Regarding the prices, they range from 9350€ to 24600€ for 4 samples sized either 10 x 10 mm or 10 x 20 mm. The purchase price depends on whether Europractice discount is applied [38].

7.5 IMB-CNM

IMB-CNM is a Spanish research center for the development of new micro and nano technologies. They offer photonics solutions for visible and mid infrared wavelengths (400 nm to 700 nm and 2.5 µm to 25 µm respectively). A PDK for Synopsys software is offered. Regarding the prices, they start at 7500€ for 20 dies containing each 7 cells of 5 x 5 mm, and 15000€ for the large cells, which are sized 5 x 10 mm [40].

7.6 SMART Photonics

SMART Photonics is a Dutch photonics manufacturing supplier. It provides PDKs for several different chip designing software, such as OptoDesigner and OptSim from Synopsys, and software from Lumerical, Nazca, Luceda and ANSYS. For the design and MPW production it becomes necessary to sign a non-disclosure agreement with the company [41].

7.7 CORNERSTONE

CORNERSTONE is a British open-source silicon photonics foundry. Their PDKs are open source in GDSII format, so they can be used with any type of designing software. An important advantage if this foundry is chosen is that being open source, more information and help about the design process may be found online, and the possible incompatibilities will be more easily overcome. The obtained chips may be passive or active, ranging from 5250£ for passive devices of 5.5 x 4.9 mm to 40250£ for an active photonics device of 11.47 x 4.9 mm [42].

7.8 AIM Photonics

AIM Photonics is an American R&D center which focuses on the manufacturing of integrated photonic circuits. To use their MPW manufacturing services it is necessary to sign a non-disclosure and a payment agreement with the company, and to fill in a US export control form. An important advantage covered while purchasing their services is the presence of a help desk for design, wafer production, test, assembly, and packaging, along with code, videos, and material online. Furthermore, their PDK can be used with OptoDesigner Software. The semiconductors work with optical wavelengths ranging from 700 nm to 1.625 µm and the price ranges from 20000 US$ for 20 passive chips sized 25mm^2 to 120000 US$ for 20 active chips sized 50mm^2 [43].

7.9 CEA-LETI

CEA-LETI is a branch of CEA: a French public institution for research. This specific branch is thought for technology and microelectronics. The optical wavelengths of the chips made by means of their technology are in the visible, infrared and THz regions. Both active and passive chips are allowed, and there is a specific PDK for OptoDesigner [44].

7.10 LIGENTEC

LIGENTEC is a Swiss company which works with silicon-nitride based Photonic Integrated Circuits from the visible to the infrared wavelengths. They have designed PDKs for different chip designing programs, such as the ones from Lumerical, KLayout, Calibre, INTERCONNECT and OptoDesigner. Helpdesk is offered for the design of circuits while working with them [45].

7.11 Applied Nanotools

Applied NanoTools is a Canadian company which offers integrated photonics foundry services. They work with both Silicon and SiN Photonics. Their MPW runs are scheduled once every six to eight weeks. Prices and their tailor-made PDK are private, and it is necessary to have an account with billing address in order to obtain both [46].

7.12 VTT

VTT is a finish company, which designs 1.2 µm to 4 µm Silicon Photonics devices. The wavelengths of use range from 1.2 to 6 µm, so it is in the range in which we would like to work. Also, it becomes possible to integrate both RF and photonics within the same chip, which is of particular interest for this project. They work with a private-owned PDK library, so it requires signing a design kit license agreement. In any case, help is offered for the design. The prices range from 7200€ to 46000€, with the size of the chips ranging from 5 x 4.75 mm and 20 x 19.5 mm [47].

8 Future Work

In the pursuit of further enhancing the performance and capabilities of the WSS chip, there are two distinct paths to explore that deserve our attention: hardware enhancement and software code enhancement. Each of these strategies offers unique opportunities to improve the efficiency of the process, and the footprint and performance of the device.

8.1 Potential strategies based on code enhancement

A significant impediment while designing the chip has been the implementation of the CDCs with the developed code. The implementation of this BB in OptoDesigner by using parameters such as the ones obtained for the realization of the chip and presented in Table 2 and Table 3 consume a considerable amount of time, up to 3 minutes per CDC in this case. Since the chip presents 96 of these components, it may take up to 5 hours to obtain the full mask.

In order to avoid this for future developments, the code of the CDC library may be changed, reducing the computing power needed to develop these components. One path may consist of reducing the functionality of the CDCs if they are not being used. Another way would consist of upgrading sections of the code so that the same actions are performed with greater efficiency.

8.2 Potential strategies based on hardware enhancement

There are two key paths that may be followed in order to further improve the WSS chip by replacing the fundamental components by some new ones to perform the same function. One of them consists of already improving the already defined layout, by searching for new positional architectures that may reduce even further the area occupied by the chip, while still minimizing crosstalk and losses. Another procedure consists of replacing the used components for the filtering and switching sections by new ones to perform the same task, such as a micro-ring resonators (MRRs) [1,3,48,49,50]; the combination of MRRs and MZIs [1,51]; or developing approaches, such as the use of Fano-resonance based devices [50,52].

8.2.1 Fano-resonance devices

Fano resonance is a specific type of resonance that occurs because of the interference between a discrete state and a continuum band of quantum states [50,53]. Devices based on this phenomenon present a small footprint, an asymmetric resonance profile, and a very sharp response in wavelength as bandpass filters. Furthermore, it can also be tuned by changing their geometrical properties to match specific requirements [52,54]. Also, they can be developed using CMOS technologies, which means they can be used with SiPh technology [50,55].

8.2.2 Micro-ring resonators

A common approach for designing switches consists of using MRRs for filtering and switching [2,56]. This approach normally presents a more compact footprint, since the size occupied by CDCs is of about 1 mm [56]. These devices are based on the resonance of a ring by constructive interference when the phase shift of the full trip equals an integer of 2π [57]. The micro-ring resonators may present different shapes, and not necessarily the exact form of a ring. Also, they may contain more than a single resonant loop, such as a two-stage ladder.

In this specific project, two different solutions have been designed and proposed employing MRRs for future use in the development of new SiPh switches, by means of OptoDesigner. They had been designed as a 4-ring resonator, implemented through a two-stage ladder of two different loops. For the phase shift, electrical components based on the thermo-optic effect had been designed and implemented to the full MRR design, using the MCS “mcsHEATER_SOI” for the heating section and the MCS “mcsVia_metal” for the electrical connections.

The first solution consists on implementing the 4 ring resonator circuit, where the phase shift control is done with an electrical arm parallel to the ring sections whose phase shift is to be controlled, as in Figure 6(a). The implementation of this solution can be seen in Figure 37. The second solution is similar to the first one, but in this case the phase shift is controlled with an electrical via perpendicular to the ring, as in Figure 6(b). The implementation of this solution can be seen in Figure 38.

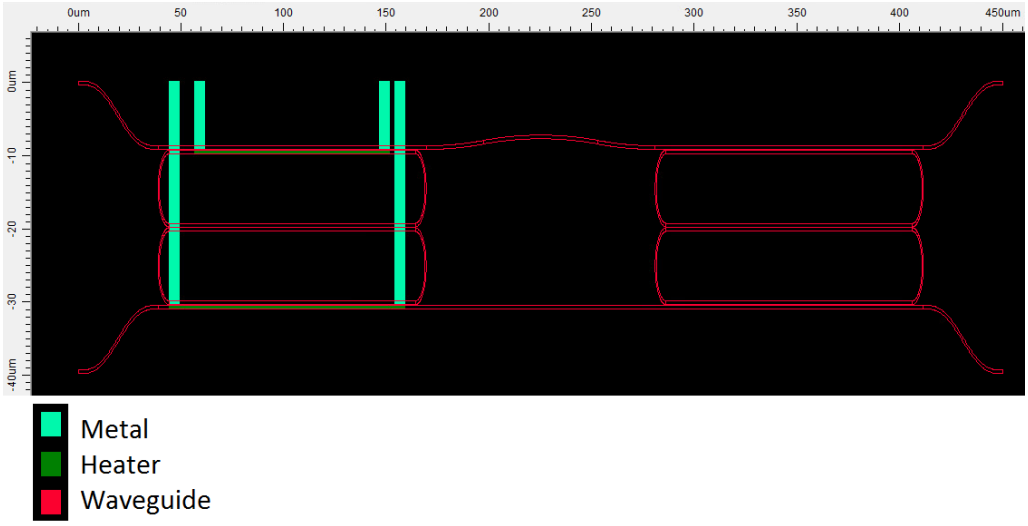


***Figure 37**. Implementation of a 4-ring resonator through a two-stage ladder with a parallel heater.*

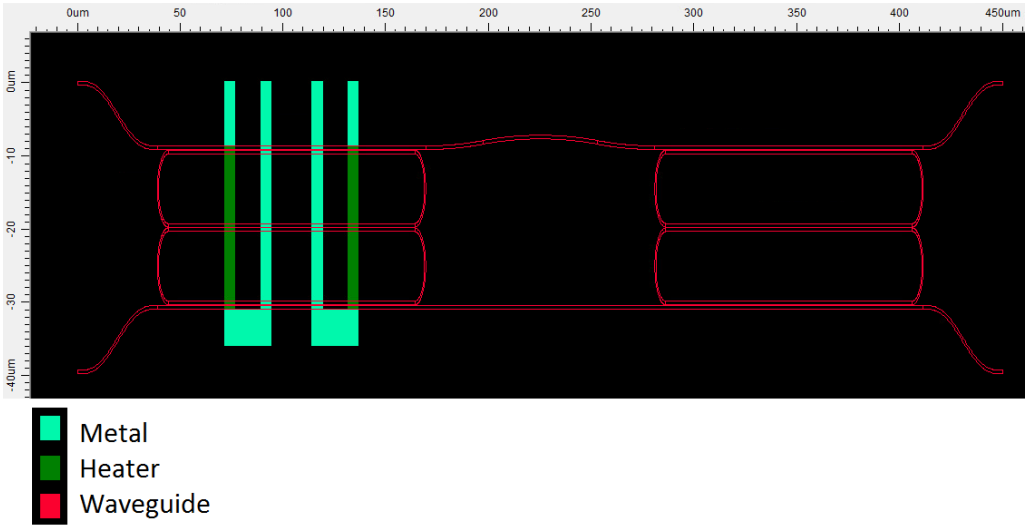


***Figure 38**. Implementation of a 4-ring resonator based on Bragg grating through a two stage ladder with a perpendicular heater.*

9 Conclusion

In this work, a state-of-the-art mask has been realized for a 1 x 3 WSS chip by using OptoDesigner Scripting language, an advanced photonic integrated circuit design tool. The main goal of this thesis was to address the increasing demand for high-performance communications systems. By a rigorous design and optimization process, a device capable of switching in the S+C+L band has been completed, which allows to distinguish in all these bands between 8 channels.

The elements forming the chip were reviewed, being their characteristics and its functionality within the layout explained, as well as the connections among them. By interconnecting the elements, two types of physical blocks were developed: the channel block, which handles the signals at the channel level; and the band block, which treats them at the band level. These two are closely related, since the latter is formed by the link of several of the former, plus specific elements to treat the signal only at the band level.

In order to complete the mask of the WSS chip, the specific elements that compose the layout were designed. These are fully parameterized functions which permit to implement each component in OptoDesigner. These components will be subsequently interconnected within the WSS, by following the design characteristics obtained by a previous numerical analysis.

The manufacturing of the device has not been completed, but several MPW providers have been reviewed. Some of them are offering possibilities for design with Synopsys PDKs, which may facilitate the process of future manufacturing.

Last, three different approaches towards improving the design have been presented. The first proposal consisted on reducing the computational complexity of the already developed CDCs. The second consisted of using Fano resonant devices, which prove to be a promising new approach towards filtering. Last, an approach on the use of MRRs for filtering is presented, for which two different implementations have been given.

References

1. X. Chen, J. Lin, and K. Wang, “A Review of Silicon-Based Integrated Optical Switches,” Laser Photonics Rev. 1, 2200571 (2023). https://doi.org/10.1002/lpor.202200571.

2. Tu, X.; Song, C.; Huang, T.; Chen, Z.; Fu, H. State of the Art and Perspectives on Silicon Photonic Switches. Micromachines 2019, 10 (1). https://doi.org/10.3390/mi10010051.

3. Yue, W.; Cai, Y.; Yu, M. Review of 2 × 2 Silicon Photonic Switches. Photonics 2023, 10 (5). https://doi.org/10.3390/photonics10050564.

4. Marchetti, R.; Lacava, C.; Carroll, L.; Gradkowski, K.; Minzioni, P. Coupling Strategies for Silicon Photonics Integrated Chips [Invited]. Photon. Res. 2019, 7, 201–239. https://doi.org/10.1364/PRJ.7.000201.

5. Blum, R. Integrated silicon photonics for high-volume data center applications. In Optical Interconnects XX; Schröder, H., Chen, R. T., Eds.; SPIE, 2020; Vol. 11286, p 112860M. https://doi.org/10.1117/12.2550326.

6. Lee, B. G.; Dupuis, N. Silicon Photonic Switch Fabrics: Technology and Architecture. Journal of Lightwave Technology 2019, 37 (1), 6–20. https://doi.org/10.1109/JLT.2018.2876828.

7. Margalit, N.; Xiang, C.; Bowers, S. M.; Bjorlin, A.; Blum, R.; Bowers, J. E. Perspective on the future of silicon photonics and electronics. Applied Physics Letters 06 2021, 118 (22), 220501. https://doi.org/10.1063/5.0050117.

8. Kaur, P.; Boes, A.; Ren, G.; Nguyen, T. G.; Roelkens, G.; Mitchell, A. Hybrid and heterogeneous photonic integration. APL Photonics 06 2021, 6 (6), 061102. https://doi.org/10.1063/5.0052700.

9. Errando-Herranz, C.; Takabayashi, A. Y.; Edinger, P.; Sattari, H.; Gylfason, K. B.; Quack, N. MEMS for Photonic Integrated Circuits. IEEE Journal of Selected Topics in Quantum Electronics 2020, 26 (2), 1–16. https://doi.org/10.1109/JSTQE.2019.2943384.

10. Rahim, A.; Spuesens, T.; Baets, R.; Bogaerts, W. Open-Access Silicon Photonics: Current Status and Emerging Initiatives. Proceedings of the IEEE 2018, 106 (12), 2313–2330. https://doi.org/10.1109/JPROC.2018.2878686.

11. Siew, S. Y.; Li, B.; Gao, F.; Zheng, H. Y.; Zhang, W.; Guo, P.; Xie, S. W.; Song, A.; Dong, B.; Luo, L. W.; Li, C.; Luo, X.; Lo, G.-Q. Review of Silicon Photonics Technology and Platform Development. J. Lightwave Technol. 2021, 39 (13), 4374–4389. https://doi.org/10.1109/jlt.2021.3066203.

12. Chrostowski, L.; Shoman, H.; Hammood, M.; Yun, H.; Jhoja, J.; Luan, E.; Lin, S.; Mistry, A.; Witt, D.; Jaeger, N. A. F.; Shekhar, S.; Jayatilleka, H.; Jean, P.; Villers, S.

B.; Cauchon, J.; Shi, W.; Horvath, C.; Westwood-Bachman, J. N.; Setzer, K.; Aktary, M.; Patrick, N. S.; Bojko, R. J.; Khavasi, A.; Wang, X.; Ferreira de Lima, T.; Tait, A. N.; Prucnal, P. R.; Hagan, D. E.; Stevanovic, D.; Knights, A. P. Silicon Photonic Circuit Design Using Rapid Prototyping Foundry Process Design Kits. IEEE Journal of Selected Topics in Quantum Electronics 2019, 25 (5), 1–26. https://doi.org/10.1109/JSTQE.2019.2917501.

13. Xiang, C.; Jin, W.; Huang, D.; Tran, M. A.; Guo, J.; Wan, Y.; Xie, W.; Kurczveil, G.; Netherton, A. M.; Liang, D.; Rong, H.; Bowers, J. E. High-Performance Silicon Photonics Using Heterogeneous Integration. IEEE Journal of Selected Topics in Quantum Electronics 2022, 28 (3: Hybrid Integration for Silicon Photonics), 1–15. https://doi.org/10.1109/JSTQE.2021.3126124.

14. Smit, M.; Williams, K.; van der Tol, J. Past, present, and future of InP-based photonic integration. APL Photonics 05 2019, 4 (5), 050901. https://doi.org/10.1063/1.5087862.

15. Porcel, M. A. G.; Hinojosa, A.; Jans, H.; Stassen, A.; Goyvaerts, J.; Geuzebroek, D.; Geiselmann, M.; Dominguez, C.; Artundo, I. [INVITED] Silicon Nitride Photonic Integration for Visible Light Applications. Optics & Laser Technology 2019, 112, 299–306. https://doi.org/10.1016/j.optlastec.2018.10.059.

16. Bowers, J. E.; Liu, A. Y. A Comparison of Four Approaches to Photonic Integration. In 2017 Optical Fiber Communications Conference and Exhibition (OFC); 2017; pp 1–3. https://doi.org/10.1364/ofc.2017.m2b.4.

17. Rahim, A.; Goyvaerts, J.; Szelag, B.; Fedeli, J.-M.; Absil, P.; Aalto, T.; Harjanne, M.; Littlejohns, C.; Reed, G.; Winzer, G.; Lischke, S.; Zimmermann, L.; Knoll, D.; Geuzebroek, D.; Leinse, A.; Geiselmann, M.; Zervas, M.; Jans, H.; Stassen, A.; Domínguez, C.; Muñoz, P.; Domenech, D.; Giesecke, A. L.; Lemme, M. C.; Baets, R. Open-Access Silicon Photonics Platforms in Europe. IEEE Journal of Selected Topics in Quantum Electronics 2019, 25 (5), 1–18. https://doi.org/10.1109/JSTQE.2019.2915949.

18. Shang, C.; Wan, Y.; Selvidge, J.; Hughes, E.; Herrick, R.; Mukherjee, K.; Duan, J.; Grillot, F.; Chow, W. W.; Bowers, J. E. Perspectives on Advances in Quantum Dot Lasers and Integration with Si Photonic Integrated Circuits. ACS Photonics 2021, 8 (9), 2555–2566. https://doi.org/10.1021/acsphotonics.1c00707.

19. Liu, A. Y.; Srinivasan, S.; Norman, J.; Gossard, A. C.; Bowers, J. E. Quantum Dot Lasers for Silicon Photonics \[Invited\]. Photon. Res. 2015, 3 (5), B1–B9. https://doi.org/10.1364/PRJ.3.0000B1.

20. LioniX international. Multi Project Wafer (MPW) Services and Runs. https://www.lionix-international.com/photonics/mpw-services/ (accessed 2023-06-09).

21. Bogaerts, W.; Chrostowski, L. Silicon Photonics Circuit Design: Methods, Tools and Challenges. Laser & Photonics Reviews 2018, 12 (4), 1700237. https://doi.org/10.1002/lpor.201700237.

22. Khan, M. U.; Xing, Y.; Ye, Y.; Bogaerts, W. Photonic Integrated Circuit Design in a Foundry+Fabless Ecosystem. IEEE Journal of Selected Topics in Quantum Electronics 2019, 25 (5), 1–14. https://doi.org/10.1109/JSTQE.2019.2918949.

23. Liu, S.; Feng, J.; Tian, Y.; Zhao, H.; Jin, L.; Ouyang, B.; Zhu, J.; Guo, J. Thermo-Optic Phase Shifters Based on Silicon-on-Insulator Platform: State-of-the-Art and a Review. Frontiers of Optoelectronics 2022, 15 (1), 9. https://doi.org/10.1007/s12200-022-00012-9.

24. Soref, R.; Bennett, B. Electrooptical Effects in Silicon. IEEE Journal of Quantum Electronics 1987, 23 (1), 123–129. https://doi.org/10.1109/JQE.1987.1073206.

25. Paschotta, R. https://www.rp-photonics.com/interferometers.html (accessed 2023-06-15).

26. Synopsys. OptoDesigner Photonic Chip and Mask Layout. https://www.synopsys.com/photonic-solutions/optocompiler/optodesigner.html (accessed 2023-06-08).

27. Doerr, C.; Chen, L.; Nielsen, T.; Aroca, R.; Chen, L.; Banaee, M.; Azemati, S.; McBrien, G.; Park, S. Y.; Geyer, J.; Guan, B.; Mikkelsen, B.; Rasmussen, C.; Givehchi, M.; Wang, Z.; Potsaid, B.; Lee, H. C.; Swanson, E.; Fujimoto, J. G. O, E, S, C, and L Band Silicon Photonics Coherent Modulator/Receiver. In 2016 Optical Fiber Communications Conference and Exhibition (OFC); 2016; pp 1–3. https://doi.org/10.1364/ofc.2016.th5c.4.

28. Montalbo, T. M. Fiber to Waveguide Couplers for Silicon Photonics. Master Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2004. https://dspace.mit.edu/bitstream/handle/1721.1/28881/60426202-MIT.pdf?sequence=2.

29. Emara, M. A Review of Optical Coupler Theory, Techniques, and Applications. Preprint, 2021. https://www.researchgate.net/publication/351428198_A_Review_of_Optical_Coupler_Theory_Techniques_and_Applications.

30. St-Yves, J. Contra-directional couplers as optical filters on the silicon on insulator platform. Master Thesis, Université Laval, Quebec City, Canada, 2017. http://hdl.handle.net/20.500.11794/27630.

31. Shi, W.; Wang, X.; Lin, C.; Yun, H.; Liu, Y.; Baehr-Jones, T.; Hochberg, M.; Jaeger, N. A. F.; Chrostowski, L. Silicon Photonic Grating-Assisted, Contra-Directional Couplers. Opt. Express 2013, 21 (3), 3633–3650. https://doi.org/10.1364/OE.21.003633.

32. Qiu, H.; Jiang, J.; Yu, P.; Mu, D.; Yang, J.; Jiang, X.; Yu, H.; Cheng, R.; Chrostowski, L. Narrow-Band Add-Drop Filter Based on Phase-Modulated Grating-Assisted Contra-Directional Couplers. J. Lightwave Technol. 2018, 36 (17), 3760–3764. https://doi.org/10.1109/jlt.2018.2852483.

33. Zetie, K. P.; Adams, S. F.; Tocknell, R. M. How Does a Mach-Zehnder Interferometer Work? Physics Education 2000, 35 (1), 46. https://doi.org/10.1088/0031-9120/35/1/308.

34. Von der Linde, D. Optical Beam Splitter, Mach–Zehnder Interferometer and the Delayed Choice Issue. Applied Physics B 2021, 127 (9), 133. https://doi.org/10.1007/s00340-021-07680-z.

35. Masood, A.; Pantouvaki, M.; Lepage, G.; Verheyen, P.; Van Campenhout, J.; Absil, P.; Van Thourhout, D.; Bogaerts, W. Comparison of Heater Architectures for Thermal Control of Silicon Photonic Circuits. In 10th International Conference on Group IV Photonics; 2013; pp 83–84. https://doi.org/10.1109/Group4.2013.6644437.

36. Europe Secures EUROPRACTICE Services to European Academia and Industry until September 2025. https://europractice-ic.com/reticles-kdtju/(accessed 2023-06-07).

37. Silicon Photonic IC Prototyping: Imec. https://www.imeciclink.com/en/asic-fabrication/si (accessed 2023-06-07).

38. Europractice. Schedules & Prices. https://europractice-ic.com/schedules-prices-2023/ (accessed 2023-06-07).

39. IHP. Schedule & Price List. https://www.ihp-microelectronics.com/services/research-and-prototyping-service/mpw-prototyping-service/schedule-price-list (accessed 2023-06-09).

40. IMB CNM. Silicon Nitride Photonic Integration Platform. https://www.imb-cnm.csic.es/en/micro-and-nanofabrication-clean-room/silicon-nitride-photonic-integration-platform (accessed 2023-06-09).

41. MPW SMART Photonics. https://smartphotonics.nl/our-offering/mpw/ (accessed 2023-06-09).

42. CORNERSTONE. MPW Schedule for 2022/23. https://www.cornerstone.sotonfab.co.uk/schedule-cost (accessed 2023-06-10).

43. AIM Photonics. Multi-Project Wafers. https://www.aimphotonics.com/mpw (accessed 2023-06-10).

44. CEA-LETI. Silicon Photonics. https://www.leti-cea.com/cea-tech/leti/english/Pages/Applied-Research/Technology-Fields/Silicon-Photonics.aspx (accessed 2023-06-10).

45. LIGENTEC. MPW - LIGENTEC. https://www.ligentec.com/offering/mpw/#1634563665378-4abd52e7-1907 (accessed 2023-06-10).

46. Applied Nanotools Inc. Nanosoi Fabrication Service: Applied Nanotools Inc. https://www.appliednt.com/nanosoi-fabrication-service/ (accessed 2023-06-12).

47. VTT. VTT Silicon Photonics. https://www.vttresearch.com/en/ourservices/silicon-photonics (accessed 2023-06-12).

48. Zhang, Q.; Han, X.; Fang, X.; Liu, M.; Ge, K.; Jiang, H.; Dong, W.; Zhang, X. A single passband microwave photonic filter with enhanced flat top and shape factor based on tunable optical bandpass filter and fiber Bragg gratings. Optics & Laser Technology 2024, 168, 109838. https://doi.org/10.1016/j.optlastec.2023.109838.

49. Liu, Y.; Chen, Y.; Wang, L.; Yu, Y.; Yu, Y.; Zhang, X. Tunable and Reconfigurable Microwave Photonic Bandpass Filter Based on Cascaded Silicon Microring Resonators. J. Lightwave Technol. 2022, 40 (14), 4655–4662. https://doi.org/10.1109/jlt.2022.3169723.

50. Arianfard, H.; Wu, J.; Juodkazis, S.; Moss, D. J. Advanced Multi-Functional Integrated Photonic Filters Based on Coupled Sagnac Loop Reflectors. Journal of Lightwave Technology 2021, 39 (5), 1400–1408. https://doi.org/10.1109/JLT.2020.3037559.

51. Li, X.; Shen, C.; Yu, X.; Zhang, Y.; Chen, C.; Zhang, X. Bandwidth-tunable optical filter based on microring resonator and MZI with Fano resonance. Journal of Optics 2020, 49 (4), 427–432. https://doi.org/10.1007/s12596-020-00642-2.

52. Najjari, V.; Mirzanejhad, S.; Ghadi, A. Plasmonic Refractive Index Sensor and Plasmonic Bandpass Filter Including Graded 4-Step Waveguide Based on Fano Resonances. Plasmonics 2022, 17 (4), 1809–1817. https://doi.org/10.1007/s11468-022-01667-y.

53. Limonov, M. F.; Rybin, M. V.; Poddubny, A. N.; Kivshar, Y. S. Fano resonances in photonics. Nature Photonics 2017, 11 (9), 543–554. https://doi.org/10.1038/nphoton.2017.142.

54. Chen, J.; Li, J.; Liu, X.; Rohimah, S.; Tian, H.; Qi, D. Fano resonance in a MIM waveguide with double symmetric rectangular stubs and its sensing characteristics. Optics Communications 2021, 482, 126563. https://doi.org/10.1016/j.optcom.2020.126563.

55. Sherif, S. M.; Swillam, M. A. Silicon-based mid infrared on-chip gas sensor using Fano resonance of coupled plasmonic microcavities. Scientific Reports 2023, 13 (1), 12311. https://doi.org/10.1038/s41598-023-38926-9.

56. Tunesi, L.; Khan, I.; Masood, M. U.; Ghillino, E.; Carena, A.; Curri, V.; Bardella, P. Modular Photonic-Integrated Device for Multi-Band Wavelength-Selective

Switching. En 2022 27th OptoElectronics and Communications Conference (OECC) and 2022 International Conference on Photonics in Switching and Computing (PSC); 2022; pp 1–3. https://doi.org/10.23919/OECC/PSC53152.2022.9850062.

57. Bogaerts, W.; De Heyn, P.; Van Vaerenbergh, T.; De Vos, K.; Kumar Selvaraja, S.; Claes, T.; Dumon, P.; Bienstman, P.; Van Thourhout, D.; Baets, R. Silicon microring resonators. Laser & Photonics Reviews 2012, 6 (1), 47–73. https://doi.org/10.1002/lpor.201100017.

58. McCaughan, A. N.; Tait, A. N.; Buckley, S. M.; Oh, D. M.; Chiles, J. T.; Shainline, J. M.; Nam, S. W. PHIDL: Python-Based Layout and Geometry Creation for Nanolithography. Journal of Vacuum Science & Technology B 2021, 39 (6), 062601. https://doi.org/10.1116/6.0001203.

59. Synopsys, Inc. Synopsys OptoDesigner Manual; 2022.

Appendix A. Introduction to PHIDL library in Python

The PHIDL library of Python has been used to represent and design an initial approach for the positioning of the elements in the layout. This library comprises an easy start towards designing circuits, because of its user-friendliness and flexibility. Also, it clarity, and consistency, along with the easiness of the Python language constitute a great factor for fast implementation [58]. This library is open-source and allows for the creation of GDS files, which are a standard in the chip designing field. Because of this, information and code examples are easily accessible through the Internet, which facilitates the process of designing. Furthermore, the lack of Design Rule Checking (DRC) makes it easier and faster to construct and attach blocks. Although this could become a problem while designing a layout to be developed into a real chip, in this case it constitutes an advantage since the design is not final.

A.1 Programming features and elements of PHIDL

Since PHIDL is a library implemented in Python, it is implemented through the object-oriented paradigm. This implies that all the different elements will be constituted as objects to which methods can be applied. There are different objects, which may be very different among them and serve different purposes. Among them, the following will be highlighted.

A.1.1 Device() object

This is the most important object in the PHIDL library, since all the designs are made through the use of these. They are equivalent to a space in which different elements can be inserted and modified. Also, these objects can be merged together in order to form a new Device object.

A.1.2 Shapes

In this library, there is a great number of different built-in shapes, such as rectangles, ellipses or even text. Furthermore, these can be packed together into a single element, with the possibility of changing their relative positions and sizes. Boolean functions can also be applied to these, in order to create new ones, based on previous elements

A.1.3 Grouping objects and references

There are two different approaches for combining together, modifying and using different elements. The first one consists of employing the class Group, which merges together the different elements, allowing for modifications as a whole or element by element, such as mirroring, rotating or moving.
The second approach consists of referencing the elements or groups of elements, which points to the location of the geometry rather than containing it. Again, by doing so, we are able to modify the element as a whole, but not element by element in this case.

A.1.4 Layers

Layers are introduced from GDS files. These are used to distinguish the type of material or process that define the elements while fabricating them. It may also be used to tell apart elements of the same material and process that are occupying the same area, but in different vertical positions. Therefore, it is an identity for manufacturing. In the case of PHIDL and GDS, they are defined by a number between 0 and 255, which permits to distinguish among them.

A.1.5 Ports

The components used in PHIDL may present ports, which are very useful in order to establish connections between the different shapes. Each of them can have a name assigned, and they occupy a specific position within the elements, with a specific angle.

A.1.6 Paths and routing

PHIDL presents functionalities to create paths and establish routes between the different ports. These functionalities range from simple algorithms, such as the creation of as a straight line, to more complex ones. For example, the algorithm may take into account the other elements in between and avoid them with a specific separation, creating an automatic route.

Appendix B. PHIDL Python Code for the layout

The code for developing the layout has been made using the PHIDL library for Python, which is accessible online at https://pypi.org/project/phidl/. This library also makes use of the gdspy library, which is accessible online at https://pypi.org/project/gdspy/.

The code is the following:

```
# Used to install the libraries if not within the computer


# gdspy is also needed for gds import
# The code of the gdspy library may be found at https://pypi.org/project/gdspy/
# The code of the PHIDL library may be found at https://pypi.org/project/phidl/
!pip install -q gdspy
!pip install -q phidl
```

```
"""
Blocks of the elements of the WSS

Includes I/O ports, CDC_channel, CDC_band, MZI, vias and its crossings and
curves.
It also includes a special structure called boot, which presents the shape
mentioned. It is used often because its simplicity and usefulness in a circuit
with our characteristics.

This one does not show the forbidden spaces
"""

import gdspy
import phidl.geometry as pg
from phidl import Device
from phidl import quickplot as qp
from phidl import set_quickplot_options as qp_set

width = 0.5      # Width of the vias (fibers)
radius = 5       # Radius of the arcs
W = 6            # Width of the MZIs and CDCs
bb = 20          # Length of the bounding box's crossing square
d = 7            # Length of the bounding box's curve square

D = Device()     # To start again all the structures while using this code



"""
Function to create a CDC for the Bands.
This function uses gdspy module and plots a CDC band with vertical or horizontal
alignment. We can set the minimum x and y positions

Inputs: pos_x = 0 -> Initial position on the x-axis
        pos_y = 0 -> Initial position on the y-axis
        alignment = 'Vertical' -> Alignment of the CDC
              ['Vertical' = 'V']  ;  ['Horizontal' = 'H']
              It does not distinguish between uppercase and lowercase letters
```

```
"""
def CDC_band(pos_x = 0, pos_y = 0, alignment = 'Vertical'):

  # Change to lowercase
  alignment = alignment.lower()

  # CDC with horizontal alignment
  if (alignment=='horizontal' or alignment=='h') :
    w = band_l  # CDC width, um
    l = W       # CDC length, um

  # CDC with vertical alignment
  elif (alignment=='vertical' or alignment=='v') :
    w = W       # CDC width, um
    l = band_l  # CDC length, um

  # Throw error if the alignment is not properly defined
  else:
    raise ValueError("The introduced alignment is not valid. Please set alignment
as 'V' for vertical or 'H' for horizontal alignment")


  p0 = (pos_x,pos_y)
  p1 = (pos_x + w, pos_y)
  p2 = (pos_x + w, pos_y + l)
  p3 = (pos_x, pos_y + l)
  D.add_polygon([p0,p1,p2,p3], layer = 0)

  #qp(D)



"""
Function to create a CDC for the channels.
This function uses gdspy module and plots a CDC channel with vertical or
horizontal alignment. We can set the minimum x and y positions

Inputs: pos_x = 0 -> Initial position on the x-axis
        pos_y = 0 -> Initial position on the y-axis
        alignment = 'Vertical' -> Alignment of the CDC
              ['Vertical' = 'V']  ;  ['Horizontal' = 'H']
              It does not distinguish between uppercase and lowercase letters

"""
def CDC_channel(pos_x = 0, pos_y = 0, alignment = 'Vertical'):

  # Change to lowercase
  alignment = alignment.lower()

  # CDC with horizontal alignment
  if (alignment=='horizontal' or alignment=='h') :
    w = chann_l # CDC width, um
    l = W       # CDC length, um

  # CDC with vertical alignment
  elif (alignment=='vertical' or alignment=='v') :
    w = W       # CDC width, um
    l = chann_l # CDC length, um
```

```
  # Throw error if the alignment is not properly defined
  else:
    raise ValueError("The introduced alignment is not valid. Please set alignment
as 'V' for vertical or 'H' for horizontal alignment")


  p0 = (pos_x,pos_y)
  p1 = (pos_x + w, pos_y)
  p2 = (pos_x + w, pos_y + l)
  p3 = (pos_x, pos_y + l)
  D.add_polygon([p0,p1,p2,p3], layer = 1)

  #qp(D)




"""
Function to create an MZI.
This function uses gdspy module and plots an MZI with vertical or horizontal
alignment. We can set the minimum x and y positions

Inputs: pos_x = 0 -> Initial position on the x-axis
        pos_y = 0 -> Initial position on the y-axis
        alignment = 'Vertical' -> Alignment of the CDC.
              ['Vertical' = 'V']  ;  ['Horizontal' = 'H']
              It does not distinguish between uppercase and lowercase letters

"""
def MZI(pos_x = 0, pos_y = 0, alignment = 'Vertical'):

  # Change to lowercase
  alignment = alignment.lower()

  # CDC with horizontal alignment
  if (alignment=='horizontal' or alignment=='h') :
    w = MZI_l   # CDC width, um
    l = W       # CDC length, um

  # CDC with vertical alignment
  elif (alignment=='vertical' or alignment=='v') :
    w = W       # CDC width, um
    l = MZI_l   # CDC length, um

  # Throw error if the alignment is not properly defined
  else:
    raise ValueError("The introduced alignment is not valid. Please set alignment
as 'V' for vertical or 'H' for horizontal alignment")


  p0 = (pos_x,pos_y)
  p1 = (pos_x + w, pos_y)
  p2 = (pos_x + w, pos_y + l)
  p3 = (pos_x, pos_y + l)
  D.add_polygon([p0,p1,p2,p3], layer = 2)

  #qp(D)
```

```
"""
Function that creates a curve in a fiber, that also contains a bounding box of
7x7 um
Inputs:   pos_x = 0 -> Initial position on x-axis
          pos_y = 0 -> Initial Position on y-axis
          initial_direction = 'Top' ; ['Top','Bottom','Right','Left'] -> Initial
direction of the fiber
          final_direction = 'Right' ; ['Top','Bottom','Right','Left'] -> Final
direction of the fiber.
                                                                        It must
be 90º apart from initial_direction
          These 2 string inputs do not distinguish between uppercase and
lowercase letters
"""
def curve90 (pos_x = 0, pos_y = 0, initial_direction = 'Top', final_direction =
'Right'):

  # Changes the value of the string to lowercase
  initial_direction = initial_direction.lower()
  final_direction = final_direction.lower()

  # From top to right
  if initial_direction == 'top' and final_direction == 'right':
    arc = pg.arc(radius = radius,
                 width = width,
                 theta = -90,
                 start_angle = 180,
                 angle_resolution = 0.5,
                 layer = 4)
    arc.movex(pos_x+radius)
    arc.movey(pos_y)
    D.add_ref(arc)

  # From top to left
  elif (initial_direction == 'top' and final_direction == 'left'):
    arc = pg.arc(radius = radius,
                 width = width,
                 theta = 90,
                 start_angle = 0,
                 angle_resolution = 0.5,
                 layer = 4)
    arc.movex(pos_x-radius)
    arc.movey(pos_y)
    D.add_ref(arc)

  # From bottom to right
  elif (initial_direction == 'bottom' and final_direction == 'right'):
    arc = pg.arc(radius = radius,
                 width = width,
                 theta = 90,
                 start_angle = 180,
                 angle_resolution = 0.5,
                 layer = 4)
    arc.movex(pos_x+radius)
    arc.movey(pos_y)
    D.add_ref(arc)
```

```
# From bottom to left
elif (initial_direction == 'bottom' and final_direction == 'left'):
  arc = pg.arc(radius = radius,
               width = width,
               theta = -90,
               start_angle = 0,
               angle_resolution = 0.5,
               layer = 4)
  arc.movex(pos_x-radius)
  arc.movey(pos_y)
  D.add_ref(arc)

# From right to top
elif (initial_direction == 'right' and final_direction == 'top'):
  arc = pg.arc(radius = radius,
               width = width,
               theta = 90,
               start_angle = -90,
               angle_resolution = 0.5,
               layer = 4)
  arc.movex(pos_x)
  arc.movey(pos_y+radius)
  D.add_ref(arc)

# From right to bottom
elif (initial_direction == 'right' and final_direction == 'bottom'):
  arc = pg.arc(radius = radius,
               width = width,
               theta = -90,
               start_angle = 90,
               angle_resolution = 0.5,
               layer = 4)
  arc.movex(pos_x)
  arc.movey(pos_y-radius)
  D.add_ref(arc)

# From left to top
elif (initial_direction == 'left' and final_direction == 'top'):
  arc = pg.arc(radius = radius,
               width = width,
               theta = -90,
               start_angle = -90,
               angle_resolution = 0.5,
               layer = 4)
  arc.movex(pos_x)
  arc.movey(pos_y+radius)
  D.add_ref(arc)

# From left to bottom
elif (initial_direction == 'left' and final_direction == 'bottom'):
  arc = pg.arc(radius = radius,
               width = width,
               theta = 90,
               start_angle = 90,
               angle_resolution = 0.5,
               layer = 4)
  arc.movex(pos_x)
  arc.movey(pos_y-radius)
```

```
    D.add_ref(arc)

  # If the initial and/or final direction are not the expected
  else:
    raise ValueError('The introduced initial and/or final directions are not
correct. Please type \'Top\', \'Bottom\', \'Right\' or \'Left\' for
initial_direction.\n \
          For final_direction, please chose a direction 90º apart from the
chosen initial one')

  #qp(D)



"""
Function that generates a crossing at the specified position with a bounding box
Inputs: pos_x = 0 -> Position on the x-axis in which the crossing takes place
        pos_y = 0-> Position on the y-axis in which the crossing takes place
        bounding_box = bb -> Length of the bounding box's square
"""
def crossing(pos_x = 0, pos_y = 0, bounding_box = bb):

  # Horizontal fiber
  p0 = (pos_x-bounding_box/2, pos_y-width/2)
  p1 = (pos_x-bounding_box/2, pos_y+width/2)
  p2 = (pos_x+bounding_box/2, pos_y+width/2)
  p3 = (pos_x+bounding_box/2, pos_y-width/2)
  D.add_polygon([p0,p1,p2,p3], layer = 4)

  # Vertical fiber
  p0 = (pos_x-width/2, pos_y-bounding_box/2)
  p1 = (pos_x-width/2, pos_y+bounding_box/2)
  p2 = (pos_x+width/2, pos_y+bounding_box/2)
  p3 = (pos_x+width/2, pos_y-bounding_box/2)
  D.add_polygon([p0,p1,p2,p3], layer = 4)

  # Defining the ports
  #D.add_port(name = 'vd', midpoint = [pos_x,pos_y-bounding_box/2], width =
width, orientation = -90)  # Vertical down
  #D.add_port(name = 'vt', midpoint = [pos_x,pos_y+bounding_box/2], width =
width, orientation = 90)   # Vertical top
  #D.add_port(name = 'hl', midpoint = [pos_x-bounding_box/2,pos_y], width =
width, orientation = 180)  # Horizontal left
  #D.add_port(name = 'hr', midpoint = [pos_x+bounding_box/2,pos_y], width =
width, orientation = 0)    # Horizontal right

  #qp(D)



'''
Function that generates a via from a starting position with a certain length
Inputs: init_x = 0 -> Starting position on the x-axis
        init_y = 0 -> Starting position on the y-axis
        length = 10 -> Length of the fiber
        orientation = 'Top' ; ['Top', 'Bottom', 'Right', 'Left'] -> Via
orientation;
```

```
        It does not distinguish between uppercase and lowercase letters
'''
def via(init_x = 0, init_y = 0, length = 10, orientation = 'Top'):

  # Change to lowercase
  orientation = orientation.lower()

  # Via going to the top
  if orientation == 'top':
    #if length > 0:
    p0 = (init_x-width/2, init_y)
    p1 = (init_x+width/2, init_y)
    p2 = (init_x+width/2, init_y+length)
    p3 = (init_x-width/2, init_y+length)
    D.add_polygon([p0,p1,p2,p3], layer = 4)
      #D.add_port(name = '1', midpoint = [init_x,init_y], width = width,
orientation = -90)  # Vertical down
      #D.add_port(name = '2', midpoint = [init_x,init_y+length], width = width,
orientation = 90)   # Vertical top
    #else:
      #raise ValueError(f"Input length of the via must be higher than 0.\nIts
current value is: {length}")

  # Via going to the top
  elif orientation == 'bottom':
    #if length > 0:
    p0 = (init_x-width/2, init_y)
    p1 = (init_x+width/2, init_y)
    p2 = (init_x+width/2, init_y-length)
    p3 = (init_x-width/2, init_y-length)
    D.add_polygon([p0,p1,p2,p3], layer = 4)
      #D.add_port(name = '2', midpoint = [init_x,init_y-length], width = width,
orientation = -90)  # Vertical down
      #D.add_port(name = '1', midpoint = [init_x,init_y], width = width,
orientation = 90)          # Vertical top
    #else:
      #raise ValueError(f"Input length of the via must be higher than 0.\nIts
current value is: {length}")

  # Via going to the right
  elif orientation == 'right':
    #if length > 0:
    p0 = (init_x, init_y-width/2)
    p1 = (init_x, init_y+width/2)
    p2 = (init_x+length, init_y+width/2)
    p3 = (init_x+length, init_y-width/2)
    D.add_polygon([p0,p1,p2,p3], layer = 4)
      #D.add_port(name = '1', midpoint = [init_x, init_y], width = width,
orientation = 180)         # Horizontal left
      #D.add_port(name = '2', midpoint = [init_x+length, init_y], width = width,
orientation = 0)    # Horizontal right
    #else:
      #raise ValueError(f"Input length of the via must be higher than 0.\nIts
current value is: {length}")

  # Via going to the left
  elif orientation == 'left':
    #if length > 0:
    p0 = (init_x, init_y-width/2)
```

```
    p1 = (init_x, init_y+width/2)
    p2 = (init_x-length, init_y+width/2)
    p3 = (init_x-length, init_y-width/2)
    D.add_polygon([p0,p1,p2,p3], layer = 4)
      #D.add_port(name = '2', midpoint = [init_x-length, init_y], width = width,
orientation = 180)         # Horizontal left
      #D.add_port(name = '1', midpoint = [init_x, init_y], width = width,
orientation = 0)    # Horizontal right
    #else:
      #raise ValueError(f"Input length of the via must be higher than 0.\nIts
current value is: {length}")

  # If the keyword specified is incorrect
  else:
    raise ValueError ("Orientation value is not correct. Please type 'Top',
'Bottom', 'Right' or 'Left'")

  #qp(D)




'''
Create a path with the same direction for the entry and exit, but with a certain
separation.
The figure has the shape of a boot.
Inputs: pos_x = 0 -> Central position of the entry (X-position)
        pos_y = 0 -> Central position of the entry (Y-position)
        initial_direction = 'Top' ; ['Top', 'Bottom', 'Right', 'Left'] ->
Direction of the entry.
        exit_via = 'Right' ; ['Top', 'Bottom', 'Right', 'Left']-> Position of the
exit from the entry.
                                                                    It must be 90º
apart from initial_direction
        safe_space = 'False' ; ['False', 'True'] -> Whether there is a safe space
to place another curve90
                                                    with the same direction as
the entry of the boot or not.
                                                    It corresponds to an extra
via on both the entry and exit
                                                    These extra vias equal to
(radius-width)/2
                                                    'False' if there are not
these extra vias
                                                    'True' if there are
        separation = radius -> Distance of separation between the exit and entry
vias. Must be higher than width,
                               This separation is maintained as the minimum of
all points to a curve90 in the
                               same direction as the first one on the entry but
with a certain separation if
                               safe_space = 'True'
        All the different string inputs do not distinguish between uppercase and
lowercase letters
'''
def boot(pos_x = 0, pos_y = 0, initial_direction = 'Top', exit_via = 'Right',
safe_space = 'False', separation = radius):
  initial_direction = initial_direction.lower()
  exit_via = exit_via.lower()
```

```
  safe_space = safe_space.lower()

  if separation >= width:
    if safe_space == 'true' or safe_space == 'false':
      if initial_direction == 'top':
        if exit_via == 'right':
          if safe_space == 'false':
            curve90(pos_x,pos_y,'Top','Left')
            curve90(pos_x-radius,pos_y+radius,'Left','Top')
            curve90(pos_x-2*radius,pos_y+2*radius,'Top','Right')
            via(pos_x-radius,pos_y+3*radius,separation+width,'Right')
            curve90(pos_x-
radius+separation+width,pos_y+3*radius,'Right','Bottom')
            via(pos_x+separation+width,pos_y+2*radius,2*radius,'Bottom')
          else:
            ss = separation-width                          # Extra vias length (To
maintain the minimum distance)
            via(pos_x,pos_y,ss,'Top')                      # Extra entry via
            via(pos_x+separation+width,pos_y,ss,'Top')     # Extra exit via
            curve90(pos_x,pos_y+ss,'Top','Left')
            curve90(pos_x-radius,pos_y+radius+ss,'Left','Top')
            curve90(pos_x-2*radius,pos_y+2*radius+ss,'Top','Right')
            via(pos_x-radius,pos_y+3*radius+ss,separation+width,'Right')
            curve90(pos_x-
radius+separation+width,pos_y+3*radius+ss,'Right','Bottom')
            via(pos_x+separation+width,pos_y+2*radius+ss,2*radius,'Bottom')

        elif exit_via == 'left':
          if safe_space == 'false':
            curve90(pos_x,pos_y,'Top','Right')
            curve90(pos_x+radius,pos_y+radius,'Right','Top')
            curve90(pos_x+2*radius,pos_y+2*radius,'Top','Left')
            via(pos_x+radius,pos_y+3*radius,separation+width,'Left')
            curve90(pos_x+radius-separation-width,pos_y+3*radius,'Left','Bottom')
            via(pos_x-separation-width,pos_y+2*radius,2*radius,'Bottom')
          else:
            ss = separation-width                            # Extra vias length
(To maintain the minimum distance)
            via(pos_x,pos_y,ss,'Top')                        # Extra entry via
            via(pos_x-separation-width,pos_y,ss,'Top')       # Extra exit via
            curve90(pos_x,pos_y+ss,'Top','Right')
            curve90(pos_x+radius,pos_y+radius+ss,'Right','Top')
            curve90(pos_x+2*radius,pos_y+2*radius+ss,'Top','Left')
            via(pos_x+radius,pos_y+3*radius+ss,separation+width,'Left')
            curve90(pos_x+radius-separation-
width,pos_y+3*radius+ss,'Left','Bottom')
            via(pos_x-separation-width,pos_y+2*radius+ss,2*radius,'Bottom')


        else:
          raise ValueError("The circuit goes initially to the top, so we can only
have exit_via = 'Right' or 'Left'")


      elif initial_direction == 'bottom':
        if exit_via == 'right':
          if safe_space == 'false':
            curve90(pos_x,pos_y,'Bottom','Left')
            curve90(pos_x-radius,pos_y-radius,'Left','Bottom')
```

```
            curve90(pos_x-2*radius,pos_y-2*radius,'Bottom','Right')
            via(pos_x-radius,pos_y-3*radius,separation+width,'Right')
            curve90(pos_x-radius+separation+width,pos_y-3*radius,'Right','Top')
            via(pos_x+separation+width,pos_y-2*radius,2*radius,'Top')
          else:
            ss = separation-width                          # Extra vias length
(To maintain the minimum distance)
            via(pos_x,pos_y,ss,'Bottom')                   # Extra entry via
            via(pos_x+separation+width,pos_y,ss,'Bottom')  # Extra exit via
            curve90(pos_x,pos_y-ss,'Bottom','Left')
            curve90(pos_x-radius,pos_y-radius-ss,'Left','Bottom')
            curve90(pos_x-2*radius,pos_y-2*radius-ss,'Bottom','Right')
            via(pos_x-radius,pos_y-3*radius-ss,separation+width,'Right')
            curve90(pos_x-radius+separation+width,pos_y-3*radius-
ss,'Right','Top')
            via(pos_x+separation+width,pos_y-2*radius-ss,2*radius,'Top')

        elif exit_via == 'left':
          if safe_space == 'false':
            curve90(pos_x,pos_y,'Bottom','Right')
            curve90(pos_x+radius,pos_y-radius,'Right','Bottom')
            curve90(pos_x+2*radius,pos_y-2*radius,'Bottom','Left')
            via(pos_x+radius,pos_y-3*radius,separation+width,'Left')
            curve90(pos_x+radius-separation-width,pos_y-3*radius,'Left','Top')
            via(pos_x-separation-width,pos_y-2*radius,2*radius,'Top')
          else:
            ss = separation-width                          # Extra vias length
(To maintain the minimum distance)
            via(pos_x,pos_y,ss,'Bottom')                   # Extra entry via
            via(pos_x-separation-width,pos_y,ss,'Bottom')  # Extra exit via
            curve90(pos_x,pos_y-ss,'Bottom','Right')
            curve90(pos_x+radius,pos_y-radius-ss,'Right','Bottom')
            curve90(pos_x+2*radius,pos_y-2*radius-ss,'Bottom','Left')
            via(pos_x+radius,pos_y-3*radius-ss,separation+width,'Left')
            curve90(pos_x+radius-separation-width,pos_y-3*radius-ss,'Left','Top')
            via(pos_x-separation-width,pos_y-2*radius-ss,2*radius,'Top')

        else:
          raise ValueError("The circuit goes initially to the bottom, so we can
only have exit_via = 'Right' or 'Left'")


      elif initial_direction == 'right':
        if exit_via == 'top':
          if safe_space == 'false':
            curve90(pos_x,pos_y,'Right','Bottom')
            curve90(pos_x+radius,pos_y-radius,'Bottom','Right')
            curve90(pos_x+2*radius,pos_y-2*radius,'Right','Top')
            via(pos_x+3*radius,pos_y-radius,separation+width,'Top')
            curve90(pos_x+3*radius,pos_y-radius+separation+width,'Top','Left')
            via(pos_x+2*radius,pos_y+separation+width,2*radius,'Left')
          else:
            ss = separation-width                          # Extra vias length
(To maintain the minimum distance)
            via(pos_x,pos_y,ss,'Right')                    # Extra entry via
            via(pos_x,pos_y+separation+width,ss,'Right')   # Extra exit via
            curve90(pos_x+ss,pos_y,'Right','Bottom')
            curve90(pos_x+radius+ss,pos_y-radius,'Bottom','Right')
            curve90(pos_x+2*radius+ss,pos_y-2*radius,'Right','Top')
```

```
            via(pos_x+3*radius+ss,pos_y-radius,separation+width,'Top')
            curve90(pos_x+3*radius+ss,pos_y-radius+separation+width,'Top','Left')
            via(pos_x+2*radius+ss,pos_y+separation+width,2*radius,'Left')

        elif exit_via == 'bottom':
          if safe_space == 'false':
            curve90(pos_x,pos_y,'Right','Top')
            curve90(pos_x+radius,pos_x+radius,'Top','Right')
            curve90(pos_x+2*radius,pos_x+2*radius,'Right','Bottom')
            via(pos_x+3*radius,pos_x+radius,separation+width,'Bottom')
            curve90(pos_x+3*radius,pos_x+radius-separation-width,'Bottom','Left')
            via(pos_x+2*radius,pos_x-separation-width,2*radius,'Left')
          else:
            ss = separation-width                          # Extra vias length
(To maintain the minimum distance)
            via(pos_x,pos_y,ss,'Right')                    # Extra entry via
            via(pos_x,pos_y-separation-width,ss,'Right')   # Extra exit via
            curve90(pos_x+ss,pos_y,'Right','Top')
            curve90(pos_x+radius+ss,pos_x+radius,'Top','Right')
            curve90(pos_x+2*radius+ss,pos_x+2*radius,'Right','Bottom')
            via(pos_x+3*radius+ss,pos_x+radius,separation+width,'Bottom')
            curve90(pos_x+3*radius+ss,pos_x+radius-separation-
width,'Bottom','Left')
            via(pos_x+2*radius+ss,pos_x-separation-width,2*radius,'Left')

        else:
          raise ValueError("The circuit goes initially to the right, so we can
only have exit_via = 'Top' or 'Bottom'")


      elif initial_direction == 'left':
        if exit_via == 'top':
          if safe_space == 'false':
            curve90(pos_x,pos_y,'Left','Bottom')
            curve90(pos_x-radius,pos_y-radius,'Bottom','Left')
            curve90(pos_x-2*radius,pos_y-2*radius,'Left','Top')
            via(pos_x-3*radius,pos_y-radius,separation+width,'Top')
            curve90(pos_x-3*radius,pos_y-radius+separation+width,'Top','Right')
            via(pos_x-2*radius,pos_y+separation+width,2*radius,'Right')
          else:
            ss = separation-width                          # Extra vias length
(To maintain the minimum distance)
            via(pos_x,pos_y,ss,'Left')                     # Extra entry via
            via(pos_x,pos_y+separation+width,ss,'Left')    # Extra exit via
            curve90(pos_x-ss,pos_y,'Left','Bottom')
            curve90(pos_x-radius-ss,pos_y-radius,'Bottom','Left')
            curve90(pos_x-2*radius-ss,pos_y-2*radius,'Left','Top')
            via(pos_x-3*radius-ss,pos_y-radius,separation+width,'Top')
            curve90(pos_x-3*radius-ss,pos_y-
radius+separation+width,'Top','Right')
            via(pos_x-2*radius-ss,pos_y+separation+width,2*radius,'Right')

        elif exit_via == 'bottom':
          if safe_space == 'false':
            curve90(pos_x,pos_y,'Left','Top')
            curve90(pos_x-radius,pos_y+radius,'Top','Left')
            curve90(pos_x-2*radius,pos_y+2*radius,'Left','Bottom')
            via(pos_x-3*radius,pos_y+radius,separation+width,'Bottom')
```

```
            curve90(pos_x-3*radius,pos_y+radius-separation-
width,'Bottom','Right')
            via(pos_x-2*radius,pos_y-separation-width,2*radius,'Right')
          else:
            ss = separation-width                          # Extra vias length
(To maintain the minimum distance)
            via(pos_x,pos_y,ss,'Left')                     # Extra entry via
            via(pos_x,pos_y-separation-width,ss,'Left')    # Extra exit via
            curve90(pos_x-ss,pos_y,'Left','Top')
            curve90(pos_x-radius-ss,pos_y+radius,'Top','Left')
            curve90(pos_x-2*radius-ss,pos_y+2*radius,'Left','Bottom')
            via(pos_x-3*radius-ss,pos_y+radius,separation+width,'Bottom')
            curve90(pos_x-3*radius-ss,pos_y+radius-separation-
width,'Bottom','Right')
            via(pos_x-2*radius-ss,pos_y-separation-width,2*radius,'Right')

        else:
          raise ValueError("The circuit goes initially to the left, so we can
only have exit_via = 'Top' or 'Bottom'")


    else:
      raise ValueError("The input safe_space can only have the values 'False' or
'True'")


  else:
    raise ValueError(f"The separation must be higher than the width of the vias
(width={width})")


  #qp(D)



'''
Function that creates an I/O port, which has the shape of a quasi-triangle, with
a minimum (width of the vias) and maximum widths defined by the user
Inputs: init_x = 0 -> X-position at which the via connects to the port
        init_y = 0 -> Y-position of the center of the quasi-triangle (and center
of the via)
        wid = 0.5 -> Width of the incoming via
        height = 50 -> Maximum height of the I/O port at its end
These I/O ports always connect to a via on its left
'''
def IO (init_x = 0, init_y = 0, wid = 0.5, height = 50, length = 40, t = "I"):
  if wid > 0 and height > 0 and length > 0:

    # Creating the quasi-triangle
    p0 = (init_x, init_y+wid/2)
    p1 = (init_x, init_y-wid/2)
    p2 = (init_x-length, init_y-height/2)
    p3 = (init_x-length, init_y+height/2)
    D.add_polygon([p0,p1,p2,p3], layer = 4)

    # Creating the text
    siz = height
    T = pg.text(text = t, size = siz,
```

```
            justify = 'left', layer = 3)
    text = D.add_ref(T)
    text.movex(init_x-length-(len(t)-t.count(' '))*siz)
    text.movey(init_y-siz/2)
  else:
    raise ValueError("The inputs width, height and length must be all higher than
0")

  #qp(D)



"""
Creating the whole Wavelength Selective Switch (WSS) with 50 um
(Without the last band CDCs (on the right))
Final Schematic
"""


MZI_l = 50                                  # Length of the MZI
chann_l = 50                                # Length of the CDC of the channels
#band_l = 50                                # Length of the CDC of the bands
band_l = MZI_l+chann_l+5/2*bb+4*radius    # Length of the CDC of the bands
(Similar proportion to reality)

IO_h = 22.152        # Height of the I/O ports
IO_l = 35.603        # Horizontal length of the I/O ports

n_channels = 8  # Number of channels per band of the WSS

# Distance between two channel structures (from the input of one to the input of
the next)
ms_x = 8*radius+3*W-3*width+bb/2

port_d = 30  # Distance between two consecutive I/O ports


D = Device()




'''
The smallest structure of our WSS, made by 1 input channel CDC, 3 output channel
CDCs and 2 MZIs.
It constitutes a single channel within the WSS.
Inputs: pos_x = 0 -> Initial position on the x-axis (Smallest of the input
channel CDC)
        pos_y = 0 -> Initial position on the y-axis (Smallest of the input
channel CDC)
        structure_type = 0 -> Type of desired channel-structure. 'Contiguous' for
a structure connected to another on its right,
                              'Initial' for one not connected to another on its
left,
                              and 'Final' for the last one which does not connect
to another channel on its right
                              It does not distinguish between uppercase and
lowercase letters
```

```
'''
def channel_struct(pos_x = 0, pos_y = 0, structure_type = 'Initial'):
  structure_type = structure_type.lower()

  if structure_type == 'initial':
    CDC_channel(pos_x,pos_y)  # Input CDC channel
    # From input CDC channel to MZI
    via(pos_x+W-width/2,pos_y,radius+width/2+bb,'Bottom')

    # MZIs and its connections
    # MZI 1 (the one on the left)
    MZI(pos_x+W-width,pos_y-(MZI_l+bb+radius+width/2))
    # I (MZI interconnection)
    curve90(pos_x+2*W-1.5*width,pos_y-(MZI_l+bb+radius+width/2),'Bottom','Right')
    via(pos_x+2*W-1.5*width+radius,pos_y-(MZI_l+2*radius+bb+width/2),3*radius,
'Right')
    curve90(pos_x+2*W-3/2*width+bb/2+2*radius,pos_y-
(MZI_l+bb+2*radius+width/2),'right','top')
    # MZI 2 (the one on the right)
    MZI(pos_x+2*W-2*width+bb/2+3*radius,pos_y-(MZI_l+bb+radius+width/2))
    # P1 (to output channel CDC 1 of the next structure)
    via(pos_x+W-width/2,pos_y-(MZI_l+bb+radius+width/2),width+bb,'Bottom')
    curve90(pos_x+W-width/2,pos_y-MZI_l-2*bb-radius-3/2*width,'Bottom', 'Right')
    via(pos_x+W+radius-width/2,pos_y-MZI_l-2*bb-2*radius-3/2*width,W-
width+radius,'Right')
    curve90(pos_x+2*W-3/2*width+2*radius,pos_y-MZI_l-2*bb-2*radius-
3/2*width,'Right','Bottom')
    via(pos_x+2*W-3/2*width+3*radius,pos_y-MZI_l-2*bb-3*radius-
3/2*width,chann_l,'Bottom')
    curve90(pos_x+2*W-3/2*width+3*radius,pos_y-MZI_l-chann_l-2*bb-3*radius-
3/2*width,'Bottom','Right')
    via(pos_x+2*W-3/2*width+4*radius,pos_y-MZI_l-chann_l-2*bb-4*radius-
3/2*width,4*W-5*width+bb/2+2*radius,'Right')
    curve90(pos_x+6*W-13/2*width+6*radius+bb/2,pos_y-MZI_l-chann_l-2*bb-4*radius-
3/2*width,'Right','Top')
    # P2 (to output channel CDC 2)
    via(pos_x+2*W-3/2*width+bb/2+3*radius,pos_y-bb-radius-width/2,bb/2-
radius,'Top')
    curve90(pos_x+2*W-3/2*width+bb/2+3*radius,pos_y-bb/2-2*radius-
width/2,'Top','Right')
    via(pos_x+2*W-3/2*width+bb/2+4*radius,pos_y-bb/2-radius-
width/2,3*radius+W,'Right')

    # P3 (to output channel CDC 3)
    curve90(pos_x+3*W-5/2*width+bb/2+3*radius,pos_y-
(bb+radius+width/2),'Top','Right')
    curve90(pos_x+3*W-5/2*width+bb/2+4*radius,pos_y-
(bb+width/2),'Right','Bottom')
    via(pos_x+3*W-5/2*width+bb/2+5*radius,pos_y-
(bb+radius+width/2),MZI_l+bb/2+width/2, 'Bottom')
    curve90(pos_x+3*W-5/2*width+bb/2+5*radius,pos_y-MZI_l-3/2*bb-radius-
width,'Bottom','Right')
    via(pos_x+3*W-5/2*width+bb/2+6*radius,pos_y-MZI_l-3/2*bb-2*radius-width,W-
width,'Right')

    # Vias between input channel CDCs
    curve90(pos_x+ms_x+width/2,pos_y,'Bottom','Left')
    curve90(pos_x+ms_x-radius+width/2,pos_y-radius,'Left','Top')
    via(pos_x+ms_x-2*radius+width/2,pos_y,chann_l+3/2*width,'Top')
```

```
    curve90(pos_x+ms_x-2*radius+width/2,pos_y+chann_l+3/2*width,'Top','Left')
    via(pos_x+ms_x-
3*radius+width/2,pos_y+chann_l+radius+3/2*width,4*radius+3*W+bb/2-3*width,'Left')
    curve90(pos_x+ms_x-7*radius-3*W+7/2*width-
bb/2,pos_y+chann_l+radius+3/2*width,'Left','Bottom')
    via(pos_x+ms_x-8*radius-3*W+7/2*width-
bb/2,pos_y+chann_l+3/2*width,3/2*width,'Bottom')


  elif structure_type == 'contiguous':
    CDC_channel(pos_x,pos_y)  # Input CDC channel
    # From input CDC channel to MZI
    via(pos_x+W-width/2,pos_y,radius+width/2,'Bottom')
    crossing(pos_x+W-width/2,pos_y-radius-width/2-bb/2)

    # MZIs and its connections
    # MZI 1 (the one on the left)
    MZI(pos_x+W-width,pos_y-(MZI_l+bb+radius+width/2))
    # I (MZI interconnection)
    curve90(pos_x+2*W-1.5*width,pos_y-(MZI_l+bb+radius+width/2),'Bottom','Right')
    via(pos_x+2*W-1.5*width+radius,pos_y-(MZI_l+2*radius+bb+width/2),3*radius,
'Right')
    curve90(pos_x+2*W-3/2*width+bb/2+2*radius,pos_y-
(MZI_l+bb+2*radius+width/2),'right','top')
    # MZI 2 (the one on the right)
    MZI(pos_x+2*W-2*width+bb/2+3*radius,pos_y-(MZI_l+bb+radius+width/2))
    # P1 (to output channel CDC 1)
    via(pos_x+W-width/2,pos_y-(MZI_l+bb+radius+width/2),radius+width/2,'Bottom')
    crossing(pos_x+W-width/2,pos_y-(MZI_l+3/2*bb+2*radius+width))
    via(pos_x+W-width/2,pos_y-
(MZI_l+2*bb+2*radius+width),radius+width/2,'Bottom')
    # P2 (to output channel CDC 2)
    via(pos_x+2*W-3/2*width+bb/2+3*radius,pos_y-bb-radius-
width/2,chann_l+bb/2+radius,'Top')
    curve90(pos_x+2*W-3/2*width+bb/2+3*radius,pos_y+chann_l-bb/2-
width/2,'Top','Left')
    curve90(pos_x+2*W-3/2*width+bb/2+2*radius,pos_y+chann_l-bb/2-
width/2+radius,'Left','Bottom')
    # P3 (to output channel CDC 3)
    curve90(pos_x+3*W-5/2*width+bb/2+3*radius,pos_y-
(bb+radius+width/2),'Top','Right')
    curve90(pos_x+3*W-5/2*width+bb/2+4*radius,pos_y-
(bb+width/2),'Right','Bottom')
    via(pos_x+3*W-5/2*width+bb/2+5*radius,pos_y-
(bb+radius+width/2),MZI_l+bb/2+2*radius+width/2, 'Bottom')

    # Output channel CDCs
    CDC_channel(pos_x+W-width,pos_y-MZI_l-chann_l-2*bb-3*radius-3/2*width)
# Output Channel CDC 1
    CDC_channel(pos_x+W-width+bb/2+radius,pos_y-width/2-bb/2)
# Output Channel CDC 2
    CDC_channel(pos_x+3*W-3*width+bb/2+5*radius,pos_y-MZI_l-chann_l-3/2*bb-
3*radius-width)      # Output Channel CDC 3

    # Vias between input channel CDCs
    curve90(pos_x+ms_x+width/2,pos_y,'Bottom','Left')
    curve90(pos_x+ms_x-radius+width/2,pos_y-radius,'Left','Top')
    via(pos_x+ms_x-2*radius+width/2,pos_y,chann_l+3/2*width,'Top')
    curve90(pos_x+ms_x-2*radius+width/2,pos_y+chann_l+3/2*width,'Top','Left')
```

```
    via(pos_x+ms_x-
3*radius+width/2,pos_y+chann_l+radius+3/2*width,4*radius+3*W+bb/2-3*width,'Left')
    curve90(pos_x+ms_x-7*radius-3*W+7/2*width-
bb/2,pos_y+chann_l+radius+3/2*width,'Left','Bottom')
    via(pos_x+ms_x-8*radius-3*W+7/2*width-
bb/2,pos_y+chann_l+3/2*width,3/2*width,'Bottom')

    # Vias between output channel CDCs #1
    curve90(pos_x+2*W-3/2*width,pos_y-MZI_l-2*bb-3*radius-
3/2*width,'Top','Right')
    via(pos_x+2*W-3/2*width+radius,pos_y-MZI_l-2*bb-2*radius-
3/2*width,radius,'Right')
    curve90(pos_x+2*W-3/2*width+2*radius,pos_y-MZI_l-2*bb-2*radius-
3/2*width,'Right','Bottom')
    via(pos_x+2*W-3/2*width+3*radius,pos_y-MZI_l-2*bb-3*radius-
3/2*width,chann_l,'Bottom')
    curve90(pos_x+2*W-3/2*width+3*radius,pos_y-MZI_l-chann_l-2*bb-3*radius-
3/2*width,'Bottom','Right')
    via(pos_x+2*W-3/2*width+4*radius,pos_y-MZI_l-chann_l-2*bb-4*radius-
3/2*width,4*W-5*width+bb/2+2*radius,'Right')
    curve90(pos_x+6*W-13/2*width+6*radius+bb/2,pos_y-MZI_l-chann_l-2*bb-4*radius-
3/2*width,'Right','Top')


    # Vias between output channel CDCs #2
    # To the right of the crossing
    curve90(pos_x+W-width/2+bb/2+radius,pos_y-radius-width/2+radius-
bb/2,'Bottom','Left')
    # To the next mini-structure
    via(pos_x+W-width/2+bb/2+radius,pos_y-radius+chann_l-bb/2-
width/2+radius,radius+width,'Top')
    curve90(pos_x+W-width/2+bb/2+radius,pos_y+chann_l-
bb/2+radius+width/2,'Top','Right')
    via(pos_x+W-width/2+bb/2+2*radius,pos_y+chann_l+2*radius-bb/2+width/2,W-
width+2*radius,'Right')
    curve90(pos_x+2*W-3/2*width+bb/2+4*radius,pos_y+chann_l+2*radius-
bb/2+width/2,'Right','Bottom')
    via(pos_x+2*W-3/2*width+bb/2+5*radius,pos_y+chann_l-
bb/2+radius+width/2,chann_l+radius+width,'Bottom')
    curve90(pos_x+2*W-3/2*width+bb/2+5*radius,pos_y-width/2-
bb/2,'Bottom','Right')
    via(pos_x+2*W-3/2*width+bb/2+6*radius,pos_y-width/2-bb/2-
radius,radius+W,'Right')

    # Vias between output channel CDCs #3
    # To the right of the crossing
    via(pos_x+W-width/2+bb/2,pos_y-(MZI_l+3/2*bb+2*radius+width),W-
width+3*radius,'Right')
    curve90(pos_x+2*W-3/2*width+bb/2+3*radius,pos_y-
(MZI_l+3/2*bb+2*radius+width),'Right','Bottom')
    via(pos_x+2*W-3/2*width+bb/2+4*radius,pos_y-
(MZI_l+3/2*bb+3*radius+width),2*radius+chann_l-bb/2,'Bottom')
    curve90(pos_x+2*W-3/2*width+bb/2+4*radius,pos_y-MZI_l-chann_l-bb-5*radius-
width,'Bottom','Right')
    via(pos_x+2*W-3/2*width+bb/2+5*radius,pos_y-MZI_l-chann_l-bb-6*radius-
width,2*W-radius-2*width,'Right')
    curve90(pos_x+4*W-7/2*width+bb/2+4*radius,pos_y-MZI_l-chann_l-bb-6*radius-
width,'Right','Top')
    # To the next mini-structure
```

```
    curve90(pos_x+4*W-7/2*width+bb/2+5*radius,pos_y-
(MZI_l+3/2*bb+3*radius+width),'Top','Right')


  elif structure_type == 'final':
    CDC_channel(pos_x,pos_y)  # Input CDC channel
    # From input CDC channel to MZI
    via(pos_x+W-width/2,pos_y,radius+width/2,'Bottom')
    crossing(pos_x+W-width/2,pos_y-radius-width/2-bb/2)

    # MZIs and its connections
    # MZI 1 (the one on the left)
    MZI(pos_x+W-width,pos_y-(MZI_l+bb+radius+width/2))
    # I (MZI interconnection)
    curve90(pos_x+2*W-1.5*width,pos_y-(MZI_l+bb+radius+width/2),'Bottom','Right')
    via(pos_x+2*W-1.5*width+radius,pos_y-(MZI_l+2*radius+bb+width/2),3*radius,
'Right')
    curve90(pos_x+2*W-3/2*width+bb/2+2*radius,pos_y-
(MZI_l+bb+2*radius+width/2),'right','top')
    # MZI 2 (the one on the right)
    MZI(pos_x+2*W-2*width+bb/2+3*radius,pos_y-(MZI_l+bb+radius+width/2))
    # P1 (to output channel CDC 1)
    via(pos_x+W-width/2,pos_y-(MZI_l+bb+radius+width/2),radius+width/2,'Bottom')
    crossing(pos_x+W-width/2,pos_y-(MZI_l+3/2*bb+2*radius+width))
    via(pos_x+W-width/2,pos_y-
(MZI_l+2*bb+2*radius+width),radius+width/2,'Bottom')
    # P2 (to output channel CDC 2)
    via(pos_x+2*W-3/2*width+bb/2+3*radius,pos_y-bb-radius-
width/2,chann_l+bb/2+radius,'Top')
    curve90(pos_x+2*W-3/2*width+bb/2+3*radius,pos_y+chann_l-bb/2-
width/2,'Top','Left')
    curve90(pos_x+2*W-3/2*width+bb/2+2*radius,pos_y+chann_l-bb/2-
width/2+radius,'Left','Bottom')
    # P3 (to output channel CDC 3)
    curve90(pos_x+3*W-5/2*width+bb/2+3*radius,pos_y-
(bb+radius+width/2),'Top','Right')
    curve90(pos_x+3*W-5/2*width+bb/2+4*radius,pos_y-
(bb+width/2),'Right','Bottom')
    via(pos_x+3*W-5/2*width+bb/2+5*radius,pos_y-
(bb+radius+width/2),MZI_l+bb/2+2*radius+width/2, 'Bottom')

    # Output channel CDCs
    CDC_channel(pos_x+W-width,pos_y-MZI_l-chann_l-2*bb-3*radius-3/2*width)
# Output Channel CDC 1
    CDC_channel(pos_x+W-width+bb/2+radius,pos_y-width/2-bb/2)
# Output Channel CDC 2
    CDC_channel(pos_x+3*W-3*width+bb/2+5*radius,pos_y-MZI_l-chann_l-3/2*bb-
3*radius-width)      # Output Channel CDC 3

    # Vias between output channel CDCs #2
    # To the right of the crossing
    curve90(pos_x+W-width/2+bb/2+radius,pos_y-radius-width/2+radius-
bb/2,'Bottom','Left')

    # Vias between output channel CDCs #3
    # To the right of the crossing
    via(pos_x+W-width/2+bb/2,pos_y-(MZI_l+3/2*bb+2*radius+width),W-
width+3*radius,'Right')
```

```
    curve90(pos_x+2*W-3/2*width+bb/2+3*radius,pos_y-
(MZI_l+3/2*bb+2*radius+width),'Right','Bottom')
    via(pos_x+2*W-3/2*width+bb/2+4*radius,pos_y-
(MZI_l+3/2*bb+3*radius+width),2*radius+chann_l-bb/2,'Bottom')
    curve90(pos_x+2*W-3/2*width+bb/2+4*radius,pos_y-MZI_l-chann_l-bb-5*radius-
width,'Bottom','Right')
    via(pos_x+2*W-3/2*width+bb/2+5*radius,pos_y-MZI_l-chann_l-bb-6*radius-
width,2*W-radius-2*width,'Right')
    curve90(pos_x+4*W-7/2*width+bb/2+4*radius,pos_y-MZI_l-chann_l-bb-6*radius-
width,'Right','Top')


  else:
    raise ValueError('The input structure_type parameter is not valid.\n \
    You have to type "Initial", "Contiguous" or "Final" for structure_type')




'''
Builds a WSS Band Structure, made by a specified number of channel structures,
the 3 band CDCs and one input band CDC
Inputs: pos_x = 0 -> Initial position on the x-axis (Smallest of the 1st input
channel CDC)
        pos_y = 0 -> Initial position on the y-axis (Smallest of the 1st input
channel CDC)
        n = 8 -> Number of channels that compose the band full structure
        band_type = 'Initial' -> Type of desired band structure. 'Contiguous' for
a structure connected to another on its right,
                                 'Initial' for one connected to another on its
right and to the ports on its left,
                                 and 'Final' for the last one which does not
connect to another channel on its right
                                 It does not distinguish between uppercase and
lowercase letters
        testing = True -> Indicates whether to include testing ports in the band
structure
                          True if there is testing, and false if there is not
Outputs: bs_x -> Position of the next structure in relation to the position of
the placed band structure
'''
def band_struct(pos_x = 0, pos_y = 0, n = 8, band_type = 'Initial'):
  band_type = band_type.lower()

  if n >= 3:

    if band_type == 'initial':
      bs_x = n*ms_x+19*radius+11*width+4*W  # Distance to the position of the
next band structure

      # Placing all the channels
      channel_struct(pos_x,pos_y,'Initial')
      for i in range(n-2):
        channel_struct(pos_x+ms_x*(i+1),pos_y, 'Contiguous')
      channel_struct(pos_x+(n-1)*ms_x,pos_y,'Final')

      # Placing the band CDCs
      CDC_band(pos_x-3*radius-W,pos_y-MZI_l-chann_l-2*bb-4*radius-5/2*width)
# Input band CDC
```

```
      CDC_band(pos_x+n*ms_x+6*radius+5*width+2*bb,pos_y-band_l+chann_l-
bb/2+radius+19/2*width)     # Output band CDC #1
      CDC_band(pos_x+n*ms_x+radius+2*width,pos_y-band_l+chann_l-
bb/2+radius+width/2)               # Output band CDC #2
      CDC_band(pos_x+n*ms_x+6*radius+5*width,pos_y-MZI_l-chann_l-bb-4*radius-
5/2*width)            # Output band CDC #3

      # Placing connections between band and channel CDCs
      # Input CDC connection (from band to channel)
      boot(pos_x-3*radius-width/2,pos_y+band_l-MZI_l-chann_l-2*bb-4*radius-
5/2*width,'Top','Right','True')
      via(pos_x-2*radius+width/2,pos_y+band_l-MZI_l-chann_l-2*bb-4*radius-
5/2*width,band_l-MZI_l-chann_l-2*bb-4*radius-5/2*width,'Bottom')
      curve90(pos_x-2*radius+width/2,pos_y,'Bottom','Right')
      curve90(pos_x-radius+width/2,pos_y-radius,'Right','Top')
      # Output CDC #1 connection (from channel to band)
      curve90(pos_x+n*ms_x-7*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-3*radius-
3/2*width,'Top','Right')
      via(pos_x+n*ms_x-6*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-2*radius-
3/2*width,radius,'Right')
      curve90(pos_x+n*ms_x-5*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-2*radius-
3/2*width,'Right','Bottom')
      via(pos_x+n*ms_x-4*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-3*radius-
3/2*width,chann_l,'Bottom')
      curve90(pos_x+n*ms_x-4*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-chann_l-2*bb-
3*radius-3/2*width,'Bottom','Right')
      via(pos_x+n*ms_x-3*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-chann_l-2*bb-
4*radius-3/2*width,6*radius+2*width+5/2*bb+2*W,'Right')
      curve90(pos_x+n*ms_x+3*radius+11/2*width+2*bb,pos_y-MZI_l-chann_l-2*bb-
4*radius-3/2*width,'Right','Top')
      via(pos_x+n*ms_x+4*radius+11/2*width+2*bb,pos_y-MZI_l-chann_l-2*bb-
3*radius-3/2*width,2*chann_l+MZI_l+3/2*bb+4*radius+11*width,'Top')
      curve90(pos_x+n*ms_x+4*radius+11/2*width+2*bb,pos_y+chann_l-
bb/2+radius+19/2*width,'Top','Right')
      curve90(pos_x+n*ms_x+5*radius+11/2*width+2*bb,pos_y+chann_l-
bb/2+2*radius+19/2*width,'Right','Bottom')
      # Output CDC #2 connection (from band to channel)
      curve90(pos_x+n*ms_x+radius+5/2*width,pos_y+chann_l-
bb/2+radius+width/2,'Top','Left')
      via(pos_x+n*ms_x+5/2*width,pos_y+chann_l-
bb/2+2*radius+width/2,5*radius+3*W-2*width,'Left')
      curve90(pos_x+n*ms_x-5*radius+9/2*width-3*W,pos_y+chann_l-
bb/2+2*radius+width/2,'Left','Bottom')
      via(pos_x+n*ms_x-6*radius+9/2*width-3*W,pos_y+chann_l-
bb/2+radius+width/2,radius+width,'Bottom')
      #Output CDC #3 connection (from channel to band)
      via(pos_x+n*ms_x-2*radius+3/2*width,pos_y-MZI_l-3/2*bb-3*radius-
width,MZI_l+bb/2+3*radius+width,'Top')
      boot(pos_x+n*ms_x-2*radius+3/2*width,pos_y-bb,'Top','Right')
      via(pos_x+n*ms_x-radius+5/2*width,pos_y-
bb,MZI_l+chann_l+bb/2+3*radius+2*width,'Bottom')
      curve90(pos_x+n*ms_x-radius+5/2*width,pos_y-MZI_l-chann_l-3/2*bb-3*radius-
2*width,'Bottom','Right')
      via(pos_x+n*ms_x+5/2*width,pos_y-MZI_l-chann_l-3/2*bb-4*radius-
2*width,3*width+3*radius,'Right')
      curve90(pos_x+n*ms_x+11/2*width+3*radius,pos_y-MZI_l-chann_l-3/2*bb-
4*radius-2*width,'Right','Top')
      via(pos_x+n*ms_x+11/2*width+4*radius,pos_y-MZI_l-chann_l-3/2*bb-3*radius-
2*width,band_l+radius-width/2,'Top')
```

```
      curve90(pos_x+n*ms_x+11/2*width+4*radius,pos_y+band_l-MZI_l-chann_l-3/2*bb-
2*radius-5/2*width,'Top','Right')
      curve90(pos_x+n*ms_x+11/2*width+5*radius,pos_y+band_l-MZI_l-chann_l-3/2*bb-
radius-5/2*width,'Right','Bottom')

      # Placing connections to the next band CDCs (on its right)
      # Input CDC (from right to left)
      curve90(pos_x-3*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-4*radius-
5/2*width,'Bottom','Right')
      via(pos_x-2*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-5*radius-
5/2*width,bs_x-4*radius,'Right')
      curve90(pos_x+bs_x-6*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-5*radius-
5/2*width,'Right','Top')
      via(pos_x+bs_x-5*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-4*radius-
5/2*width,band_l,'Top')
      curve90(pos_x+bs_x-5*radius-W+width/2,pos_y+band_l-MZI_l-chann_l-2*bb-
4*radius-5/2*width,'Top','Right')
      curve90(pos_x+bs_x-4*radius-W+width/2,pos_y+band_l-MZI_l-chann_l-2*bb-
3*radius-5/2*width,'Right','Bottom')
      # Output CDC #1 (from right to left) (1st and 2nd crossings between band
CDCs #2 and #3 respectively)
      curve90(pos_x+n*ms_x+6*radius+9/2*width+2*bb+W,pos_y-band_l+chann_l-
bb/2+radius+19/2*width,'Bottom','Right')
      curve90(pos_x+n*ms_x+7*radius+9/2*width+2*bb+W,pos_y-band_l+chann_l-
bb/2+19/2*width,'Right','Top')
      via(pos_x+n*ms_x+8*radius+9/2*width+2*bb+W,pos_y-band_l+chann_l-
bb/2+radius+19/2*width,band_l-6*width,'Top')
      curve90(pos_x+n*ms_x+8*radius+9/2*width+2*bb+W,pos_y+chann_l-
bb/2+radius+7/2*width,'Top','Right')
      curve90(pos_x+n*ms_x+9*radius+9/2*width+2*bb+W,pos_y+chann_l-
bb/2+2*radius+7/2*width,'Right','Top')

via(pos_x+n*ms_x+6*radius+9/2*width+W+3*bb,pos_y+chann_l+3*radius+7/2*width+3/2*b
b,bb/2-radius,'Top')

curve90(pos_x+n*ms_x+6*radius+9/2*width+W+3*bb,pos_y+chann_l+2*radius+7/2*width+2
*bb,'Top','Right')

via(pos_x+n*ms_x+7*radius+9/2*width+W+3*bb,pos_y+chann_l+3*radius+7/2*width+2*bb,
bb/2-radius,'Right')
      # Output CDC #2 (from right to left)
      boot(pos_x+n*ms_x+radius+3/2*width+W,pos_y-band_l+chann_l-
bb/2+radius+width/2,'Bottom','Right')
      via(pos_x+n*ms_x+2*radius+5/2*width+W,pos_y-band_l+chann_l-
bb/2+radius+width/2,band_l+radius+3*width+bb/2,'Top')

curve90(pos_x+n*ms_x+2*radius+5/2*width+W,pos_y+chann_l+2*radius+7/2*width,'Top',
'Right')

via(pos_x+n*ms_x+3*radius+5/2*width+W,pos_y+chann_l+3*radius+7/2*width,3*radius+2
*width-bb/2,'Right')

crossing(pos_x+n*ms_x+6*radius+9/2*width+W,pos_y+chann_l+3*radius+7/2*width)

crossing(pos_x+n*ms_x+6*radius+9/2*width+W+bb,pos_y+chann_l+3*radius+7/2*width)

crossing(pos_x+n*ms_x+6*radius+9/2*width+W+2*bb,pos_y+chann_l+3*radius+7/2*width)

crossing(pos_x+n*ms_x+6*radius+9/2*width+W+3*bb,pos_y+chann_l+3*radius+7/2*width)
```

```
      # Output CDC #3 (from right to left) (1st crossing between band CDCs #2)
      curve90(pos_x+n*ms_x+6*radius+9/2*width+W,pos_y-MZI_l-chann_l-bb-4*radius-
5/2*width,'Bottom','Right')
      curve90(pos_x+n*ms_x+7*radius+9/2*width+W,pos_y-MZI_l-chann_l-bb-5*radius-
5/2*width,'Right','Top')
      via(pos_x+n*ms_x+8*radius+9/2*width+W,pos_y-MZI_l-chann_l-bb-4*radius-
5/2*width,2*chann_l+MZI_l+5*radius+6*width+bb/2,'Top')
      curve90(pos_x+n*ms_x+8*radius+9/2*width+W,pos_y+chann_l-
bb/2+radius+7/2*width,'Top','Right')
      curve90(pos_x+n*ms_x+9*radius+9/2*width+W,pos_y+chann_l-
bb/2+2*radius+7/2*width,'Right','Top')

via(pos_x+n*ms_x+6*radius+9/2*width+W+bb,pos_y+chann_l+3*radius+7/2*width+bb/2,bb
/2-radius,'Top')

curve90(pos_x+n*ms_x+6*radius+9/2*width+W+bb,pos_y+chann_l+2*radius+7/2*width+bb,
'Top','Right')

via(pos_x+n*ms_x+7*radius+9/2*width+W+bb,pos_y+chann_l+3*radius+7/2*width+bb,bb/2
-radius,'Right')

crossing(pos_x+n*ms_x+6*radius+9/2*width+W+2*bb,pos_y+chann_l+bb+3*radius+7/2*wid
th)

crossing(pos_x+n*ms_x+6*radius+9/2*width+W+3*bb,pos_y+chann_l+bb+3*radius+7/2*wid
th)

      # Connections to the ports
      # for the first addition (input channel CDC) (Testing 1)
      via(pos_x+(n-1)*ms_x+width/2,pos_y+chann_l,5/2*width+radius,'Top')
      curve90(pos_x+(n-
1)*ms_x+width/2,pos_y+chann_l+radius+5/2*width,'Top','Left')
      via(pos_x+(n-1)*ms_x+width/2-radius,pos_y+chann_l+2*radius+5/2*width,(n-
1)*ms_x+4*radius+W+width,'Left')
      curve90(pos_x-width/2-5*radius-
W,pos_y+chann_l+2*radius+5/2*width,'Left','Bottom')
      via(pos_x-width/2-6*radius-
W,pos_y+chann_l+radius+5/2*width,MZI_l+2*chann_l-
3*port_d+11/2*width+2*bb+5*radius,'Bottom')
      curve90(pos_x-width/2-6*radius-W,pos_y-MZI_l-chann_l+3*port_d-4*radius-
3*width-2*bb,'Bottom','Left')
      via(pos_x-width/2-7*radius-W,pos_y-MZI_l-chann_l+3*port_d-5*radius-3*width-
2*bb,5*width+5*radius,'Left')
      IO(pos_x-12*radius-W-11/2*width,pos_y-MZI_l-chann_l+3*port_d-5*radius-
3*width-2*bb,width,IO_h,IO_l,'Testing 1')
      # for the second addition (from port to input channel CDC) (Testing 2)
      IO(pos_x-12*radius-W-11/2*width,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+7*port_d,width,IO_h,IO_l,'Testing 2')
      via(pos_x-12*radius-11/2*width-W,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+7*port_d,radius+width,'Right')
      curve90(pos_x-11*radius-9/2*width-W,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+7*port_d,'Right','Top')
      via(pos_x-10*radius-9/2*width-W,pos_y-MZI_l-chann_l-2*bb-4*radius-
3*width+7*port_d,MZI_l+2*chann_l-7*port_d+7*radius+13/2*width+9/2*bb,'Top')
      curve90(pos_x-10*radius-9/2*width-
W,pos_y+chann_l+3*radius+7/2*width+5/2*bb,'Top','Right')
      via(pos_x-9*radius-9/2*width-
W,pos_y+chann_l+4*radius+7/2*width+5/2*bb,n*ms_x+15*radius+9*width+2*W+7/2*bb,'Ri
ght')
```

```
      # for the 3rd addition (from port to input channel CDC) (Testing 3)
      IO(pos_x-12*radius-W-11/2*width,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+8*port_d,width,IO_h,IO_l,'Testing 3')
      curve90(pos_x-12*radius-W-11/2*width,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+8*port_d,'Right','Top')
      via(pos_x-11*radius-W-11/2*width,pos_y-MZI_l-chann_l-2*bb-4*radius-
3*width+8*port_d,pos_y+MZI_l-8*port_d+2*chann_l+8*radius+15/2*width+9/2*bb,'Top')
      curve90(pos_x-11*radius-W-
11/2*width,pos_y+chann_l+4*radius+9/2*width+5/2*bb,'Top','Right')
      via(pos_x-10*radius-W-
11/2*width,pos_y+chann_l+5*radius+9/2*width+5/2*bb,n*ms_x+16*radius+10*width+2*W+
7/2*bb,'Right')
      # from Input CDC to Input Port
      curve90(pos_x-3*radius-W+width/2,pos_y+band_l-MZI_l-chann_l-2*bb-4*radius-
5/2*width,'Top','Left')
      curve90(pos_x-4*radius-W+width/2,pos_y+band_l-MZI_l-chann_l-2*bb-3*radius-
5/2*width,'Left','Bottom')
      via(pos_x-5*radius-W+width/2,pos_y+band_l-MZI_l-chann_l-2*bb-4*radius-
5/2*width,band_l+width/2-2*port_d,'Bottom')
      curve90(pos_x-5*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-4*radius-
3*width+2*port_d,'Bottom','Left')
      via(pos_x-6*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+2*port_d,6*radius+6*width,'Left')
      IO(pos_x-12*radius-W-11/2*width,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+2*port_d,width,IO_h,IO_l,'Input')
      # From output band CDC #1 to output port #1
      via(pos_x+n*ms_x+6*radius+9/2*width+2*bb+W,pos_y+chann_l-
bb/2+radius+19/2*width,2*radius-6*width+bb/2,'Top')

via(pos_x+n*ms_x+6*radius+9/2*width+W+2*bb,pos_y+chann_l+3/2*bb+3*radius+7/2*widt
h,bb/2-radius,'Top')

curve90(pos_x+n*ms_x+6*radius+9/2*width+W+2*bb,pos_y+chann_l+2*bb+2*radius+7/2*wi
dth,'Top','Left')

via(pos_x+n*ms_x+5*radius+9/2*width+W+2*bb,pos_y+chann_l+2*bb+3*radius+7/2*width,
n*ms_x+13*radius+8*width+2*W+2*bb,'Left')
      curve90(pos_x-8*radius-7/2*width-
W,pos_y+chann_l+2*bb+3*radius+7/2*width,'Left','Bottom')
      via(pos_x-9*radius-7/2*width-
W,pos_y+chann_l+2*bb+2*radius+7/2*width,2*chann_l+MZI_l-
6*port_d+4*bb+6*radius+13/2*width,'Bottom')
      curve90(pos_x-9*radius-7/2*width-W,pos_y+6*port_d-MZI_l-chann_l-2*bb-
4*radius-3*width,'Bottom','Left')
      via(pos_x-10*radius-7/2*width-W,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+6*port_d,2*radius+2*width,'Left')
      IO(pos_x-12*radius-W-11/2*width,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+6*port_d,width,IO_h,IO_l,'Output 1')
      # From output band CDC #2 to output port #2
      via(pos_x+n*ms_x+radius+3/2*width+W,pos_y+chann_l-
bb/2+radius+width/2,bb/2+radius+3*width,'Top')

curve90(pos_x+n*ms_x+radius+3/2*width+W,pos_y+chann_l+2*radius+7/2*width,'Top','L
eft')

via(pos_x+n*ms_x+3/2*width+W,pos_y+chann_l+3*radius+7/2*width,n*ms_x+2*W+6*radius
+3*width,'Left')
      curve90(pos_x-3/2*width-W-
6*radius,pos_y+chann_l+3*radius+7/2*width,'Left','Bottom')
```

```
      via(pos_x-3/2*width-W-
7*radius,pos_y+chann_l+2*radius+7/2*width,2*chann_l+MZI_l-
4*port_d+6*radius+13/2*width+2*bb,'Bottom')
      curve90(pos_x-3/2*width-W-7*radius,pos_y-chann_l-4*radius-3*width-2*bb-
MZI_l+4*port_d,'Bottom','Left')
      via(pos_x-8*radius-W-3/2*width,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+4*port_d,4*radius+4*width,'Left')
      IO(pos_x-12*radius-W-11/2*width,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+4*port_d,width,IO_h,IO_l,'Output 2')
      # From output band CDC #3 to output port #3 (Crossing between CDC bands #2)
      via(pos_x+n*ms_x+6*radius+9/2*width+W,pos_y+band_l-MZI_l-chann_l-bb-
4*radius-5/2*width,2*chann_l+MZI_l-band_l+7*radius+6*width+bb/2,'Top')

via(pos_x+n*ms_x+6*radius+9/2*width+W,pos_y+chann_l+3*radius+7/2*width+bb/2,bb/2-
radius,'Top')

curve90(pos_x+n*ms_x+6*radius+9/2*width+W,pos_y+chann_l+2*radius+7/2*width+bb,'To
p','Left')

via(pos_x+n*ms_x+5*radius+9/2*width+W,pos_y+chann_l+3*radius+7/2*width+bb,n*ms_x+
12*radius+7*width+2*W,'Left')
      curve90(pos_x-7*radius-5/2*width-
W,pos_y+chann_l+3*radius+7/2*width+bb,'Left','Bottom')
      via(pos_x-8*radius-5/2*width-
W,pos_y+chann_l+2*radius+7/2*width+bb,2*chann_l+MZI_l-
5*port_d+6*radius+13/2*width+3*bb,'Bottom')
      curve90(pos_x-8*radius-5/2*width-W,pos_y-MZI_l-chann_l-2*bb-4*radius-
3*width+5*port_d,'Bottom','Left')
      via(pos_x-9*radius-5/2*width-W,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+5*port_d,3*radius+3*width,'Left')
      IO(pos_x-12*radius-W-11/2*width,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+5*port_d,width,IO_h,IO_l,'Output 3')


      return bs_x


    elif band_type == 'contiguous':
      bs_x = n*ms_x+19*radius+11*width+4*W  # Distance to the position of the
next band structure

      # Placing all the channels
      channel_struct(pos_x,pos_y,'Initial')
      for i in range(n-2):
        channel_struct(pos_x+ms_x*(i+1),pos_y, 'Contiguous')
      channel_struct(pos_x+(n-1)*ms_x,pos_y,'Final')

      # Placing the band CDCs
      CDC_band(pos_x-3*radius-W,pos_y-MZI_l-chann_l-2*bb-4*radius-5/2*width)
# Input band CDC
      CDC_band(pos_x+n*ms_x+6*radius+5*width+2*bb,pos_y-band_l+chann_l-
bb/2+radius+19/2*width)     # Output band CDC #1
      CDC_band(pos_x+n*ms_x+radius+2*width,pos_y-band_l+chann_l-
bb/2+radius+width/2)               # Output band CDC #2
      CDC_band(pos_x+n*ms_x+6*radius+5*width,pos_y-MZI_l-chann_l-bb-4*radius-
5/2*width)            # Output band CDC #3

      # Placing connections between band and channel CDCs
      # Input CDC connection (from band to channel)
```

```
      boot(pos_x-3*radius-width/2,pos_y+band_l-MZI_l-chann_l-2*bb-4*radius-
5/2*width,'Top','Right','True')
      via(pos_x-2*radius+width/2,pos_y+band_l-MZI_l-chann_l-2*bb-4*radius-
5/2*width,band_l-MZI_l-chann_l-2*bb-4*radius-5/2*width,'Bottom')
      curve90(pos_x-2*radius+width/2,pos_y,'Bottom','Right')
      curve90(pos_x-radius+width/2,pos_y-radius,'Right','Top')
      # Output CDC #1 connection (from channel to band)
      curve90(pos_x+n*ms_x-7*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-3*radius-
3/2*width,'Top','Right')
      via(pos_x+n*ms_x-6*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-2*radius-
3/2*width,radius,'Right')
      curve90(pos_x+n*ms_x-5*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-2*radius-
3/2*width,'Right','Bottom')
      via(pos_x+n*ms_x-4*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-3*radius-
3/2*width,chann_l,'Bottom')
      curve90(pos_x+n*ms_x-4*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-chann_l-2*bb-
3*radius-3/2*width,'Bottom','Right')
      via(pos_x+n*ms_x-3*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-chann_l-2*bb-
4*radius-3/2*width,6*radius+2*width+5/2*bb+2*W,'Right')
      curve90(pos_x+n*ms_x+3*radius+11/2*width+2*bb,pos_y-MZI_l-chann_l-2*bb-
4*radius-3/2*width,'Right','Top')
      via(pos_x+n*ms_x+4*radius+11/2*width+2*bb,pos_y-MZI_l-chann_l-2*bb-
3*radius-3/2*width,2*chann_l+MZI_l+3/2*bb+4*radius+11*width,'Top')
      curve90(pos_x+n*ms_x+4*radius+11/2*width+2*bb,pos_y+chann_l-
bb/2+radius+19/2*width,'Top','Right')
      curve90(pos_x+n*ms_x+5*radius+11/2*width+2*bb,pos_y+chann_l-
bb/2+2*radius+19/2*width,'Right','Bottom')
      # Output CDC #2 connection (from band to channel)
      curve90(pos_x+n*ms_x+radius+5/2*width,pos_y+chann_l-
bb/2+radius+width/2,'Top','Left')
      via(pos_x+n*ms_x+5/2*width,pos_y+chann_l-
bb/2+2*radius+width/2,5*radius+3*W-2*width,'Left')
      curve90(pos_x+n*ms_x-5*radius+9/2*width-3*W,pos_y+chann_l-
bb/2+2*radius+width/2,'Left','Bottom')
      via(pos_x+n*ms_x-6*radius+9/2*width-3*W,pos_y+chann_l-
bb/2+radius+width/2,radius+width,'Bottom')
      #Output CDC #3 connection (from channel to band)
      via(pos_x+n*ms_x-2*radius+3/2*width,pos_y-MZI_l-3/2*bb-3*radius-
width,MZI_l+bb/2+3*radius+width,'Top')
      boot(pos_x+n*ms_x-2*radius+3/2*width,pos_y-bb,'Top','Right')
      via(pos_x+n*ms_x-radius+5/2*width,pos_y-
bb,MZI_l+chann_l+bb/2+3*radius+2*width,'Bottom')
      curve90(pos_x+n*ms_x-radius+5/2*width,pos_y-MZI_l-chann_l-3/2*bb-3*radius-
2*width,'Bottom','Right')
      via(pos_x+n*ms_x+5/2*width,pos_y-MZI_l-chann_l-3/2*bb-4*radius-
2*width,3*width+3*radius,'Right')
      curve90(pos_x+n*ms_x+11/2*width+3*radius,pos_y-MZI_l-chann_l-3/2*bb-
4*radius-2*width,'Right','Top')
      via(pos_x+n*ms_x+11/2*width+4*radius,pos_y-MZI_l-chann_l-3/2*bb-3*radius-
2*width,band_l+radius-width/2,'Top')
      curve90(pos_x+n*ms_x+11/2*width+4*radius,pos_y+band_l-MZI_l-chann_l-3/2*bb-
2*radius-5/2*width,'Top','Right')
      curve90(pos_x+n*ms_x+11/2*width+5*radius,pos_y+band_l-MZI_l-chann_l-3/2*bb-
radius-5/2*width,'Right','Bottom')

      # Placing connections to the next band CDCs (on its right)
      # Input CDC (from right to left)
      curve90(pos_x-3*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-4*radius-
5/2*width,'Bottom','Right')
```

```
      via(pos_x-2*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-5*radius-
5/2*width,bs_x-4*radius,'Right')
      curve90(pos_x+bs_x-6*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-5*radius-
5/2*width,'Right','Top')
      via(pos_x+bs_x-5*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-4*radius-
5/2*width,band_l,'Top')
      curve90(pos_x+bs_x-5*radius-W+width/2,pos_y+band_l-MZI_l-chann_l-2*bb-
4*radius-5/2*width,'Top','Right')
      curve90(pos_x+bs_x-4*radius-W+width/2,pos_y+band_l-MZI_l-chann_l-2*bb-
3*radius-5/2*width,'Right','Bottom')
      # Output CDC #1 (from right to left) (1st and 2nd crossings between band
CDCs #2 and #3 respectively)
      curve90(pos_x+n*ms_x+6*radius+9/2*width+2*bb+W,pos_y-band_l+chann_l-
bb/2+radius+19/2*width,'Bottom','Right')
      curve90(pos_x+n*ms_x+7*radius+9/2*width+2*bb+W,pos_y-band_l+chann_l-
bb/2+19/2*width,'Right','Top')
      via(pos_x+n*ms_x+8*radius+9/2*width+2*bb+W,pos_y-band_l+chann_l-
bb/2+radius+19/2*width,band_l-6*width,'Top')
      curve90(pos_x+n*ms_x+8*radius+9/2*width+2*bb+W,pos_y+chann_l-
bb/2+radius+7/2*width,'Top','Right')
      #via(pos_x+n*ms_x+9*radius+9/2*width+2*bb+W,pos_y+chann_l-
bb/2+2*radius+7/2*width,-2*radius+bb,'Right')
      curve90(pos_x+n*ms_x+9*radius+9/2*width+2*bb+W,pos_y+chann_l-
bb/2+2*radius+7/2*width,'Right','Top')

via(pos_x+n*ms_x+6*radius+9/2*width+W+3*bb,pos_y+chann_l+3*radius+7/2*width+3/2*b
b,bb/2-radius,'Top')

curve90(pos_x+n*ms_x+6*radius+9/2*width+W+3*bb,pos_y+chann_l+2*radius+7/2*width+2
*bb,'Top','Right')

via(pos_x+n*ms_x+7*radius+9/2*width+W+3*bb,pos_y+chann_l+3*radius+7/2*width+2*bb,
bb/2-radius,'Right')
      # Output CDC #2 (from right to left)
      boot(pos_x+n*ms_x+radius+3/2*width+W,pos_y-band_l+chann_l-
bb/2+radius+width/2,'Bottom','Right')
      via(pos_x+n*ms_x+2*radius+5/2*width+W,pos_y-band_l+chann_l-
bb/2+radius+width/2,band_l+radius+3*width+bb/2,'Top')

curve90(pos_x+n*ms_x+2*radius+5/2*width+W,pos_y+chann_l+2*radius+7/2*width,'Top',
'Right')

via(pos_x+n*ms_x+3*radius+5/2*width+W,pos_y+chann_l+3*radius+7/2*width,3*radius+2
*width-bb/2,'Right')

crossing(pos_x+n*ms_x+6*radius+9/2*width+W,pos_y+chann_l+3*radius+7/2*width)

crossing(pos_x+n*ms_x+6*radius+9/2*width+W+bb,pos_y+chann_l+3*radius+7/2*width)

crossing(pos_x+n*ms_x+6*radius+9/2*width+W+2*bb,pos_y+chann_l+3*radius+7/2*width)

crossing(pos_x+n*ms_x+6*radius+9/2*width+W+3*bb,pos_y+chann_l+3*radius+7/2*width)
      # Output CDC #3 (from right to left) (1st crossing between band CDCs #2)
      curve90(pos_x+n*ms_x+6*radius+9/2*width+W,pos_y-MZI_l-chann_l-bb-4*radius-
5/2*width,'Bottom','Right')
      curve90(pos_x+n*ms_x+7*radius+9/2*width+W,pos_y-MZI_l-chann_l-bb-5*radius-
5/2*width,'Right','Top')
      via(pos_x+n*ms_x+8*radius+9/2*width+W,pos_y-MZI_l-chann_l-bb-4*radius-
5/2*width,2*chann_l+MZI_l+5*radius+6*width+bb/2,'Top')
```

```
      curve90(pos_x+n*ms_x+8*radius+9/2*width+W,pos_y+chann_l-
bb/2+radius+7/2*width,'Top','Right')
      curve90(pos_x+n*ms_x+9*radius+9/2*width+W,pos_y+chann_l-
bb/2+2*radius+7/2*width,'Right','Top')

via(pos_x+n*ms_x+6*radius+9/2*width+W+bb,pos_y+chann_l+3*radius+7/2*width+bb/2,bb
/2-radius,'Top')

curve90(pos_x+n*ms_x+6*radius+9/2*width+W+bb,pos_y+chann_l+2*radius+7/2*width+bb,
'Top','Right')

via(pos_x+n*ms_x+7*radius+9/2*width+W+bb,pos_y+chann_l+3*radius+7/2*width+bb,bb/2
-radius,'Right')

crossing(pos_x+n*ms_x+6*radius+9/2*width+W+2*bb,pos_y+chann_l+bb+3*radius+7/2*wid
th)

crossing(pos_x+n*ms_x+6*radius+9/2*width+W+3*bb,pos_y+chann_l+bb+3*radius+7/2*wid
th)

      # Connections to the previous band CDCs
      # Output CDC #1 (from right band to left band)
      via(pos_x+n*ms_x+6*radius+9/2*width+2*bb+W,pos_y+chann_l-
bb/2+radius+19/2*width,2*radius-6*width+bb/2,'Top')

via(pos_x+n*ms_x+6*radius+9/2*width+W+2*bb,pos_y+chann_l+3/2*bb+3*radius+7/2*widt
h,bb/2-radius,'Top')

curve90(pos_x+n*ms_x+6*radius+9/2*width+W+2*bb,pos_y+chann_l+2*bb+2*radius+7/2*wi
dth,'Top','Left')

via(pos_x+n*ms_x+5*radius+9/2*width+W+2*bb,pos_y+chann_l+2*bb+3*radius+7/2*width,
ms_x+5*radius+4*width+W+3/2*bb,'Left')
      crossing(pos_x+(n-1)*ms_x+width/2,pos_y+chann_l+2*bb+3*radius+7/2*width)
      via(pos_x+(n-1)*ms_x+width/2-bb/2,pos_y+chann_l+2*bb+3*radius+7/2*width,(n-
1)*ms_x+W+width-bb+4*radius,'Left')

      # Output CDC #2 (from the contiguous to the previous one)
      via(pos_x+n*ms_x+radius+3/2*width+W,pos_y+chann_l-
bb/2+radius+width/2,bb/2+radius+3*width,'Top')

curve90(pos_x+n*ms_x+radius+3/2*width+W,pos_y+chann_l+2*radius+7/2*width,'Top','L
eft')
      via(pos_x+n*ms_x+3/2*width+W,pos_y+chann_l+3*radius+7/2*width,ms_x+width+W-
bb/2,'Left')
      crossing(pos_x+(n-1)*ms_x+width/2,pos_y+chann_l+3*radius+7/2*width)
      via(pos_x+(n-1)*ms_x+width/2-bb/2,pos_y+chann_l+3*radius+7/2*width,(n-
1)*ms_x+width-bb+4*radius+W,'Left')


      # Output CDC #3 (from the contiguous to the previous one) (Crossing on band
CDC #2 going to the next band)
      via(pos_x+n*ms_x+6*radius+9/2*width+W,pos_y+band_l-MZI_l-chann_l-bb-
4*radius-5/2*width,2*chann_l+MZI_l-band_l+7*radius+6*width+bb/2,'Top')

via(pos_x+n*ms_x+6*radius+9/2*width+W,pos_y+chann_l+3*radius+7/2*width+bb/2,bb/2-
radius,'Top')
```

```
curve90(pos_x+n*ms_x+6*radius+9/2*width+W,pos_y+chann_l+2*radius+7/2*width+bb,'To
p','Left')

via(pos_x+n*ms_x+5*radius+9/2*width+W,pos_y+chann_l+3*radius+7/2*width+bb,ms_x+5*
radius+4*width+W-bb/2,'Left')
      crossing(pos_x+(n-1)*ms_x+width/2,pos_y+chann_l+3*radius+7/2*width+bb)
      via(pos_x+(n-1)*ms_x+width/2-bb/2,pos_y+chann_l+3*radius+7/2*width+bb,(n-
1)*ms_x+width+4*radius+W-bb,'Left')

      # Connections to the ports
      # For the second addition (input channel CDC to the previous band) (Testing
2) (Crossing on the band CDCs going to previous bands)
      via(pos_x+(n-1)*ms_x+width/2,pos_y+chann_l,3*radius+7/2*width-bb/2,'Top')
      curve90(pos_x+(n-
1)*ms_x+width/2,pos_y+chann_l+3*radius+7/2*width+5/2*bb,'Top','Left')
      via(pos_x+(n-1)*ms_x+width/2-
radius,pos_y+chann_l+4*radius+7/2*width+5/2*bb,(n-1)*ms_x+width+4*radius+W-bb/2-
radius,'Left')
      # For the 3rd addition (input channel CDC to the previous band) (Testing 3)
      via(pos_x+bs_x-4*radius-
W+bb/2,pos_y+chann_l+5*radius+9/2*width+5/2*bb,bs_x+width/2,'Left')


      return bs_x


    elif band_type == 'final':
      bs_x = n*ms_x+19*radius+11*width+4*W  # Distance to the position of the
next band structure

      # Placing all the channels
      channel_struct(pos_x,pos_y,'Initial')
      for i in range(n-2):
        channel_struct(pos_x+ms_x*(i+1),pos_y, 'Contiguous')
      channel_struct(pos_x+(n-1)*ms_x,pos_y,'Final')

      # Placing the band CDCs
      CDC_band(pos_x-3*radius-W,pos_y-MZI_l-chann_l-2*bb-4*radius-5/2*width)    #
Input band CDC

      # Placing connections between band and channel CDCs
      # Input CDC connection (from band to channel)
      boot(pos_x-3*radius-width/2,pos_y+band_l-MZI_l-chann_l-2*bb-4*radius-
5/2*width,'Top','Right','True')
      via(pos_x-2*radius+width/2,pos_y+band_l-MZI_l-chann_l-2*bb-4*radius-
5/2*width,band_l-MZI_l-chann_l-2*bb-4*radius-5/2*width,'Bottom')
      curve90(pos_x-2*radius+width/2,pos_y,'Bottom','Right')
      curve90(pos_x-radius+width/2,pos_y-radius,'Right','Top')
      # Output CDC #1 connection (from channel to the previous band)
      curve90(pos_x+n*ms_x-7*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-3*radius-
3/2*width,'Top','Right')
      via(pos_x+n*ms_x-6*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-2*radius-
3/2*width,radius,'Right')
      curve90(pos_x+n*ms_x-5*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-2*radius-
3/2*width,'Right','Bottom')
      via(pos_x+n*ms_x-4*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-2*bb-3*radius-
3/2*width,chann_l+width,'Bottom')
```

```
      curve90(pos_x+n*ms_x-4*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-chann_l-2*bb-
3*radius-5/2*width,'Bottom','Right')
      via(pos_x+n*ms_x-3*radius+7/2*width-2*W-bb/2,pos_y-MZI_l-chann_l-2*bb-
4*radius-5/2*width,radius-width+2*W+bb/2,'Right')
      curve90(pos_x+n*ms_x-2*radius+5/2*width,pos_y-MZI_l-chann_l-2*bb-4*radius-
5/2*width,'Right','Top')
      via(pos_x+n*ms_x-radius+5/2*width,pos_y-MZI_l-chann_l-2*bb-3*radius-
5/2*width,MZI_l+2*chann_l+4*bb+5*radius+6*width,'Top')
      curve90(pos_x+n*ms_x-
radius+5/2*width,pos_y+chann_l+2*bb+2*radius+7/2*width,'Top','Left')
      via(pos_x+n*ms_x-
2*radius+5/2*width,pos_y+chann_l+2*bb+3*radius+7/2*width,ms_x+2*width-bb/2-
2*radius,'Left')
      crossing(pos_x+(n-1)*ms_x+width/2,pos_y+chann_l+2*bb+3*radius+7/2*width)
      via(pos_x+(n-1)*ms_x+width/2-bb/2,pos_y+chann_l+2*bb+3*radius+7/2*width,(n-
1)*ms_x+W+width-bb+4*radius,'Left')
      # Output CDC #2 connection (from channel to the previous band)
      via(pos_x+(n-1)*ms_x+W-width/2+bb/2+radius,pos_y+chann_l-bb/2-
width/2,2*radius+4*width+bb/2,'Top')
      curve90(pos_x+(n-1)*ms_x+W-
width/2+bb/2+radius,pos_y+chann_l+7/2*width+2*radius,'Top','Left')
      via(pos_x+(n-1)*ms_x+W-width/2+bb/2,pos_y+chann_l+7/2*width+3*radius,bb/2-
radius+width,'Left')
      via(pos_x+(n-1)*ms_x+width/2-bb/2,pos_y+chann_l+3*radius+7/2*width,(n-
1)*ms_x+width-bb+4*radius+W,'Left')
      crossing(pos_x+(n-1)*ms_x+width/2,pos_y+chann_l+3*radius+7/2*width)
      #Output CDC #3 connection (from channel to the previous band)
      via(pos_x+n*ms_x-2*radius+3/2*width,pos_y-MZI_l-3/2*bb-3*radius-
width,MZI_l+chann_l+5*radius+9/2*width+5/2*bb,'Top')
      curve90(pos_x+n*ms_x-
2*radius+3/2*width,pos_y+chann_l+bb+2*radius+7/2*width,'Top','Left')
      via(pos_x+n*ms_x-
3*radius+3/2*width,pos_y+chann_l+7/2*width+3*radius+bb,ms_x-3*radius+width-bb/2
,'Left')
      crossing(pos_x+(n-1)*ms_x+width/2,pos_y+chann_l+3*radius+7/2*width+bb)
      via(pos_x+(n-1)*ms_x+width/2-bb/2,pos_y+chann_l+3*radius+7/2*width+bb,(n-
1)*ms_x+width+4*radius+W-bb,'Left')

      # Connections to the ports
      # For the 3rd addition (input channel CDC to the previous band) (Testing 3)
      via(pos_x+(n-1)*ms_x+width/2,pos_y+chann_l,3*radius+7/2*width-bb/2,'Top')
      via(pos_x+(n-
1)*ms_x+width/2,pos_y+chann_l+3*radius+7/2*width+5/2*bb,radius+width,'Top')
      curve90(pos_x+(n-
1)*ms_x+width/2,pos_y+chann_l+4*radius+9/2*width+5/2*bb,'Top','Left')
      via(pos_x+(n-1)*ms_x+width/2-
radius,pos_y+chann_l+5*radius+9/2*width+5/2*bb,(n-1)*ms_x+width/2+4*radius+W-
bb/2-radius,'Left')
      # Testing 4 (Addition of the last input band CDC) (from the CDC to the
left)
      via(pos_x-3*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-4*radius-
5/2*width,radius+width,'Bottom')
      curve90(pos_x-3*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-5*radius-
7/2*width,'Bottom','Left')
      via(pos_x-4*radius-W+width/2,pos_y-MZI_l-chann_l-2*bb-6*radius-
7/2*width,2*bs_x-radius+width,'Left')
      curve90(pos_x-2*bs_x-3*radius-W-width/2,pos_y-MZI_l-chann_l-2*bb-6*radius-
7/2*width,'Left','Top')
```

```
      via(pos_x-2*bs_x-4*radius-W-width/2,pos_y-MZI_l-chann_l-2*bb-5*radius-
7/2*width,port_d-radius+width/2,'Top')
      curve90(pos_x-2*bs_x-4*radius-W-width/2,pos_y+port_d-MZI_l-chann_l-2*bb-
6*radius-3*width,'Top','Left')
      via(pos_x-2*bs_x-5*radius-W-width/2,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+port_d,7*radius+5*width,'Left')
      IO(pos_x-2*bs_x-12*radius-W-11/2*width,pos_y-MZI_l-chann_l-2*bb-5*radius-
3*width+port_d,width,IO_h,IO_l,'Testing 4')


      return bs_x



    else:
      raise ValueError("The input band_type is not correct. Please type
'Initial', 'Contiguous' or 'Final")


  else:
    raise ValueError(f"The introduced number of channels n={n} is not correct. It
must be equal or higher than 3")




'''
Creates a full WSS (Wavelength Selective Switch) structure with defined numbers
of channels and 3 bands.
Inputs: pos_x = 0 -> Initial position on the x-axis (Smallest position of the 1st
input band CDC)
        pos_y = 0 -> Initial position on the y-axis (Smallest position of the 1st
input band CDC)
        n_channels = 8 -> Number of channels per band.
                          It must be higher than 3
        testing = True -> Indicates whether to include testing ports in the WSS
                          True if there is testing, and false if there is not
Output: position_next -> Position in which to place the next structure
'''
def wss(pos_x = 0, pos_y = 0, n_channels = 8):
  if n_channels >= 3:

    # Connecting the band structures
    b_l = band_struct(pos_x,pos_y,n_channels,'Initial')         # Initial band
structure
    b_m = band_struct(pos_x+b_l,pos_y,n_channels,'Contiguous')  # Middle band
structure
    l_b = band_struct(pos_x+b_l+b_m,pos_y,n_channels,'Final')   # Last band
structure

    position_next = 2*b_l+l_b
    return position_next

  else:
    raise ValueError(f"The introduced number of channels was: {n_channels}\n\
            This number must be equal or higher than 3.")
```

```
channel_struct(0,0,'Initial')    # Creation of an initial channel structure
(Without output CDCs)
D.write_gds('Initial_channel.gds')

D = Device()                            # To restart the structure
channel_struct(0,0,'Final')             # Creation of a normal channel structure
D.write_gds('Contiguous_channel.gds')

D = Device()                               # To restart the structure
band_struct(0,0,n_channels,'Initial')      # Creation of an initial band structure
D.write_gds('Initial_band.gds')

D = Device()                               # To restart the structure
band_struct(0,0,n_channels,'Contiguous')   # Creation of a contiguous band
structure
D.write_gds('Normal_band.gds')

D = Device()                            # To restart the structure
band_struct(0,0,n_channels,'Final')     # Creation of a final band structure
D.write_gds('Final_band.gds')

D = Device()            # To restart the structure
wss(0,0,n_channels)     # Creation of the full WSS chip structure (50um)
D.write_gds('WSS_50um.gds')



'''
Creation of the MZI schematic
'''

len = 5.5  # Length of the connections

D = Device()

via(+W-width/2,-(bb+radius+width/2),len,'Top')
# MZIs and its connections
# MZI 1 (the one on the left)
MZI(+W-width,-(MZI_l+bb+radius+width/2))
# I (MZI interconnection)
curve90(+2*W-1.5*width,-(MZI_l+bb+radius+width/2),'Bottom','Right')
via(+2*W-1.5*width+radius,-(MZI_l+2*radius+bb+width/2),3*radius, 'Right')
curve90(+2*W-3/2*width+bb/2+2*radius,-(MZI_l+bb+2*radius+width/2),'right','top')
# MZI 2 (the one on the right)
MZI(+2*W-2*width+bb/2+3*radius,-(MZI_l+bb+radius+width/2))
# P1 (to output channel CDC 1 of the next structure)
via(+W-width/2,-(MZI_l+bb+radius+width/2),len,'Bottom')
# P2 (to output channel CDC 2)
via(+2*W-3/2*width+bb/2+3*radius,-bb-radius-width/2,len,'Top')
# P3 (to output channel CDC 3)
via(+3*W-5/2*width+bb/2+3*radius,-(bb+radius+width/2),len,'Top')

D.write_gds('MZI_schematic.gds')



"""
Creating the whole Wavelength Selective Switch (WSS) with the real measures
```

```
(Without the last band CDCs (on the right))
Final Schematic
"""

MZI_l = 500      # Length of the MZI (500 um)
chann_l = 900    # Length of the CDC of the channels (900 um)
band_l = 1500    # Length of the CDC of the bands (1.5mm)

IO_h = 22.152        # Height of the I/O ports
IO_l = 35.603        # Horizontal length of the I/O ports

n_channels = 8  # Number of channels per band of the WSS

# Distance between two channel structures (from the input of one to the input of
the next)
ms_x = 8*radius+3*W-3*width+bb/2

port_d = 127  # Distance between two consecutive I/O ports

D = Device()

wss(0,0,n_channels)     # Creation of the full WSS chip structure
D.write_gds('WSS.gds')
```

Appendix C. Introduction to OptoDesigner Script Language

The mask of the chip will be designed using the OptoDesigner Script Language (Spt) from Synopsys. It consists of a PIC design tool that offers inherent support for layout design, covering diverse shapes and angles. This is achieved through completely defined photonic libraries. [59]. This software allows for the creation of PICs by means of scripting and it can also be used along with other software from Synopsys such as OptSim, which serves as a Schematic Capture Tool. It allows to virtually create the prototype as a schematic and exports later code representing the created schematic as a layout to OptoDesigner. In any case, for the purpose of this project, the design of the switch will be done just by using Spt.

Spt is based on scripting, and by using these scripts it is possible to define a structure, perform simulations and generate mask layouts [59]. The language contains a certain resemblance to the C programming language, supporting some object-oriented programming concepts, such as classes, objects, and inheritance. Among its programming features it can be seen that the programming tool allows for the use of variables, arrays, loops and conditionals, functions, functors, classes, results, overriding and non-overriding capabilities, and documentation and comments functionalities.

It also possesses specific computer-aided design (CAD) features, specifically related to circuit fabrication, among which we remark materials, cross-sections, building blocks, elements, connectors, ports and PDKs.

C.1 Programming features of OptoDesigner

Spt contains lot of different programming features that allow for the resolution and creation of the design. Among them, the following will be highlighted due to their usefulness in developing the WSS chip and their differences to the most commonly used programming languages.

C.1.1 Variables

Variables are used to assign an alphanumerical value to a symbolic name. They can be changed over the course of the program and using OptoDesigner it is possible not to define the specific type of variable to be used. Apart from classical programming variable types such as float or int, OptoDesigner implements the use of complex numbers, points with two coordinates referring to a position within the layout and matrices.

C.1.2 Attributes

Attributes are specific elements attached to the variables that store additional information about it. They are similar to the attributes of the classes present in Java, despite their specific syntax being different.

C.1.3 Classes

In Spt, classes can only have one parent class, therefore the inheritance is partial. For these classes, the variables are set to be private.

C.1.4 Loops and conditionals

Loops and conditionals are remarkably similar in syntax to the ones in C. The do, while, switch, for and if cases are present, and there is also a specific operator denoted by "?" which is used to separate the condition and the statements for the use of an if-else conditional.

C.1.5 Functions

Programming functions are similarly used as in other programming languages, but there is no need to define argument types. Spt allows as well for the implementation of default inputs. Functions might be accessible via the *Kamus* dialog box, which is a specific box of OptoDesigner program. Using this dialog box, the user can enter the value of different arguments by using several widgets.

C.1.6 Documentation and Comments

The documentation may be done by making use of fields, which are specific keywords given to functions, classes, and materials. There are different fields which have been predefined in Spt. Among these fields there is one called domain, which can be set to specify the nature of the circuit or sub circuit being designed. Thus, it can be specified whether it should be treated in the optical domain, RF domain or DC domain.

The comments used in OptoDesigner present the same syntax as the ones in C, and they can be multiline or just a single line.

C.1.7 Functors

Functors are user-definable mathematical functions, which allow us to interpret the function at any specific required value. These are also present in functional programming. For the specific case of Spt, the functor $f(x)$ and its derivatives are known. It is important to notice that functors cannot be passed as such to functions, and rather it is necessary to pass a string containing the name of the functor to be used. This is caused due to its static nature, which makes it not possible to overwrite it during the whole script execution. On the other hand, this characteristic allows for it to be evaluated much faster than a function.

C.1.8 Override Control

Functions and layouts can be overridden if they are redefined later in the script. This characteristic may bring harsh consequences, so the keyword *nonoverridable* is given to prevent a function or layout from being overridden. There exists also the keyword *forceoverriding* and overriding. The first one forces the overriding to happen in any case, even if the function or layout was defined as *nonoverridable*. The latter instead checks whether the function, layout, or class that you want to override exists, and throws an error. It is specifically important because if we define the same function on

a subclass and on its parent class, there may be overriding from the subclass function to the parent class.

C.1.9 Result

Results are like variables, but they contain output from simulation and have a specific storage in a Result Database. It is important to notice that results and variables can be used interchangeably while performing mathematical operations.

C.1.10 find command

The *find* command is a specific keyword that can be used to find the minimum, maximum or the root value within a mathematical function $f(x)$. The outcome is stored in a Result Database, but there are no guarantees for the returned value to be the true one.

C.2 Computer-aided design features of OptoDesigner

Spt contains as well specific features for CAD design, from which the following will be highlighted due to their usefulness in order to create the WSS chip.

C.2.1 Materials

These represent the ones used for any component in the layout. They have specific characteristics which module their mechanical, electromagnetic, and optical behavior. They can be added or imported by uploading a PDK. They present inheritance the same way as classes do.

C.2.2 Cross-sections

Cross-sections consist of vertical slices made up of layers of different materials. They can be placed using different geometries, and there is the possibility to modify the overlapping sections in which two distinct materials intersect.

C.2.3 Elements, Connectors and Ports

These are the basic blocks for designing. The elements are geometric figures defined by a mathematical function and some properties. The connectors are a specific type of elements which are used to place connections between the elements. To do so, these connections will be established between ports, which are defined at specific positions. These are also used as the reference for the connections between the elements in OptoDesigner. One important characteristic of ports is that they can be defined either at a relative position with respect to other ports, or as an absolute position within the full schematic. This characteristic obliges to have at least one port defined at an absolute position, so that floating elements are avoided.

C.2.4 Building Blocks

Group of elements packed together than can be later treated as a single element. They work like layout functions that can be inserted in the full layout. They can be given a specific mask cross-section and ports.

For the development of the WSS chip the building blocks will be particularly useful since they allow for the capsularization of the different devices and functioning blocks of the switch.

C.2.5 Process Design Kit

PDKs are libraries which include BBs and information about the technologies of a foundry. For the purpose of this project the Demofab library will be used, and all the different elements will be based upon BBs and materials from it.

Appendix D. OptoDesigner code of single components

D.1 Mach-Zehnder Interferometer Building Block

The code used to create the MZI BB for OptoDesigner is presented below. It should be saved as a file at the same folder as the one implementing the WSS (Appendix E), with the name "*MZI_lib.spt*".

```
/*****************************************************************************
 Library for the creation of MZIs
*****************************************************************************/


//Name of the file to be used as a library: "MZI_lib.spt"


// Select the PDK library that you would like to activate from the PDK file
// function layouts from the DEMOFAB library are used
pda::enableFoundry("DEMOFAB, DEMOFAB.SOI");
#include @layout;

pdkAutoRouter_showOnlyArrows(0);









/*
Creates a single period of a Mach-Zehnder Interferometer, made of an exchange and
phaseshift subcomponents plus 2 S-bend phases to go from one subcomponent to the
other.

Inputs: double ptc_length -> Length of the thermal phase shift section (default =
100um)
        double exchanger_length -> Length of the coupling section (default =
16um)
        double interphase_length -> Horizontal length of the interphase section
(default = 20um)
        double height -> Height of the interphase section (default = 0.5um)
        double width -> Waveguides width (default = 0.5um)
        double gap -> Separation between the closer points of the two waveguides
(default = 0.3um)

** Important Remarks ** All the size measures are expressed in micrometers
*/
layout periodMZI(double ptc_length = 100,
                 double exchanger_length = 16,
                 double interphase_length = 20,
                 double height = 0.5,
                 double width = 0.5,
                 double gap = 0.3,
                 string mk = "mcsSOI_DEEP"
                 )

  Domain_Optics
  AuthorInfo "Alberto Otero Casado"
```

```
{
   // To uncommented only if used alone
   //var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);

   mask::CSselect(mk);

   // Elements of the upper arm
   ml::demofabStraight( : exchanger_length, width) EX_up;
   ml::demofabSbend(in0->EX_up@out0 : interphase_length, height, width) SB2_up;
   ml::demofabStraight(in0->SB2_up@out0 : ptc_length, width) PS_up;
   ml::demofabSbend(in0->PS_up@out0: interphase_length, -height, width) SB1_up;
   ml::demofabStraight(in0->SB1_up@out0 : exchanger_length, width) EX2_up;
   ml::demofabSbend(in0->EX2_up@out0 : interphase_length, height, width) SB3_up;

   // Elements of the lower arm
   ml::demofabStraight(in0->EX_up@in0+[0,-gap-width,0] : exchanger_length, width)
EX_down;
   ml::demofabSbend(in0->EX_down@out0 : interphase_length, -height, width)
SB2_down;
   ml::demofabStraight(in0->SB2_down@out0 : ptc_length, width) PS_down;
   ml::demofabSbend(in0->PS_down@out0 : interphase_length, height, width)
SB1_down;
   ml::demofabStraight(in0->SB1_down@out0 : exchanger_length, width) EX2_down;
   ml::demofabSbend(in0->EX2_down@out0 : interphase_length, -height, width)
SB3_down;

   // Adding ports
   ml::setPort(this : Input->EX_up@in0);
   ml::setPort(this : Through->SB3_up@out0);
   ml::setPort(this : Drop->SB3_down@out0);


   // Persist routing parameters into the MoML file
   this = Tech.populateAutoRouterInformation(this);

   // Set the promoted I/O ports domain
   mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);
   mask::port2layout(&this);
}




/*
Creates the heating section of the MZI, which is placed above the phase control
section.

Inputs: double height -> Height of the electrical section connecting the
electrical ports to the heater
        double length -> Length of the heating section
        double heater_width -> Width of the heater (also of the MZI waveguides)
        double wire_width -> Width of the electrical connections between the
heater and the electrical ports

** Important Remarks ** All the size measures are defined in micrometers
*/
```

```
layout heaterPeriod (double height,
                     double length,
                     double heater_width,
                     double wire_width
                     )

  Domain_DC
  AuthorInfo "Alberto Otero Casado"

{
   mask::CSselect("mcsVia_metal");
   ml::demofabStraight( : height-heater_width, wire_width) DW;      // First
line, coming down closer to the phase control section
   mask::CSselect("mcsHEATER_SOI");
   ml::demofabStraight(in0->DW@out0+[heater_width/2,-wire_width/2,90] : length,
heater_width) HZ;        // Second line, parallel and close to the phase control
section
   mask::CSselect("mcsVia_metal");
   ml::demofabStraight(in0->HZ@out0+[-wire_width/2,heater_width/2,90] : height-
heater_width, wire_width) UP;  // Third line, moving away from the phase control
section


   // Input and output electrical ports
   ml::setPort(this : VI->DW@in0);
   ml::setPort(this : VO->UP@out0);

}




/*
Creates a Mach-Zehnder Interferometer, boxing included (Adding waveguides to
connect to external components)

Inputs: double ptc_length -> Length of the thermal phase shift section (default =
100um)
        double exchanger_length -> Length of the coupling section (default =
16um)
        double interphase_length -> Horizontal length of the interphase section
(default = 20um)
        double height -> Height of the interphase section (default = 0.5um)
        double width -> Waveguides width (default = 0.5um)
        double gap -> Separation between the closer points of the two waveguides
(default = 0.3um)
        double length -> Length of the MZI device (in the box) (default = 500um)
        double box_width -> Width of the box in which the MZI is placed (default
= 6um)
        double w_out -> Width of the external waveguides to which the MZI
connects (default = 0.5um)
        string mk -> Mask to be applied to the components of the MZI (default =
"mcsSOI_DEEP")

** Important Remarks ** All the measure sizes are expressed in micrometers. This
function does not include the thermo-optical heaters.
*/
layout MZI_pck_NoHeat(double ptc_length = 100,
```

```
                        double exchanger_length = 16,
                        double interphase_length = 20,
                        double height = 0.5,
                        double width = 0.5,
                        double gap = 0.3,
                        double length = 500,
                        double box_width = 6,
                        double w_out = 0.5,
                        string mk = "mcsSOI_DEEP"
                        )

  dlgname "MZI with boxing (no phase shift control)"
  Domain_Optics
  AuthorInfo "Alberto Otero Casado"

{

   // To be uncommented if it is used alone
   //var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);

   double length_bend = (length-
(3*interphase_length+2*exchanger_length+ptc_length))/2;
   double h_bend1 = (box_width-gap-w_out-width)/2;         // Height of the first
S-bend (Input Port)
   double h_bend2 = (box_width-gap-w_out-width)/2-height;  // Height of the 2nd
and 3rd S-bends (Through and Drop Ports)

   ml::periodMZI(:
ptc_length,exchanger_length,interphase_length,height,width,gap,mk) P0;

   // Adding the boxing lines (For connections to external waveguides)
   ml::SBend(cout->P0@Input: wlin(w_out,width), length_bend, -h_bend1) S1;
   ml::SBend(cin->P0@Through: wlin(width,w_out), length_bend, h_bend2) S2;
   ml::SBend(cin->P0@Drop: wlin(width,w_out), length_bend, -h_bend2) S3;


   // Adding the ports
   ml::setPort(this : Input->S1@cin);
   ml::setPort(this : Through->S2@cout);
   ml::setPort(this : Drop->S3@cout);

   // Persist routing parameters into the MoML file
   this = Tech.populateAutoRouterInformation(this);

   // Set the promoted I/O ports domain
   mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);
   mask::port2layout(&this);
}




/*
Creates a Mach-Zehnder Interferometer, boxing and thermo-optical heaters included

Inputs: double ptc_length -> Length of the thermal phase shift section (default =
100um)
```

```
        double exchanger_length -> Length of the coupling section (default =
16um)
        double interphase_length -> Horizontal length of the interphase section
(default = 20um)
        double height -> Height of the interphase section (default = 0.5um)
        double width -> Waveguides width (default = 0.5um)
        double gap -> Separation between the closer points of the two waveguides
(default = 0.3um)
        double length -> Length of the MZI device (in the box) (default = 500um)
        double box_width -> Width of the box in which the MZI is placed (default
= 6um)
        double w_out -> Width of the external waveguides to which the MZI
connects (default = 0.5um)
        double w_elec -> Width of the external electrical connections to which
the heater is connected (default = 0.5um)
        string mk -> Mask to be applied to the components of the MZI (default =
"mcsSOI_DEEP")

** Important Remarks ** All the measure sizes are expressed in micrometers.
*/
layout MZI(double ptc_length = 100,
           double exchanger_length = 16,
           double interphase_length = 20,
           double height = 0.5,
           double width = 0.5,
           double gap = 0.3,
           double length = 500,
           double box_width = 6,
           double w_out = 0.5,
           double w_elec = 0.5,
           string mk = "mcsSOI_DEEP")

  dlgname "MZI with boxing and heaters"
  Domain_Optics
  AuthorInfo "Alberto Otero Casado"

{
   // To be uncommented if it is used alone
   //var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);

   double l_bend = (length-
(3*interphase_length+2*exchanger_length+ptc_length))/2;
   double h_bend1 = (box_width-gap-w_out-width)/2;         // Height of the first
S-bend (Input Port)
   double h_bend2 = (box_width-gap-w_out-width)/2-height;  // Height of the 2nd
and 3rd S-bends (Through and Drop Ports)

   ml::periodMZI(:
ptc_length,exchanger_length,interphase_length,height,width,gap,mk) P0;

   // Adding the boxing waveguides
   ml::SBend(cout->P0@Input: wlin(w_out,width), l_bend, -h_bend1) S1;
   ml::SBend(cin->P0@Through: wlin(width,w_out), l_bend, h_bend2) S2;
   ml::SBend(cin->P0@Drop: wlin(width,w_out), l_bend, -h_bend2) S3;

   // Adding the heaters
   ml::heaterPeriod(VI-
>S1@cin+[l_bend+exchanger_length+interphase_length+w_elec/2,w_out/2,-90] :
```

```
                    h_bend1+width/2+w_out/2-height , ptc_length,width,w_elec) H;

   // Adding the optical ports
   ml::setPort(this : Input->S1@cin);
   ml::setPort(this : Through->S2@cout);
   ml::setPort(this : Drop->S3@cout);
   // Adding the electrical ports
   ml::setPort(this: VI->H@VI);
   ml::setPort(this: VO->H@VO);

   // Persist routing parameters into the MoML file
   this = Tech.populateAutoRouterInformation(this);

   // Set the promoted I/O ports domain
   mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);
   mask::port2layout(&this);
}




/*****************************************************************************
// Trials for the Full MZI

//**Important Remark ** The sentence at the beginning of the function layouts
(var RoutingGrid) must be uncommented to implement these trials
*****************************************************************************/

/*
// All the specified lengths are in um
double ptc_length = 30;            // Phase Thermal Control length
double exchanger_length = 5;       // Exchanger length
double interphase_length = 8;      // Length of the interphase between the
exchange and the PTC (in the same axis of these 2)
double height = 0.5;               // Height of the interphases
double width = 1;                  // Width of the lines
double gap = 0.3;                  // Gap between the 2 exchangers
double length = 100;               // Length in which the MZI must be done
double b_w = 6;                    // Width of the MZI after boxing
double w_elec = 5;                 // Width of the electrical connections
double w_out = 0.5;                // Width of the waveguides outside the MZI
*/

// 1st trial (withouth boxing)
//ml::periodMZI(Input->[0] :
ptc_length,exchanger_length,interphase_length,height,width,gap) P0;

// 2nd trial (boxing included but no heaters)
//ml::MZI_pck_NoHeat(Input->[0,-w_out/2,0]:
ptc_length,exchanger_length,interphase_length,height,width,gap,length,b_w,w_out)
MNH;

// Third trial (boxing and heaters included)
// ml::MZI(Input->[0,-w_out/2,0]:
ptc_length,exchanger_length,interphase_length,height,width,gap,length,b_w,w_out,w
_elec) MZ;
```

D.2 Contra-Directional Coupler Building Block

The code used to create the CDC BB for OptoDesigner is presented below. It should be saved as a file at the same folder as the one implementing the WSS (Appendix E), with the name "*CDC_lib.spt*".

```
/*****************************************************************************
 Library for the creation of CDCs
*****************************************************************************/


//Name of the file to be used as a library: "CDC_lib.spt"


// Select the PDK library that you would like to activate from the PDK file
// function layouts from the DEMOFAB library are used
pda::enableFoundry("DEMOFAB, DEMOFAB.SOI");
#include @layout;

pdkAutoRouter_showOnlyArrows(0);









/*
Creates a one period Bragg Phaseshift, made by 2 subelements having the tooth and
the width (upper and lower)"

Inputs: string f -> Name of the functor (function) to be applied to the width of
the large-width subelement. (default=functor1)
        double width -> Width of the elements (default=450nm)
        double delta_width -> Width of the teeth of the bragg cell (times the
functor f) (default=900nm)
        double length_1 -> Length of the first tooth-width subelement
(default=121nm)
        double length_2 -> Length of the second tooth-width subelement
(default=121nm)
        double starting_point -> Starting position to evaluate the function f for
the large-width subelement. It will be evaluated at
starting_point+length_short+(length_large/2) (default=0)
        int UpOrDown -> Boolean stating if the first segment goes up or down. If
it equals 0 it will go down, while any other value means it goes up (default = 0
-> Down)
        string mk -> Mask to be applied to the element (default = "mcsSOI_DEEP")

**Important Remarks** The functor must be defined outside of the layout and its
name (string variable) is passed as the first input. If not specified, the width
of the large subelement would be the one specified at bragg_width. Also, the name
functor1 would better not be used to define the functor (equals 1 in all cases).
All the size measures are expressed in micrometers
*/
layout braggPeriod(string f = "functor1",
                   double width = 0.5,
                   double delta_width = 0.1,
                   double length_1 = 0.121,
                   double length_2 = 0.121,
```

```
                    double starting_point = 0,
                    int UpOrDown = 0,
                    string mk = "mcsSOI_DEEP"
                    )

  Domain_Optics
  AuthorInfo "Alberto Otero Casado"

{
  double fx = sys::callFunctor(f, starting_point+length_1+(length_2/2));
  double ws;   // Shift to the width in the vertical position (with respect to
the previous element)

  if (UpOrDown == 0){
      ws = -delta_width*fx;
  }
  else{
      ws = delta_width*fx;
  }
  mask::setLayoutPort(this, "in0", "out0");
  double w = width+delta_width;

  // Problems with the layout using the following command
  //var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);

  // Subelements of the period
  mask::CSselect(mk);
  ml::demofabStraight(: length_1,w) Input;
  ml::pxDomainTransition(in0->Input@out0:) Matcher;                          // To
avoid width mismatch error
  ml::demofabStraight(in0-> Matcher@out0+[0,-ws,0]: length_2, w) Output;
  ml::pxDomainTransition(in0->Output@out0:) Matcher_Output;                  // To
avoid width mismatch error

  // Defining the ports
  ml::setPort(this : in0->Input@in0);
  ml::setPort(this : out0->Matcher_Output@out0);

  // Persist routing parameters into the MoML file
  this = Tech.populateAutoRouterInformation(this);

  // Set the promoted I/O ports domain
  mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);

  mask::port2layout(&this);
 }




/*
Create a Periodic Bragg Phaseshift, made by alternating upper and lower tooth-
width subelements.

Inputs: string f -> Name of the functor (function) to be applied to the width of
the large-width subelements. (default=functor1)
        double width -> Width of the phaseshift (default=500nm)
```

```
        double delta_width -> Width of the teeth of the phaseshift (to be
multiplied by a function f) (default=100nm)
        double period -> (Initial Pitch) Length of the period composed by the
first and second subelements (default=242nm)
        double duty_cycle -> Duty-cycle for all the periods, (percentage of
length occupied by the second subelement over the full period) (default=50%)
        double length -> Length of the whole phaseshift (default=24.2um)
        int UpOrDown -> Boolean stating if the first segment goes up or down. If
0 it will go down, else it goes up (default = 0 -> Down)
        double chirp -> Periodical difference between the first and the last
pitchs. (default=0)
        string mk -> Mask to be applied to the element (default = "mcsSOI_DEEP")

**Important Remarks** The functor must be defined outside of the layout and its
name (string variable) is passed as the first input. . Also, the name functor1
would better not be used to define the functor (equals 1 in all cases). If a one-
period Bragg phaseshift is to be used, the layout braggPeriod may be preferably
used. All the size measures are expressed in micrometers
*/
layout braggPhaseshift(string f = "functor1",
                  double width = 0.5,
                  double delta_width = 0.1,
                  double period = 0.242,
                  double duty_cycle = 50,
                  double length = 24.2,
                  int UpOrDown = 0,
                  double chirp = 0,
                  string mk = "mcsSOI_DEEP"
                  )
  Domain_Optics
  AuthorInfo "Alberto Otero Casado"

{
  mask::setLayoutPort(this, "in0", "out0");

  // To be used if the element is placed alone
  //var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);


  double ws;   // Shift to the tooth subelement in the vertical position (with
respect to the previous subelement)
  double ps = period*(100-duty_cycle)/100;   // Length of the small subelement
(in the period)
  double pl = period*duty_cycle/100;         // Length of the large subelement
(in the period)
  double p1;                                 // Changed length of the first
subelement
  double p2;                                 // Changed length of the 2nd
subelement
  double rest;                               // Difference from the length to the
end of the device

  // Calculate how many cycles are necessary
  int cycles = 1;
  int j;
  double length_now = cycles*period;
  double length_variable;
  double length_prev;
```

```
  // Calculating number of cycles, length of the device and remaining to the
specified length
  while(length_now <= length){
      cycles = cycles+1;
      length_prev = length_now;
      length_variable = chirp*cycles/2;     // Equal to summing all the variable
lengths
      length_now = cycles*period+length_variable;
  }

   cycles = cycles-1;
   rest = length-length_prev;

  // To check how to start the teeth positioning
  double fx = sys::callFunctor(f,(period*cycles)+ps+(pl/2));
  if (UpOrDown == 0){
      ws = -delta_width*fx;
  }
  else{
      ws = delta_width*fx;
  }


  // Creating the whole bragg periodic phaseshift
  var phase_element[cycles+2];
  // First bragg_period (has absolute position)
  phase_element[0] = ml::braggPeriod( : f,width,delta_width,ps,pl,0,UpOrDown,mk);
  //All the bragg_period elements for all the full periods
  for (int i=1; i<cycles; i++){
         p1 = ps+i*chirp/(cycles-1)/2;
         p2 = pl+i*chirp/(cycles-1)/2;
         //p2 = pl*(1+(i+1)*chirp/period/cycles);
         phase_element[i] = ml::braggPeriod(in0-> phase_element[i-
1]@out0+[0,ws]:f,width,delta_width,p1,p2,i*period,UpOrDown,mk);
  }
  phase_element[cycles] = ml::pxDomainTransition(in0->phase_element[cycles-
1]@out0:);
  phase_element[cycles+1] = ml::demofabStraight(in0-> phase_element[cycles] @
out0 + [0,ws/2,0]: rest,width);
  // Defining the ports
  ml::setPort(this : in0->phase_element[0]@in0);           // First bragg_period
element
  ml::setPort(this : out0->phase_element[cycles+1]@out0);  // Last bragg_period
element

  // Persist routing parameters into the MoML file
  this = Tech.populateAutoRouterInformation(this);

  // Set the promoted I/O ports domain
  mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);

  mask::port2layout(&this);
}
```

```
/*
Create a Periodic CDC, made by two close corrugated waveguides. Prior to boxing,
only containg the two Bragg grating waveguides.

Inputs: string f -> Name of the functor (function) to be applied to the width of
the elements (default=functor1)
        double width_up -> Width of the teeth of the upper phaseshift
(default=600nm)
        double width_down -> Width of the teeth of the lower phaseshift
(default=400nm)
        double shift_up -> Shift made to the teeth with respect to the previous
one in the upper phaseshift (default=20nm)
        double shift_down -> Shift made to the teeth with respect to the previous
one in the lower phaseshift (default=40nm)
        double period -> (Initial pitch length) Length of the period composed by
the 2 subelements (default=242nm)
        double duty_cycle -> Duty-cycle for all the periods, (percentage of
length occupied by the second subelement over the full period) (default=50%)
        double length -> Maximum length of the whole phaseshift (default=24.2um).
Will use the number of cycles to be the closer without surpassing it. The
difference between the length and the rest will be occupied by flat waveguides in
the center whose width equals to the ones of the corrugated waveguides without
the teeth
        double chirp -> Periodical difference between the first and the last
pitchs (default=0)
        double gap -> determines the gap between the closest points of the two
phaseshifts (default=300nm)
        string mk -> Mask to be applied to the element (default = "mcsSOI_DEEP")

**Important Remarks** The functor must be defined outside of the layout and its
name (string variable) is passed as the first input. For optimal functioning, the
CDC should have a length at least bigger than one full period. Also, the name
functor1 would better not be used to define the functor (equals 1 in all cases).
All the size measures are expressed in micrometers
*/
layout CDC_prebox(string f = "functor1",
           double width_up = 0.6,
           double width_down = 0.4,
           double teeth_up = 0.02,
           double teeth_down = 0.04,
           double period = 0.242,
           double duty_cycle = 50,
           double length = 24.2,
           double chirp = 0,
           double gap = 0.3,
           string mk = "mcsSOI_DEEP"
           )

  dlgname "Contra directional Coupler (prior to boxing)"
  Domain_Optics
  AuthorInfo "Alberto Otero Casado"

{
   //mask::setLayoutPort(this, "in0", "out0");
   //var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);

   double ps = period*(100-duty_cycle)/100;   // Length of the small subelement
(in the period)
```

```
   double pl = period*duty_cycle/100;          // Length of the large subelement
(in the period)
   int cycles = length/period;                 // Number of cycles of the Periodic
Bragg phaseshift (down-rounded)
   double rest = length - (cycles*period);     // The remainder of length after
completing all full periods
   double fx = sys::callFunctor(f,(period*cycles)+ps+(pl/2));
   double shift_up = teeth_up*fx;
   double shift_down = teeth_down*fx;

   ml::braggPhaseshift(: "functor1", width_up, teeth_up, period, duty_cycle,
length, 1, chirp,mk) BPS_up;
   ml::braggPhaseshift(in0-> BPS_up@in0 + [0,-gap-shift_up-shift_down-
(width_up+teeth_up)/2-(width_down+teeth_down)/2,0] :
                       "functor1", width_down, teeth_down, period, duty_cycle,
length, 0, chirp,mk) BPS_down;

   // Defining the ports
   ml::setPort(this : Input->BPS_up@in0);
   ml::setPort(this : Through->BPS_up@out0);
   ml::setPort(this : Drop->BPS_down@in0);
   ml::setPort(this : Add->BPS_down@out0);

   this = Tech.populateAutoRouterInformation(this);
   // Set the promoted I/O ports domain
   mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);
   mask::port2layout(&this);
}




/* Create a Periodic CDC, made by alternating short-width and large-width
subelements. Uses boxing to connect it to other elements.
If the length is a multiple of the period, the CDC is created with equal length
subelements. Otherwise the last one may be higher, containing also the rest.

Inputs: string f -> Name of the functor (function) to be applied to the width of
the large-width subelements.
        double width_up -> Width of the upper corrugated waveguide
(default=600nm)
        double width_down -> Width of the lower corrugated waveguide
(default=400nm)
        double teeth_up -> Width of the teeth of the upper corrugated waveguide
(default=20nm)
        double teeth_down -> Width of the teeth of the lower corrugated waveguide
(default=40nm)
        double period -> Initial pitch length,  composed by the 2 subelements
(default=242nm)
        double duty_cycle -> Duty-cycle of the periods (percentage of length
occupied by the second subelement over the full period) (default=50%)
        double length -> Maximum length of the whole bragg grating
(default=24.2um). Will use the number of cycles to be the closer without
surpassing it. The difference between the length and the rest will be occupied by
flat waveguides in on the right side whose width equals to the one of the
waveguides without the teeth.
        double chirp -> Periodical difference between the first and the last
pitchs (default=0)
```

```
        double gap -> determines the gap between the closest points of the two
corrugated waveguides (default=300nm)
        double box_width -> Determines the width of the box in which the CDC is
contained (default=6um)
        double l_bend -> Length of the S-bends that connects the CDC to external
waveguides (default=5um)
        doueble w_out -> Width of the external waveguides connecting to the CDC
        string mk -> Specifies the mask to be applied to the elements of the
layout (default = "mcsSOI_DEEP")

**Important Remarks** The functor must be defined outside of the layout and its
name (string variable) is passed as the first input. For optimal functioning, the
CDC should have a length at least bigger than one full period. Also, the name
functor1 would better not be used to define the functor (equals 1 in all cases).
All the size measures are defined in micrometers.
*/
layout CDC(string f = "functor1",
           double width_up = 0.6,
           double width_down = 0.4,
           double teeth_up = 0.02,
           double teeth_down = 0.04,
           double period = 0.242,
           double duty_cycle = 50,
           double length = 24.2,
           double chirp = 0,
           double gap = 0.3,
           double box_width = 6,
           double l_bend = 5,
           double w_out = 0.5,
           string mk = "mcsSOI_DEEP"
           )

  dlgname "Contra directional Coupler (CDC) with boxing"
  Domain_Optics
  AuthorInfo "Alberto Otero Casado"

{
   // Following two lines only to be used if the function is used alone
   //mask::setLayoutPort(this, "in0", "out0");
   //var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);

   double ps = period*(100-duty_cycle)/100;   // Length of the small subelement
(in the period)
   double pl = period*duty_cycle/100;         // Length of the large subelement
(in the period)
   int cycles = length/period;                // Number of cycles of the Periodic
Bragg phaseshift (down-rounded)
   double fx = sys::callFunctor(f,(period*cycles)+ps+(pl/2));
   double shift_up = teeth_up*fx;
   double shift_down = teeth_down*fx;
   double h_bend = (box_width-teeth_up-teeth_down-width_up-width_down-gap)/2;  //
Height of the S-bend


   // 1st waveguide
   mask::CSselect(mk);
   ml::SBend(: wlin(w_out,width_up), l_bend, -h_bend) S1;
```

```
   ml::braggPhaseshift(in0->S1@cout+[0,teeth_up/2,0]:
"functor1",width_up,teeth_up,period,duty_cycle,length,1,chirp,mk) BPS_up;
   ml::SBend(cin->BPS_up@out0: wlin(width_up,w_out), l_bend, h_bend) S2;

   // 2nd waveguide
   ml::braggPhaseshift(in0-> BPS_up@in0 + [0,-gap-shift_up-shift_down-
(width_up+teeth_up)/2-(width_down+teeth_down)/2,0] :

"functor1",width_down,teeth_down,period,duty_cycle,length,0,chirp,mk) BPS_down;
   ml::SBend(cout->BPS_down@in0+[0,teeth_down/2,0]: wlin(w_out,width_down),
l_bend, h_bend) S3;
   ml::SBend(cin->BPS_down@out0: wlin(width_down,w_out), l_bend, -h_bend) S4;


   // Defining the ports
   ml::setPort(this : Input->S1@cin);
   ml::setPort(this : Through->S2@cout);
   ml::setPort(this : Drop->S3@cin);
   ml::setPort(this : Add->S4@cout);

   this = Tech.populateAutoRouterInformation(this);

   // Set the promoted I/O ports domain
   mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);
   mask::port2layout(&this);
}







/*****************************************************************************
Trials for the CDC

**Important Remark ** The sentence at the beginning of the function layouts (var
RoutingGrid) must be uncommented to implement these trials
*****************************************************************************/

/*
double w_up = 0.57;                // Width of the upper phaseshift
double w_down = 0.430;             // Width of the lower phaseshift
double teeth_up = 0.1;             // Width of the teeth of the upper phaseshift
double teeth_down = 0.06;          // Width of the teeth of the lower phaseshift
double period = 0.275;             // Length of the period of the Bragg
Phaseshift
double duty_cycle = 50;            // Duty-cycle = 50%
double L_tot = 15;                 // Length of the full CDC
double gap = 0.05;                 // Gap between the closest points of the CDC
double chirp = 0.1;                // Chirp of the CDC
double b_w = 6;                    // Box width of the CDC
double l_bend = 5;                 // Length of the S-bend (in the same axis as
the bragg grating waveguides)
double w_out = 0.5;                // Width of the wvg outside the CDC

functor functor1 1;                // F(x) = 1 --> So that the large subelements
have always the same width (1st trial)
*/
```

```
// First trial example (Full CDC)
//ml::CDC( Input->[0] :
"functor1",w_up,w_down,teeth_up,teeth_down,period,duty_cycle,L_tot,chirp,gap,b_w,
l_bend,w_out) CDC1;

// Second trial example (CDC without boxing)
//ml::CDC_prebox( Input->[0] :
"functor1",w_up,w_down,teeth_up,teeth_down,period,duty_cycle,L_tot,chirp,gap)
CDC2;
```

D.3 Pseudo-MZI Building Block

The code used to create the pseudo-MZI BB for OptoDesigner is presented below. It should be saved as a file at the same folder as the one implementing the WSS (Appendix E), with the name "*MZI_lib.spt*":

```
/*****************************************************************************
 Library for the creation of Pseudo-MZIs
*****************************************************************************/


//Name of the file to be used as a library: "MZI_lib.spt"


// Select the PDK library that you would like to activate from the PDK file
// function layouts from the DEMOFAB library are used
pda::enableFoundry("DEMOFAB, DEMOFAB.SOI");
#include @layout;

pdkAutoRouter_showOnlyArrows(0);







/*
Function layout to mirror the MZI BB

Inputs: double ptc_length -> Does not change the behaviour of the block
        double exchanger_length -> Does not change the behaviour of the block
        double interphase_length -> Does not change the behaviour of the block
        double height -> Does not change the behaviour of the block
        double width -> Does not change the behaviour of the block
        double gap -> Does not change the behaviour of the block
        double length -> Length of the box in which the pseudo-MZI is contained
        double box_width -> Width of the box
        double w_out -> Width of the external waveguides to which the pseudo-MZI
connects
        double w_elec -> Width of the external electrical connections to which
the heater is connected
        string mk -> Mask to be applied to the components of the MZI

** Important Remarks ** The parameters of this function layout are the same as
the ones given to the real MZI blocks (heating and boxing included), although in
this case only length, box_width, w_out, w_elec and mk will be really used. These
```

```
define the measures of the introduced fake block. All the measure sizes are
expressed in micrometers.
*/
layout MZI(double ptc_length,
           double exchanger_length,
           double interphase_length,
           double height,
           double width,
           double gap,
           double length,
           double box_width,
           double w_out,
           double w_elec,
           string mk
           )

  AuthorInfo "Alberto Otero Casado"

{
// To be uncommented if the block is used alone
//var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);

double l_bend = (length-(3*interphase_length+2*exchanger_length+ptc_length))/2;
// Length of the bend

mask::CSselect(mk);
ml::demofabStraight(: length, box_width) PCK;   // Box
ml::demofabStraight(in0->PCK@in0+[0,(box_width-w_out)/2,0] : length, w_out) C1;
ml::Straight(cin->C1@in0 : wlin(w_out,w_out/4), length/2) Ini ;
ml::demofabStraight(out0->PCK@out0+[0,(w_out-box_width)/2,0] : length/2, w_out)
C2;

// Heater input port (VI)
mask::CSselect("mcsVia_metal");
ml::demofabStraight(in0-
>PCK@in0+[w_elec/2+l_bend+exchanger_length+interphase_length,box_width/2,-90] :
w_out,w_elec) H0;
// Heater output port (VO)
ml::demofabStraight(in0->PCK@in0+[length-l_bend-2*interphase_length-
exchanger_length-w_elec/2,box_width/2,-90] : w_out,w_elec) Hn;

// Setting the ports
ml::setPort(this : Input->C1@in0);
ml::setPort(this : Through->C1@out0);
ml::setPort(this : Drop->C2@out0);
// Electrical ports
ml::setPort(this : VI->H0@in0);
ml::setPort(this : VO->Hn@in0+[0,0,180]);

this = Tech.populateAutoRouterInformation(this);
// Set the promoted I/O ports domain
mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);
mask::port2layout(&this);

}
```

```
/*************************************************************************
Trial to use the pseudo-MZI and see the electrical port connections

**Important Remark ** The sentence at the beginning of the function layout (var
RoutingGrid) must be uncommented to implement these trials
*************************************************************************/


/*
// To be uncommented if the trials are to be implemented
double ptc_length = 30;            // Phase Thermal Control length
double exchanger_length = 5;       // Exchanger length
double interphase_length = 8;      // Length of the interphase between the
exchange and the PTC (in the same axis of these 2)
double height = 0.5;               // Height of the interphases
double wi = 1;                     // Width of the lines
double gap = 0.3;                  // Gap between the 2 exchangers
double length = 100;               // Length in which the MZI must be done
double b_w = 6;                    // Width of the MZI after boxing
double w_elec = 5;                 // Width of the electrical connections
double w_out = 0.5;                // Width of the waveguides outside the MZI
string mask_MZI = "mcsSOI_Mod1";   // Mask to be used for the pseudo-MZI

// Trial: Implementation of the pseudo-MZI
ml::MZI(Input->[0] : ptc_length, exchanger_length, interphase_length, height, wi,
gap, length, b_w, w_out, w_elec, mask_MZI) MZ;
*/
```

D.4 Pseudo-CDC Building Block

The code used to create the pseudo-CDC BB for OptoDesigner is presented below. It should be saved as a file at the same folder as the one implementing the WSS (Appendix E), with the name "*CDC_lib.spt*":

```
/*************************************************************************
 Library for the creation of Pseudo-CDCs
*************************************************************************/


//Name of the file to be used as a library: "CDC_lib.spt"


// Select the PDK library that you would like to activate from the PDK file
// function layouts from the DEMOFAB library are used
pda::enableFoundry("DEMOFAB, DEMOFAB.SOI");
#include @layout;

pdkAutoRouter_showOnlyArrows(0);
```

```
/*
Function layout to mirror the CDC BB

Inputs: string f -> Does not change the behaviour of the block
        double width_up -> Width of the upper waveguide
        double width_down -> Width of the lower waveguide
        double teeth_up -> Does not change the behaviour of the block
        double teeth_down -> Does not change the behaviour of the block
        double period -> Does not change the behaviour of the block
        double duty_cycle -> Does not change the behaviour of the block
        double length -> Length to be added to the block (corresponds to the
length of the Bragg grating in the real CDC)
        double chirp -> Does not change the behaviour of the block
        double gap -> Does not change the behaviour of the block
        double box_width -> Determines the width of the box in which the CDC is
contained
        double l_bend -> Length to be added to the block (x2) (Corresponds to the
S-bends in the real CDC)
        doueble w_out -> Width of the external waveguides connecting to the CDC
        string mk -> Specifies the mask to be applied to the elements of the
layout

**Important Remarks** The parameters of this function layout are the same as the
ones given to the real CDC blocks (boxing included), although in this case only
width_up, width_down, length, l_bend, box_width and mk will be really used. These
define measures used for the fake block. All the measure sizes are expressed in
micrometers.
*/
layout CDC(string f,
           double w_up,
           double w_down,
           double teeth_up,
           double teeth_down,
           double period,
           double duty_cycle,
           double length,
           double chirp,
           double gap,
           double b_w,
           double l_bend,
           double w_out,
           string mk
           )

  AuthorInfo "Alberto Otero Casado"

{
// To be uncommented only if the block is used alone
//var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);

mask::CSselect(mk);

ml::demofabStraight(: length+2*l_bend, b_w) PCK;   // Box
ml::demofabStraight(in0->PCK@in0+[0,(b_w-w_out)/2,0] : length+2*l_bend, w_out)
C1;
ml::Straight(cin->C1@in0 : wlin(w_out,w_out/4), length/2) Ini ;
```

```
ml::demofabStraight(in0->PCK@in0+[0,(w_out-b_w)/2,0] : length+2*l_bend, w_out)
C2;

// Setting the ports
ml::setPort(this : Input->C1@in0);
ml::setPort(this : Through->C1@out0);
ml::setPort(this : Drop->C2@in0);
ml::setPort(this : Add->C2@out0);

this = Tech.populateAutoRouterInformation(this);
// Set the promoted I/O ports domain
mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);
mask::port2layout(&this);

}




/*****************************************************************************
Pseudo-CDC trial section

**Important Remark ** The sentence at the beginning of the function layouts (var
RoutingGrid) must be uncommented to implement these trials
*****************************************************************************/


/*
double w_up = 0.57;                // Width of the upper phaseshift
double w_down = 0.430;             // Width of the lower phaseshift
double teeth_up = 0.1;             // Width of the teeth of the upper phaseshift
double teeth_down = 0.06;          // Width of the teeth of the lower phaseshift
double period = 0.275;             // Length of the period of the Bragg
Phaseshift
double duty_cycle = 50;            // Duty-cycle = 50%
double L_tot = 15;                 // Length of the full CDC
double gap = 0.05;                 // Gap between the closest points of the CDC
double chirp = 0.1;                // Chirp of the CDC
double b_w = 6;                    // Box width of the CDC
double l_bend = 5;                 // Length of the S-bend (in the same axis as
the bragg grating waveguides)
double w_out = 0.5;                // Width of the wvg outside the CDC
string mk_CDC = "mcsSOI_Mod1";     // Mask cross-section to be used for the
pseudo-CDC

functor functor1 1;                // F(x) = 1 --> So that the large subelements
have always the same width (1st trial)

// Trial: Implementation of the pseudo-CDC
ml::CDC( Input->[0] :
"functor1",w_up,w_down,teeth_up,teeth_down,period,duty_cycle,L_tot,chirp,gap,b_w,
l_bend,w_out,mk_CDC) CDC1;
*/
```

Appendix E. OptoDesigner code of the full WSS design

The code used to develop the full chip in OptoDesigner is presented below. Along with it, two files should be saved in the same folder to implement the CDCs and MZIs, with the names "*CDC_lib.spt*" and "*MZI_lib.spt*" respectively. The code of these is presented in Appendix D.

```
/*******************************************************************************
Implementation of the full WSS chip
*******************************************************************************/


// Select the PDK library that you would like to activate from the PDK file
// function layouts from the DEMOFAB library are used
pda::enableFoundry("DEMOFAB, DEMOFAB.SOI");

#include @layout;
#include CDC_lib.spt;     // Library containing the Contra-directional Couplers
#include MZI_lib.spt;     // Library containing the Mach-Zehnder Interferometers

pdkAutoRouter_showOnlyArrows(0);

// Lengths of the components
double MZIS_l = 500;      // Length of the MZI in the S-band (500 um)
double MZIC_l = 500;      // Length of the MZI in the C-band (500 um)
double MZIL_l = 500;      // Length of the MZI in the L-band (500 um)
double chan_l = 940;      // Length of the CDC of the channels (940 um)
double bandS_l = 1500;    // Length of the CDC of the S-band (1.5mm)
double bandC_l = 800;     // Length of the CDC of the C-band (0.8mm)
double bandL_l = 1400;    // Length of the CDC of the L-band (1.4mm)

// Chirps and periods of the CDC components
double chirpS = 0.02;     // Chirp of the S-band
double chirpC = 0.009;    // Chirp of the C-band
double chirpL = 0.018;    // Chirp of the L-band
double pitchS = 0.275;    // Pitch of the S-Band (band CDC)
double pitchC = 0.293;    // Pitch of the C-Band (band CDC)
double pitchL = 0.302;    // Pitch of the L-Band (band CDC)
double pitchS1 = 0.2842; // Pitch of the 1st channel of the S-band
double pitchS2 = 0.284;  // Pitch of the 2nd channel of the S-band
double pitchS3 = 0.2838; // Pitch of the 3rd channel of the S-band
double pitchS4 = 0.2836; // Pitch of the 4th channel of the S-band
double pitchS5 = 0.2834; // Pitch of the 5th channel of the S-band
double pitchS6 = 0.2832; // Pitch of the 6th channel of the S-band
double pitchS7 = 0.283;  // Pitch of the 7th channel of the S-band
double pitchS8 = 0.2828; // Pitch of the 8th channel of the S-band
double pitchC1 = 0.2989; // Pitch of the 1st channel of the C-band
double pitchC2 = 0.2987; // Pitch of the 2nd channel of the C-band
double pitchC3 = 0.2985; // Pitch of the 3rd channel of the C-band
double pitchC4 = 0.2982; // Pitch of the 4th channel of the C-band
double pitchC5 = 0.298;  // Pitch of the 5th channel of the C-band
double pitchC6 = 0.2978; // Pitch of the 6th channel of the C-band
double pitchC7 = 0.2976; // Pitch of the 7th channel of the C-band
double pitchC8 = 0.2973; // Pitch of the 8th channel of the C-band
double pitchL1 = 0.3126; // Pitch of the 1st channel of the L-band
double pitchL2 = 0.3124; // Pitch of the 2nd channel of the L-band
double pitchL3 = 0.3121; // Pitch of the 3rd channel of the L-band
```

```
double pitchL4 = 0.3119; // Pitch of the 4th channel of the L-band
double pitchL5 = 0.3116; // Pitch of the 5th channel of the L-band
double pitchL6 = 0.3114; // Pitch of the 6th channel of the L-band
double pitchL7 = 0.311;  // Pitch of the 7th channel of the L-band
double pitchL8 = 0.3109; // Pitch of the 8th channel of the L-band

// Width, corrugations and gaps of the CDCs
double band_W1 = 0.57;   // Width of the 1st waveguide of a band CDC
double band_W2 = 0.43;   // Width of the 2nd waveguide of a band CDC
double band_corr1 = 0.1; // Width of the corrugations of the 1st waveguide of a
band CDC
double band_corr2 = 0.06;// Width of the corrugations of the 2nd waveguide of a
band CDC
double band_G = 0.1;     // Gap between the two waveguides of a band CDC
double chan_W1 = 0.57;   // Width of the 1st waveguide of a channel CDC
double chan_W2 = 0.43;   // Width of the 2nd waveguide of a channel CDC
double chan_corr1 = 0.1; // Width of the corrugations of the 1st waveguide of a
band CDC
double chan_corr2 = 0.06;// Width of the corrugations of the 2nd waveguide of a
band CDC
double chan_G = 0.1;     // Gap between the two waveguides of a band CDC

// Parameters of the MZIs
double ex_l = 16;       // Length of the exchanger section (16um)
double ptc_l = 100;     // Length of the Phase Thermal Control section (100um)
double inter_l = 20;    // Length of the Interphase section (20um)
double MZI_G = 0.3;     // Gap between the two waveguides in the exchanger
section (0.3um)
double MZI_w = 0.5;     // Width of the waveguides of the MZI (0.5um)
double MZI_h = 0.5;     // Height between the Phase Thermal Control and Exchange
sections (0.5um)



// Different masks to be used if we want to use the real MCS
// Either this or the following definition must be uncommented
string mask_MZI_S = "mcsSOI_DEEP";      // Mask to represent the S-band MZIs
string mask_MZI_C = "mcsSOI_DEEP";      // Mask to represent the C-band MZIs
string mask_MZI_L = "mcsSOI_DEEP";      // Mask to represent the L-band MZIs
string mask_chanS = "mcsSOI_DEEP";      // Mask to represent the S-band channel
CDCs
string mask_chanC = "mcsSOI_DEEP";      // Mask to represent the C-band channel
CDCs
string mask_chanL = "mcsSOI_DEEP";      // Mask to represent the L-band channel
CDCs
string mask_CDC_S = "mcsSOI_DEEP";      // Mask to represent the S-band CDCs
(band CDC)
string mask_CDC_C = "mcsSOI_DEEP";      // Mask to represent the C-band CDCs
(band CDC)
string mask_CDC_L = "mcsSOI_DEEP";      // Mask to represent the L-band CDCs
(band CDC)


/*
// Different masks to be used if we want to distinguish between the componentes
// Either this or the previous definition must be uncommented
string mask_MZI_S = "mcsBB_RF";         // Mask to represent the S-band MZIs
string mask_MZI_C = "mcsBB_DC";         // Mask to represent the C-band MZIs
string mask_MZI_L = "mcsDie_Demo";      // Mask to represent the L-band MZIs
```

```
string mask_chanS = "mcsBB_TAPER";      // Mask to represent the S-band channel
CDCs
string mask_chanC = "mcsPCM_align";     // Mask to represent the C-band channel
CDCs
string mask_chanL = "mcsPackageDC";     // Mask to represent the L-band channel
CDCs
string mask_CDC_S = "mcsDiceline";      // Mask to represent the S-band CDCs
(band CDC)
string mask_CDC_C = "mcsSOI_RIB";       // Mask to represent the C-band CDCs
(band CDC)
string mask_CDC_L = "mcsSOI_Mod1";      // Mask to represent the L-band CDCs
(band CDC)
*/

// Variables for the full design
double radius = 5;   // Radius for the bendings
double W = 6;        // Width of the CDC and MZI boxing
double width = 0.5;  // Width of the waveguides
double bb = 20;      // Distance of the bounding box of a crossing (nothing can
be placed)
functor functor1 1;  // F(x) = 1
double dt = 50;      // 50% Duty-cycle
double port_d = 127; // Distance between two consecutive ports (127um) (Photonics
standard)
double ms_x = 8*radius+3*W-3*width+bb/2;        // Distance to the position of
the next channel block (Input CDC to Input CDC)
double bs_x = 8*ms_x+19*radius+11*width+4*W;    // Distance to the position of
the next band block
functor functor1 1;        // Constant Function for the Bragg elements
int n_mzi = 48;            // Number of MZIs, and pads to apply electrical
voltage (There will also be one GND)
double pad_bb = 220;       // Length and height of the bounding boxes of the pads
double w_pad = 10;         // Size of the electrical connections when exiting the
DC pads
int nr = 7;                // Number of rows of electrical pads
int nc = 7;                // Number of rows of electrical pads

// Lengths of the components after boxing (to be used with the waveguides outside
the components)
double l_bend = radius;               // Length of the S-bends added to the CDCs
double chan_lp = chan_l+2*l_bend;     // Length of the CDC of the channels (950
um)
double bandS_lp = bandS_l+2*l_bend;   // Length of the CDC of the S-band (1.51mm)
double bandC_lp = bandC_l+2*l_bend;   // Length of the CDC of the C-band (0.81mm)
double bandL_lp = bandL_l+2*l_bend;   // Length of the CDC of the L-band (1.41mm)









/* Creates a boot layout for the circuit (Connection with the shape of a boot)

Inputs: double r -> Radius of curvature of the boot
        double w -> Width of the waveguide

** Important Remarks ** All the size measures are expressed in micrometers
*/
```

```
layout boot(double r,
            double w)

{
   // From the heel side to the arch side (anterior of the boot to posterior)
   ml::demofabStraight( : 2*r, w) S1;
   ml::demofabArc(in0->S1@out0 : 90,r,w) A1;
   ml::demofabStraight(in0->A1@out0 : r+w,w) S2;
   ml::demofabArc(in0->S2@out0 : 180,r,w) A2;
   ml::demofabArc(in0->A2@out0 : -90,r,w) A3;

   // Setting the ports
   ml::setPort(this : AN->S1@in0);
   ml::setPort(this : PO->A3@out0);

   this = Tech.populateAutoRouterInformation(this);
   // Set the promoted I/O ports domain
   mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);
   mask::port2layout(&this);

}




/* Creates a channel block.

Inputs: int type -> Type of channel to be used: 0 for the initial ones of
band_blocks, 1 for the middle ones of a band and 2 for the final ones
        double MZI_l -> Length of the MZI
        double pitchInput -> pitch of the input channel CDC
        double pitch1 -> Pitch of output channel CDC #1
        double pitch2 -> Pitch of output channel CDC #2
        double pitch3 -> Pitch of output channel CDC #3
        string mk_MZI -> Mask cross-section (MCS) of the MZI
        string mk_CDC_I -> Mask cross-section (MCS) of the input channel CDC
        string mk_CDC_O -> Mask cross-section (MCS) of the output channel CDCs

** Important Remarks ** All the size measures are expressed in micrometers
(includes pitches)
*/
layout chan_block(int type,
                  double MZI_l,
                  double pitchInput,
                  double pitch1,
                  double pitch2,
                  double pitch3,
                  string mk_MZI,
                  string mk_CDC_I,
                  string mk_CDC_O
                   )

  dlgname "Channel Block"
  Domain_Optics
  AuthorInfo "Alberto Otero Casado"

{
// To be uncommented if this function layout is used alone
```

```
//var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);


// CDC Input to MZI 1
ml::CDC( :
"functor1",chan_W1,chan_W2,chan_corr1,chan_corr2,pitchInput,dt,chan_l,0,chan_G,W,
radius,width, mk_CDC_I) CI;
mask::CSselect("mcsSOI_DEEP");
ml::demofabStraight(in0->CI@Drop+[0,0,180] : bb+width/2+radius, width) S1;
ml::MZI(Input->S1@flipPortY(out0) : ptc_l, ex_l, inter_l, MZI_h, MZI_w, MZI_G,
MZI_l, W, width, w_pad, mk_MZI) M1;

// MZI 1 to 2
mask::CSselect("mcsSOI_DEEP");
ml::demofabArc(in0->M1@Drop : -90,radius,width) A1;
ml::demofabStraight(in0->A1@out0 : 3*radius, width) MM;
ml::demofabArc(in0->MM@out0 : -90,radius,width) A2;
ml::MZI(Input->A2@flipPortY(out0) : ptc_l, ex_l, inter_l, MZI_h, MZI_w, MZI_G,
MZI_l, W, width, w_pad, mk_MZI) M2;
mask::CSselect("mcsSOI_DEEP");

ml::setPort(this : In_CDC->CI@Input);  // Input of the input channel CDC (To
place the blocks correctly later)

// Electrical ports
ml::setPort(this : VO1->M1@VO);  // VO of the first MZI
ml::setPort(this : VO2->M2@VO);  // VO of the second MZI
ml::setPort(this : VI1->M1@VI);  // VI of the first MZI
ml::setPort(this : VI2->M2@VI);  // VI of the second MZI

switch (type){
   // If it is the initial channel (leftmost)
   case(0) :
      // Connections to the next channel block
      // From Input CDC to the next one
      ml::demofabStraight(in0->CI@Through : 1.5*width, width) S2;
      ml::demofabArc(in0->S2@out0 : -90,radius,width) A3;
      ml::demofabStraight(in0->A3@out0 : 4*radius+3*W+bb/2-3*width, width) S3;
      ml::demofabArc(in0->S3@out0 : -90,radius,width) A4;
      ml::demofabStraight(in0->A4@out0 : chan_lp+1.5*width, width) S4;
      ml::demofabArc(in0->S4@out0 : 180,radius,width) A5;
      // From MZI 1 to output CDC #1 (P1)
      ml::demofabStraight(in0->M1@Through : width+bb, width) S5;
      ml::demofabArc(in0->S5@out0 : -90,radius,width) A6;
      ml::demofabStraight(A6@out0 : W-width+radius, width) S6;
      ml::demofabArc(S6@out0 : 90,radius,width) A7;
      ml::demofabStraight(A7@out0 : chan_lp, width) S7;
      ml::demofabArc(in0->S7@out0 : -90,radius,width) A8;
      ml::demofabStraight(in0->A8@out0 : 4*W-5*width+bb/2+2*radius, width) S8;
      ml::demofabArc(in0->S8@out0 : -90,radius,width) A9;
      // From MZI 2 to output CDC #2 (P2)
      ml::demofabStraight(in0->M2@Through : bb/2-radius, width) S9;
      ml::demofabArc(in0->S9@out0 : -90,radius,width) A10;
      ml::demofabStraight(in0->A10@out0 : 3*radius+W+bb, width) S10;
      ml::demofabArc(in0->S10@out0 : 90,radius,width) A11;
      // From MZI 2 to output CDC #3 (P3)
      ml::demofabArc(in0->M2@Drop : -180,radius,width) A12;
```

```
      ml::demofabStraight(in0->A12@out0 : MZI_l+2*radius+bb/2+width/2, width)
S11;
      ml::demofabArc(in0->S11@out0 : 90,radius,width) A13;
      ml::demofabStraight(in0->A13@out0 : 2*W-2*width+bb+3*radius, width) S12;
      ml::demofabArc(in0->S12@out0 : -90,radius,width) A14;
      ml::demofabStraight(in0->A14@out0 : 2*radius+chan_lp-bb/2, width) S13;
      ml::demofabArc(in0->S13@out0 : 90,radius,width) A15;
      ml::demofabStraight(in0->A15@out0 : 2*W-radius-2*width, width) S14;
      ml::demofabArc(in0->S14@out0 : 90,radius,width) A16;

      // Setting the ports
      ml::setPort(this : IN->A5@out0);   // To the next input CDC on the next
channel block
      ml::setPort(this : P1->A9@out0);
      ml::setPort(this : P2->A11@out0);
      ml::setPort(this : P3->A16@out0);
      ml::setPort(this : IP->CI@Input);  // To the Input band CDC

      break;

   // If it is a channel block in the middle of the band block
   case(1) :
      // MZI 1 to output CDC #1 (P1)
      ml::demofabStraight(in0->M1@Through : 2*radius+width+bb, width) S2;
      ml::CDC(Add->S2@out0+[0,0,180] :
"functor1",chan_W1,chan_W2,chan_corr1,chan_corr2,pitch1,dt,chan_l,0,chan_G,W,radi
us,width, mk_CDC_O) CO1;

      // MZI 2 to output CDC #2 (P2)
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabStraight(in0->M2@Through : chan_lp+bb/2+radius, width) S3;
      ml::demofabArc(S3@out0 : 180,radius,width) A3;
      ml::CDC(Add->A3@out0+[0,0,180] :
"functor1",chan_W1,chan_W2,chan_corr1,chan_corr2,pitch2,dt,chan_l,0,chan_G,W,radi
us,width, mk_CDC_O) CO2;

      // MZI 2 to output CDC #3 (P3)
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabArc(in0->M2@Drop : -180,radius,width) A4;
      ml::demofabStraight(in0->A4@out0 : MZI_l+bb/2+4*radius+width/2, width) S4;
      ml::CDC(Add-
>S4@flipPortY(out0)+[0,0,180]:"functor1",chan_W1,chan_W2,chan_corr1,chan_corr2,pi
tch3,dt,chan_l,0,chan_G,W,radius,width, mk_CDC_O) CO3;
      mask::CSselect("mcsSOI_DEEP");

      // Connections to the next channel block
      // From Input CDC to the next one (on the next channel)
      ml::demofabStraight(in0->CI@Through : 1.5*width, width) S5;
      ml::demofabArc(in0->S5@out0 : -90,radius,width) A5;
      ml::demofabStraight(in0->A5@out0 : 4*radius+3*W+bb/2-3*width, width) S6;
      ml::demofabArc(in0->S6@out0 : -90,radius,width) A6;
      ml::demofabStraight(in0->A6@out0 : chan_lp+1.5*width, width) S7;
      ml::demofabArc(in0->S7@out0 : 180,radius,width) A7;
      // From output CDC #1 to the next one (on the next channel)
      ml::demofabArc(in0->CO1@Through : 90,radius,width) A8;
      ml::demofabStraight(in0->A8@out0 : radius, width) S8;
      ml::demofabArc(in0->S8@out0 : 90,radius,width) A9;
      ml::demofabStraight(in0->A9@out0 : chan_lp, width) S9;
      ml::demofabArc(in0->S9@out0 : -90,radius,width) A10;
```

```
      ml::demofabStraight(in0->A10@out0 : 4*W-5*width+bb/2+2*radius, width) S10;
      ml::demofabArc(in0->S10@out0 : -90,radius,width) A11;
      // From output CDC #2 to the next one (on the next channel)
      ml::demofabStraight(in0->CO2@Through : radius+width, width) S11;
      ml::demofabArc(in0->S11@out0 : -90,radius,width) A12;
      ml::demofabStraight(in0->A12@out0 : W-width+2*radius, width) S12;
      ml::demofabArc(in0->S12@out0 : -90,radius,width) A13;
      ml::demofabStraight(in0->A13@out0 : chan_lp+radius+width, width) S13;
      ml::demofabArc(in0->S13@out0 : 90,radius,width) A14;
      ml::demofabStraight(in0->A14@out0 : radius+W+bb, width) S14;
      ml::demofabArc(in0->S14@out0 : 90,radius,width) A15;
      // From output CDC #3 to the next one (on the next channel)
      ml::demofabArc(in0->CO3@Through : 90,radius,width) A16;
      ml::demofabStraight(in0->A16@out0 : W-width+3*radius+bb, width) S15;
      ml::demofabArc(in0->S15@out0 : 90,radius,width) A17;
      ml::demofabStraight(in0->A17@out0 : 2*radius+chan_lp-bb/2, width) S16;
      ml::demofabArc(in0->S16@out0 : -90,radius,width) A18;
      ml::demofabStraight(in0->A18@out0 : 2*W-radius-2*width, width) S17;
      ml::demofabArc(in0->S17@out0 : -90,radius,width) A19;

      // Setting the ports
      ml::setPort(this : IN->A7@out0);   // To the next Input CDC on the next
channel block
      ml::setPort(this : IP->CI@Input);  // To the previous Input CDC on the
previous channel block

      break;

   // If it is the last channel block on the band (rightmost one)
   case(2) :
      // MZI 1 to output CDC #1 (P1)
      ml::demofabStraight(in0->M1@Through : 2*radius+width+bb, width) S2;
      ml::CDC(Add->S2@out0+[0,0,180] :
"functor1",chan_W1,chan_W2,chan_corr1,chan_corr2,pitch1,dt,chan_l,0,chan_G,W,radi
us,width,mk_CDC_O) CO1;

      // MZI 2 to output CDC #2 (P2)
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabStraight(in0->M2@Through : chan_lp+bb/2+radius, width) S3;
      ml::demofabArc(S3@out0 : 180,radius,width) A3;
      ml::CDC(Add->A3@out0+[0,0,180] :
"functor1",chan_W1,chan_W2,chan_corr1,chan_corr2,pitch2,dt,chan_l,0,chan_G,W,radi
us,width,mk_CDC_O) CO2;

      // MZI 2 to output CDC #3 (P3)
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabArc(in0->M2@Drop : -180,radius,width) A4;
      ml::demofabStraight(in0->A4@out0 : MZI_l+bb/2+4*radius+width/2, width) S4;
      ml::CDC(Add-
>S4@flipPortY(out0)+[0,0,180]:"functor1",chan_W1,chan_W2,chan_corr1,chan_corr2,pi
tch3,dt,chan_l,0,chan_G,W,radius,width,mk_CDC_O) CO3;

      // Setting the port connections
      ml::setPort(this : IP->CI@Input);      // Connection to the input CDC on
the previous channel block
      ml::setPort(this : Test->CI@Through);  // Connection to the Testing Port
      ml::setPort(this : C1B->CO1@Through);  // Connection from last output
channel CDC #1
```

```
      ml::setPort(this : C2B->CO2@Through);  // Connection from last output
channel CDC #2
      ml::setPort(this : C3B->CO3@Through);  // Connection from last output
channel CDC #3
      ml::setPort(this : C2I->CO2@Input);    // Input port of the output channel
CDC #2

      break;
   }

   this = Tech.populateAutoRouterInformation(this);
   // Set the promoted I/O ports domain
   mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);
   mask::port2layout(&this);
}




/* Creates a band block formed by 8 channel blocks (an initial one to the left, 6
in the middle and a final one).
   Does not include the connections to the ports.
   There are 3 different types of bands: Initial, Middle one and Final one. The
latter only includes an input band CDC, while the others also have 3 output band
CDCs of the same band. Nevertheless, the Initial and Middle one possess different
connections to the next band CDCs

Inputs: int type -> Type of band block. 0 for initial one, 1 for the middle one
and 2 for the final one
*/
layout band_block(int type)

  dlgname "Band Block"
  Domain_Optics
  AuthorInfo "Alberto Otero Casado"

{
// To be uncommented only if the block is used alone
//var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);

switch(type) {

   // If it is an Initial band block (leftmost)
   case(0) :
      // Channel block within the band
      ml::chan_block( : 0, MZIS_l, pitchS1, pitchS1, pitchS1,
pitchS1,mask_MZI_S,mask_chanS,mask_chanS) C1;
      ml::chan_block(IP->C1@IN : 1, MZIS_l, pitchS2, pitchS2, pitchS2,
pitchS2,mask_MZI_S,mask_chanS,mask_chanS) C2;
      ml::chan_block(IP->C2@IN : 1, MZIS_l, pitchS3, pitchS3, pitchS3,
pitchL3,mask_MZI_S,mask_chanS,mask_chanS) C3;
      ml::chan_block(IP->C3@IN : 1, MZIS_l, pitchS4, pitchS4, pitchS4,
pitchL4,mask_MZI_S,mask_chanS,mask_chanS) C4;
      ml::chan_block(IP->C4@IN : 1, MZIS_l, pitchS5, pitchS5, pitchS5,
pitchL5,mask_MZI_S,mask_chanS,mask_chanS) C5;
      ml::chan_block(IP->C5@IN : 1, MZIS_l, pitchS6, pitchS6, pitchS6,
pitchL6,mask_MZI_S,mask_chanS,mask_chanS) C6;
```

```
      ml::chan_block(IP->C6@IN : 1, MZIS_l, pitchS7, pitchS7, pitchS7,
pitchL7,mask_MZI_S,mask_chanS,mask_chanS) C7;
      ml::chan_block(IP->C7@IN : 2, MZIS_l, pitchS8, pitchS8, pitchS8,
pitchL8,mask_MZI_S,mask_chanS,mask_chanS) C8;

      // From the 1st input channel CDC to the input band CDC
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabArc(in0->C1@In_CDC+[0,0,180] : -90,radius,width) A1;
      ml::demofabStraight(in0->A1@out0 : radius, width) S1;
      ml::demofabArc(in0->S1@out0 : 90,radius,width) A1X;
      //ml::demofabStraight(in0->B1@PO : radius-width,width) S2;
      ml::CDC(Drop-
>A1X@flipPortY(out0):"functor1",band_W1,band_W2,band_corr1,band_corr2,pitchS,dt,b
andS_l,chirpS,band_G,W,radius,width,mask_CDC_S) CIB;

      // From the last output channel CDC #1 to the band CDC #1
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabArc(in0->C8@C1B : 90,radius,width) A2;
      ml::demofabStraight(in0->A2@out0 : radius, width) S3;
      ml::demofabArc(in0->S3@out0 : 90,radius,width) A3;
      ml::demofabStraight(in0->A3@out0 : chan_lp, width) S4;
      ml::demofabArc(in0->S4@out0 : -90,radius,width) A4;
      ml::demofabStraight(in0->A4@out0 : 8*radius+2.5*width+5/2*bb+2*W, width)
S5;
      ml::demofabArc(in0->S5@out0 : -90,radius,width) A5;
      ml::demofabStraight(in0->A5@out0 :
2*chan_lp+MZIS_l+3/2*bb+6*radius+11*width, width) S6;
      ml::demofabArc(in0->S6@out0 : 180,radius,width) A6;
      ml::CDC(Add->A6@out0+[0,0,180] :
"functor1",band_W1,band_W2,band_corr1,band_corr2,pitchL,dt,bandL_l,chirpL,band_G,
W,radius,width,mask_CDC_L) CO1B;

      // From the last output channel CDC #2 to the band CDC #2
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabStraight(in0->C8@C2B : radius+width, width) S7;
      ml::demofabArc(in0->S7@out0 : -90,radius,width) A7;
      ml::demofabStraight(in0->A7@out0 : 5*radius+3*W-2*width, width) S8;
      ml::demofabArc(in0->S8@out0 : -90,radius,width) A8;
      ml::CDC(Add-
>A8@flipPortY(out0)+[0,0,180]:"functor1",band_W1,band_W2,band_corr1,band_corr2,pi
tchL,dt,bandL_l,chirpL,band_G,W,radius,width,mask_CDC_L) CO2B;

      // From the last output channel CDC #3 to the band CDC #3
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabStraight(in0->C8@C3B : MZIS_l+bb/2+5*radius+width) S9;
      ml::boot(PO->S9@flipPortY(out0)+[0,0,180] : radius,width) B2;
      ml::demofabStraight(in0->B2@AN+[0,0,180] :
MZIS_l+chan_lp+bb/2+5*radius+2*width, width) S10;
      ml::demofabArc(in0->S10@out0 : 90,radius,width) A9;
      ml::demofabStraight(in0->A9@out0 : 2.5*width+3*radius, width) S11;
      ml::demofabArc(in0->S11@out0 : 90,radius,width) A10;
      ml::demofabStraight(in0->A10@out0 : bandL_lp+radius-width/2, width) S12;
      ml::demofabArc(in0->S12@out0 : -180,radius,width) A11;
      ml::CDC(Add->A11@flipPortY(out0)+[0,0,180] :
"functor1",band_W1,band_W2,band_corr1,band_corr2,pitchL,dt,bandL_l,chirpL,band_G,
W,radius,width,mask_CDC_L) CO3B;

      // Connections to the next band block
      // From this input band CDC to the one at next block
```

```
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabStraight(in0->CIB@Through : 2.5*width, width) S13;
      ml::demofabArc(in0->S13@out0 : -90,radius,width) A12;
      ml::demofabStraight(in0->A12@out0 : bs_x-4*radius+width/2, width) S14;
      ml::demofabArc(in0->S14@out0 : -90,radius,width) A13;
      ml::demofabStraight(in0->A13@out0 : bandC_lp, width) S15;
      ml::demofabArc(in0->S15@out0 : 180,radius,width) A14;
      // From this output band CDC #1 to the one at next block
      ml::demofabArc(in0->CO1B@flipPortY(Input)+[0,0,180] : 180,radius,width)
A15;
      ml::demofabStraight(in0->A15@out0 : bandL_lp-6*width, width) S16;
      ml::demofabArc(in0->S16@out0 : -90,radius,width) A16;
      ml::demofabArc(in0->A16@out0 : 90,radius,width) A17;
      ml::demofabStraight(in0->A17@out0 : 2.5*bb+radius, width) S17;
      ml::demofabArc(in0->S17@out0 : -90,radius,width) A18;
      ml::demofabStraight(in0->A18@out0 : 8*ms_x+2*W+5*width+2*bb+8*radius,
width) S18;
      ml::demofabArc(in0->S18@out0 : -90,radius,width) A19;
      ml::demofabStraight(in0->A19@out0 : 2.5*bb+2*radius-7*width, width) S19;
      // From this output band CDC #2 to the one at next block
      ml::boot(PO->CO2B@Input : radius,width) B3;
      ml::demofabStraight(in0->B3@AN+[0,0,180] : bandL_lp-7*width+3/2*bb) S20;
      ml::demofabArc(in0->S20@out0 : 90,radius,width) A20;
      ml::demofabStraight(in0->A20@out0 : 7*radius+4.5*width+3*bb+2*W+8*ms_x,
width) S21;
      ml::demofabArc(in0->S21@out0 : 90,radius,width) A21;
      ml::demofabStraight(in0->A21@out0 : 3/2*bb-8*width-radius, width) S22;
      // From this output band CDC #3 to the one at next block
      ml::demofabArc(in0->CO3B@Input+[0,0,180] : -180,radius,width) A22;
      ml::demofabStraight(in0->A22@out0 : 2*chan_lp+MZIS_l+5*radius+6*width+bb/2,
width) S23;
      ml::demofabArc(in0->S23@out0 : 90,radius,width) A23;
      ml::demofabArc(in0->A23@out0 : -90,radius,width) A24;
      ml::demofabStraight(in0->A24@out0 : bb+3*radius, width) S24;
      ml::demofabArc(in0->S24@out0 : 90,radius,width) A25;
      ml::demofabStraight(in0->A25@out0 : 2*bb+8*radius+8*ms_x+6*width+2*W,
width) S25;
      ml::demofabArc(in0->S25@out0 : 90,radius,width) A26;
      ml::demofabStraight(in0->A26@out0 : 2*bb+5*width+7*radius+2*chan_lp+MZIC_l-
bandC_lp, width) S26;

      // Setting the ports
      ml::setPort(this : CDC1->C1@In_CDC);
      ml::setPort(this : IB->CIB@Input);    // Input of the Input Band CDC
      ml::setPort(this : OB->A14@out0);     // Last line connecting to the next
Input Band CDC
      ml::setPort(this : T->C8@Test);       // Through of the last input channel
CDC (To test port #1))
      ml::setPort(this : O1->CO1B@Through); // Through of the output L-band CDC
#1 (To output port #1)
      ml::setPort(this : O2->CO2B@Through); // Through of the output L-band CDC
#2 (To output port #2)
      ml::setPort(this : O3->CO3B@Through); // Through of the output L-band CDC
#3 (To output port #3)

      // Electrical ports (Output)
      ml::setPort(this : VO_1->C1@VO1);
      ml::setPort(this : VO_2->C1@VO2);
      ml::setPort(this : VO_3->C2@VO1);
```

```
      ml::setPort(this : VO_4->C2@VO2);
      ml::setPort(this : VO_5->C3@VO1);
      ml::setPort(this : VO_6->C3@VO2);
      ml::setPort(this : VO_7->C4@VO1);
      ml::setPort(this : VO_8->C4@VO2);
      ml::setPort(this : VO_9->C5@VO1);
      ml::setPort(this : VO_10->C5@VO2);
      ml::setPort(this : VO_11->C6@VO1);
      ml::setPort(this : VO_12->C6@VO2);
      ml::setPort(this : VO_13->C7@VO1);
      ml::setPort(this : VO_14->C7@VO2);
      ml::setPort(this : VO_15->C8@VO1);
      ml::setPort(this : VO_16->C8@VO2);
      ml::setPort(this : VI_1->C1@VI1);
      ml::setPort(this : VI_2->C1@VI2);
      ml::setPort(this : VI_3->C2@VI1);
      ml::setPort(this : VI_4->C2@VI2);
      ml::setPort(this : VI_5->C3@VI1);
      ml::setPort(this : VI_6->C3@VI2);
      ml::setPort(this : VI_7->C4@VI1);
      ml::setPort(this : VI_8->C4@VI2);
      ml::setPort(this : VI_9->C5@VI1);
      ml::setPort(this : VI_10->C5@VI2);
      ml::setPort(this : VI_11->C6@VI1);
      ml::setPort(this : VI_12->C6@VI2);
      ml::setPort(this : VI_13->C7@VI1);
      ml::setPort(this : VI_14->C7@VI2);
      ml::setPort(this : VI_15->C8@VI1);
      ml::setPort(this : VI_16->C8@VI2);


      break;

   // If it is the middle band block
   case(1) :
      // Channel blocks within the band
      ml::chan_block( : 0, MZIC_l, pitchC1, pitchC1, pitchC1,
pitchC1,mask_MZI_C,mask_chanC,mask_chanC) C1;
      ml::chan_block(IP->C1@IN : 1, MZIC_l, pitchC2, pitchC2, pitchC2,
pitchC2,mask_MZI_C,mask_chanC,mask_chanC) C2;
      ml::chan_block(IP->C2@IN : 1, MZIC_l, pitchC3, pitchC3, pitchC3,
pitchC3,mask_MZI_C,mask_chanC,mask_chanC) C3;
      ml::chan_block(IP->C3@IN : 1, MZIC_l, pitchC4, pitchC4, pitchC4,
pitchC4,mask_MZI_C,mask_chanC,mask_chanC) C4;
      ml::chan_block(IP->C4@IN : 1, MZIC_l, pitchC5, pitchC5, pitchC5,
pitchC5,mask_MZI_C,mask_chanC,mask_chanC) C5;
      ml::chan_block(IP->C5@IN : 1, MZIC_l, pitchC6, pitchC6, pitchC6,
pitchC6,mask_MZI_C,mask_chanC,mask_chanC) C6;
      ml::chan_block(IP->C6@IN : 1, MZIC_l, pitchC7, pitchC7, pitchC7,
pitchC7,mask_MZI_C,mask_chanC,mask_chanC) C7;
      ml::chan_block(IP->C7@IN : 2, MZIC_l, pitchC8, pitchC8, pitchC8,
pitchC8,mask_MZI_C,mask_chanC,mask_chanC) C8;

      // From the 1st input channel CDC to the input band CDC
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabArc(in0->C1@In_CDC+[0,0,180] : -90,radius,width) A1;
      ml::demofabStraight(in0->A1@out0 : radius, width) S1;
      ml::demofabArc(in0->S1@out0 : 90,radius,width) A1X;
```

```
      ml::demofabStraight(in0->A1X@out0 : bandS_lp-bandC_lp+2*radius-
6.5*width,width) S2;
      ml::CDC(Drop->S2@flipPortY(out0) :
"functor1",band_W1,band_W2,band_corr1,band_corr2,pitchC,dt,bandC_l,chirpC,band_G,
W,radius,width,mask_CDC_C) CIB;

      // From the last output channel CDC #1 to the band CDC #1
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabArc(in0->C8@C1B : 90,radius,width) A2;
      ml::demofabStraight(in0->A2@out0 : radius, width) S3;
      ml::demofabArc(in0->S3@out0 : 90,radius,width) A3;
      ml::demofabStraight(in0->A3@out0 : chan_lp, width) S4;
      ml::demofabArc(in0->S4@out0 : -90,radius,width) A4;
      ml::demofabStraight(in0->A4@out0 : 8*radius+2*width+5/2*bb+2*W, width) S5;
      ml::demofabArc(in0->S5@out0 : -90,radius,width) A5;
      ml::demofabStraight(in0->A5@out0 :
2*chan_lp+MZIC_l+3/2*bb+6*radius+11*width, width) S6;
      ml::demofabArc(in0->S6@out0 : 180,radius,width) A6;
      ml::CDC(Add->A6@out0+[0,0,180] :
"functor1",band_W1,band_W2,band_corr1,band_corr2,pitchC,dt,bandC_l,chirpC,band_G,
W,radius,width,mask_CDC_C) CO1B;

      // From the last output channel CDC #2 to the band CDC #2
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabStraight(in0->C8@C2B : radius+width, width) S7;
      ml::demofabArc(in0->S7@out0 : -90,radius,width) A7;
      ml::demofabStraight(in0->A7@out0 : 5*radius+3*W-2*width, width) S8;
      ml::demofabArc(in0->S8@out0 : -90,radius,width) A8;
      ml::CDC(Add-
>A8@flipPortY(out0)+[0,0,180]:"functor1",band_W1,band_W2,band_corr1,band_corr2,pi
tchC,dt,bandC_l,chirpC,band_G,W,radius,width,mask_CDC_C) CO2B;

      // From the last output channel CDC #3 to the band CDC #3
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabStraight(in0->C8@C3B : MZIC_l+bb/2+5*radius+width) S9;
      ml::boot(PO->S9@flipPortY(out0)+[0,0,180] : radius,width) B2;
      ml::demofabStraight(in0->B2@AN+[0,0,180] :
MZIC_l+chan_lp+bb/2+5*radius+2*width, width) S10;
      ml::demofabArc(in0->S10@out0 : 90,radius,width) A9;
      ml::demofabStraight(in0->A9@out0 : 3*width+3*radius, width) S11;
      ml::demofabArc(in0->S11@out0 : 90,radius,width) A10;
      ml::demofabStraight(in0->A10@out0 : bandC_lp+radius-width/2, width) S12;
      ml::demofabArc(in0->S12@out0 : -180,radius,width) A11;
      ml::CDC(Add->A11@flipPortY(out0)+[0,0,180] :
"functor1",band_W1,band_W2,band_corr1,band_corr2,pitchC,dt,bandC_l,chirpC,band_G,
W,radius,width,mask_CDC_C) CO3B;

      // Connections to the next band block
      // From this input band CDC to the one at next block
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabArc(in0->CIB@Through : -90,radius,width) A12;
      ml::demofabStraight(in0->A12@out0 : bs_x-4*radius, width) S14;
      ml::demofabArc(in0->S14@out0 : -90,radius,width) A13;
      ml::demofabStraight(in0->A13@out0 : bandL_lp, width) S15;
      ml::demofabArc(in0->S15@out0 : 180,radius,width) A14;
      // From this output band CDC #1 to the one at next block
      ml::demofabArc(in0->CO1B@flipPortY(Input)+[0,0,180] : 180,radius,width)
A15;
      ml::demofabStraight(in0->A15@out0 : bandC_lp-6*width, width) S16;
```

```
      ml::demofabArc(in0->S16@out0 : -90,radius,width) A16;
      ml::demofabArc(in0->A16@out0 : 90,radius,width) A17;
      ml::demofabStraight(in0->A17@out0 : 2.5*bb-width, width) S17;
      ml::demofabArc(in0->S17@out0 : -90,radius,width) A18;
      ml::demofabStraight(in0->A18@out0 : 8*ms_x+5*width+2*radius, width) S18;
      ml::demofabArc(in0->S18@out0 : -90,radius,width) A19;
      ml::demofabStraight(in0->A19@out0 : 2*chan_lp+MZIL_l+7*radius+4*bb+5*width,
width) S19;
      ml::demofabArc(in0->S19@out0 : -90,radius,width) A19_A;
      ml::demofabStraight(in0->A19_A@out0 : 0.5*bb+W+2*radius+width, width)
S19_A;
      ml::demofabArc(in0->S19_A@out0 : -90,radius,width) A19_B;
      ml::demofabStraight(in0->A19_B@out0 : chan_lp, width) S19_B;
      ml::demofabArc(in0->S19_B@out0 : 90,radius,width) A19_C;
      ml::demofabStraight(in0->A19_C@out0 : radius, width) S19_C;
      ml::demofabArc(in0->S19_C@out0 : 90,radius,width) A19_D;
      // From this output band CDC #2 to the one at next block
      ml::boot(PO->CO2B@Input : radius,width) B3;
      ml::demofabStraight(in0->B3@AN+[0,0,180] : bandC_lp-8*width+3/2*bb-radius,
width) S20;
      ml::demofabArc(in0->S20@out0 : 90,radius,width) A20;
      ml::demofabStraight(in0->A20@out0 : 6*radius+3*width+bb+8*ms_x, width) S21;
      ml::demofabArc(in0->S21@out0 : 90,radius,width) A21;
      ml::demofabStraight(in0->A21@out0 : 0.5*bb+4*width+2*radius, width) S22;
      // From this output band CDC #3 to the one at next block
      ml::demofabArc(in0->CO3B@Input+[0,0,180] : -180,radius,width) A22;
      ml::demofabStraight(in0->A22@out0 : 2*chan_lp+MZIC_l+5*radius+6*width+bb/2,
width) S23;
      ml::demofabArc(in0->S23@out0 : 90,radius,width) A23;
      ml::demofabArc(in0->A23@out0 : -90,radius,width) A24;
      ml::demofabStraight(in0->A24@out0 : 1.5*bb-width, width) S24;
      ml::demofabArc(in0->S24@out0 : 90,radius,width) A25;
      ml::demofabStraight(in0->A25@out0 : 3*bb+8*radius+7*ms_x+3*width+2*W,
width) S25;
      ml::demofabArc(in0->S25@out0 : 90,radius,width) A26;
      ml::demofabStraight(in0->A26@out0 : chan_lp+MZIL_l+3*bb+5*radius+4.5*width,
width) S26;

      // Setting the ports
      ml::setPort(this : CDC1->C1@In_CDC);
      ml::setPort(this : IB->CIB@Input);    // Input of the Input Band CDC
      ml::setPort(this : OB->A14@out0);     // Last arch connecting to the next
Input Band CDC
      ml::setPort(this : T->C8@Test);       // Testing port of the last input
channel CDC

      // Port set to have the position of the bottom-right corner of the bounding
box of the optical parts
      ml::setPort(this : southeast->S19@out0+[width/2,-radius-width/2,0]);
      // Port set to have the position of the top-right corner of the bounding
box of the optical parts
      ml::setPort(this : northeast->S19@in0+[-radius-width/2,width/2,-90]);

      // Electrical ports (MZIs)
      ml::setPort(this : VO_1->C1@VO1);
      ml::setPort(this : VO_2->C1@VO2);
      ml::setPort(this : VO_3->C2@VO1);
      ml::setPort(this : VO_4->C2@VO2);
      ml::setPort(this : VO_5->C3@VO1);
```

```
      ml::setPort(this : VO_6->C3@VO2);
      ml::setPort(this : VO_7->C4@VO1);
      ml::setPort(this : VO_8->C4@VO2);
      ml::setPort(this : VO_9->C5@VO1);
      ml::setPort(this : VO_10->C5@VO2);
      ml::setPort(this : VO_11->C6@VO1);
      ml::setPort(this : VO_12->C6@VO2);
      ml::setPort(this : VO_13->C7@VO1);
      ml::setPort(this : VO_14->C7@VO2);
      ml::setPort(this : VO_15->C8@VO1);
      ml::setPort(this : VO_16->C8@VO2);
      ml::setPort(this : VI_1->C1@VI1);
      ml::setPort(this : VI_2->C1@VI2);
      ml::setPort(this : VI_3->C2@VI1);
      ml::setPort(this : VI_4->C2@VI2);
      ml::setPort(this : VI_5->C3@VI1);
      ml::setPort(this : VI_6->C3@VI2);
      ml::setPort(this : VI_7->C4@VI1);
      ml::setPort(this : VI_8->C4@VI2);
      ml::setPort(this : VI_9->C5@VI1);
      ml::setPort(this : VI_10->C5@VI2);
      ml::setPort(this : VI_11->C6@VI1);
      ml::setPort(this : VI_12->C6@VI2);
      ml::setPort(this : VI_13->C7@VI1);
      ml::setPort(this : VI_14->C7@VI2);
      ml::setPort(this : VI_15->C8@VI1);
      ml::setPort(this : VI_16->C8@VI2);

      break;

   // If it is a Final band block (rightmost)
   case(2) :
      // Channel blocks within the band
      ml::chan_block( : 0, MZIL_l, pitchL1, pitchL1, pitchL1,
pitchL1,mask_MZI_L,mask_chanL,mask_chanL) C1;
      ml::chan_block(IP->C1@IN : 1, MZIL_l, pitchL2, pitchL2, pitchL2, pitchL2,
mask_MZI_L, mask_chanL, mask_chanL) C2;
      ml::chan_block(IP->C2@IN : 1, MZIL_l, pitchL3, pitchL3, pitchL3, pitchL3,
mask_MZI_L, mask_chanL, mask_chanL) C3;
      ml::chan_block(IP->C3@IN : 1, MZIL_l, pitchL4, pitchL4, pitchL4, pitchL4,
mask_MZI_L, mask_chanL, mask_chanL) C4;
      ml::chan_block(IP->C4@IN : 1, MZIL_l, pitchL5, pitchL5, pitchL5, pitchL5,
mask_MZI_L, mask_chanL, mask_chanL) C5;
      ml::chan_block(IP->C5@IN : 1, MZIL_l, pitchL6, pitchL6, pitchL6, pitchL6,
mask_MZI_L, mask_chanL, mask_chanL) C6;
      ml::chan_block(IP->C6@IN : 1, MZIL_l, pitchL7, pitchL7, pitchL7, pitchL7,
mask_MZI_L, mask_chanL, mask_chanL) C7;
      ml::chan_block(IP->C7@IN : 2, MZIL_l, pitchL8, pitchL8, pitchL8, pitchL8,
mask_MZI_L, mask_chanL, mask_chanL) C8;

      // From the 1st input channel CDC to the input band CDC
      mask::CSselect("mcsSOI_DEEP");
      ml::demofabStraight(in0->C1@In_CDC+[0,0,180] : bandS_lp-bandL_lp+2.5*width,
width) S0;
      ml::demofabArc(in0->S0@out0 : -90,radius,width) A1;
      ml::demofabStraight(in0->A1@out0 : radius, width) S1;
      ml::demofabArc(in0->S1@out0 : 90,radius,width) A2;
```

```
      ml::CDC(Drop->A2@flipPortY(out0) :
"functor1",band_W1,band_W2,band_corr1,band_corr2,pitchL,dt,bandL_l,chirpL,band_G,
W,radius,width,mask_CDC_L) CIB;

      // Setting the ports
      ml::setPort(this : CDC1->C1@In_CDC);
      ml::setPort(this : IB->CIB@Input);                  // Input of the Input
Band CDC
      ml::setPort(this : TC->C8@Test);                    // Through port of the
last input channel CDC (To testing Port #3)
      ml::setPort(this : TB->CIB@Through);                // Throught port of the
input L-band CDC (To testing Port #4)
      ml::setPort(this : LC->C8@C3B+[0,MZIL_l,0]);        // Last output channel
CDC #3 of the band (IN port)
      ml::setPort(this : L2->C8@C2I);                     // Last output channel
CDC #2 of the last band (IN port)

      // Electrical ports of the MZIs
      ml::setPort(this : VO_1->C1@VO1);
      ml::setPort(this : VO_2->C1@VO2);
      ml::setPort(this : VO_3->C2@VO1);
      ml::setPort(this : VO_4->C2@VO2);
      ml::setPort(this : VO_5->C3@VO1);
      ml::setPort(this : VO_6->C3@VO2);
      ml::setPort(this : VO_7->C4@VO1);
      ml::setPort(this : VO_8->C4@VO2);
      ml::setPort(this : VO_9->C5@VO1);
      ml::setPort(this : VO_10->C5@VO2);
      ml::setPort(this : VO_11->C6@VO1);
      ml::setPort(this : VO_12->C6@VO2);
      ml::setPort(this : VO_13->C7@VO1);
      ml::setPort(this : VO_14->C7@VO2);
      ml::setPort(this : VO_15->C8@VO1);
      ml::setPort(this : VO_16->C8@VO2);
      ml::setPort(this : VI_1->C1@VI1);
      ml::setPort(this : VI_2->C1@VI2);
      ml::setPort(this : VI_3->C2@VI1);
      ml::setPort(this : VI_4->C2@VI2);
      ml::setPort(this : VI_5->C3@VI1);
      ml::setPort(this : VI_6->C3@VI2);
      ml::setPort(this : VI_7->C4@VI1);
      ml::setPort(this : VI_8->C4@VI2);
      ml::setPort(this : VI_9->C5@VI1);
      ml::setPort(this : VI_10->C5@VI2);
      ml::setPort(this : VI_11->C6@VI1);
      ml::setPort(this : VI_12->C6@VI2);
      ml::setPort(this : VI_13->C7@VI1);
      ml::setPort(this : VI_14->C7@VI2);
      ml::setPort(this : VI_15->C8@VI1);
      ml::setPort(this : VI_16->C8@VI2);

      break;
}


this = Tech.populateAutoRouterInformation(this);
// Set the promoted I/O ports domain
mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);
mask::port2layout(&this);
```

```
}



/*
Creates the WSS layout with the optical ports and the band blocks

Input: e_pads -> 0 if the electrical connections + pads will be set in place. 1
if they won't. (default = 0)
*/
layout WSS(int e_pads = 0)

  dlgname "WSS block"
  Domain_Optics
  AuthorInfo "Alberto Otero Casado"

{

   var RoutingGrid = Tech.getDefaultOrNewAutoRouterGrid(this, "Optical",
"FinalCircuitOptical", 40, 40);

   // Concatenating the bands
   ml::band_block( : 0) B1;
   ml::band_block(IB->B1@OB : 1) B2;
   ml::band_block(IB->B2@OB : 2) B3;

   // Connections to the ports
   // From the input S-band CDC to the input port
   mask::CSselect("mcsSOI_DEEP");
   ml::demofabArc(in0->B1@IB+[0,0,180] : -180,radius,width) A1;
   ml::demofabStraight(in0->A1@out0 : bandS_lp+3.5*width-2*port_d+5*radius-bb,
width) S1;
   ml::demofabArc(in0->S1@out0 : 90, radius,width) A2;
   ml::demofabStraight(in0->A2@out0 : 6*radius+6*width, width) S2;
   ml::demofabFiberCoupler(out0->S2@out0+[0,0,180] : ) FC1;
   // From the last input channel CDC of the first band block to Testing port #1
   ml::demofabStraight(in0->B1@T : 2.5*width+radius, width) S3;
   ml::demofabArc(in0->S3@out0 : 90,radius,width) A3;
   ml::demofabStraight(in0->A3@out0 : 7*ms_x+4*radius+W, width) S4;
   ml::demofabArc(in0->S4@out0 : 90,radius,width) A4;
   ml::demofabStraight(in0->A4@out0 : MZIS_l+2*chan_lp-
3*port_d+6*width+2*bb+8*radius) S5;
   ml::demofabArc(in0->S5@out0 : -90,radius,width) A5;
   ml::demofabStraight(in0->A5@out0 : 5*width+5*radius, width) S6;
   ml::demofabFiberCoupler(out0->S6@out0+[0,0,180] : ) FC2;
   // From the output L-band CDC #2 to output port #2
   ml::demofabStraight(in0->B1@O2 : 0.5*bb+radius+3*width, width) S7;
   ml::demofabArc(in0->S7@out0 : -90,radius,width) A6;
   ml::demofabStraight(in0->A6@out0 : 8*ms_x+2*W+6*radius+2*width, width) S8;
   ml::demofabArc(in0->S8@out0 : -90,radius,width) A7;
   ml::demofabStraight(in0->A7@out0 : 2*chan_lp+MZIS_l-
4*port_d+7*radius+7*width+2.5*bb, width) S9;
   ml::demofabArc(in0->S9@out0 : 90,radius,width) A8;
   ml::demofabStraight(in0->A8@out0 : 4*width+4*radius, width) S10;
   ml::demofabFiberCoupler(out0->S10@out0+[0,0,180] : ) FC3;
   // From the output L-band CDC #3 to output port #3
```

```
   ml::demofabStraight(in0->B1@O3 : 2*chan_lp+MZIS_l-
bandL_lp+6*radius+6*width+2.5*bb, width) S11;
   ml::demofabArc(in0->S11@out0 : -90,radius,width) A9;
   ml::demofabStraight(in0->A9@out0 : 8*ms_x+12*radius+5.5*width+2*W, width) S12;
   ml::demofabArc(in0->S12@out0 : -90,radius,width) A10;
   ml::demofabStraight(in0->A10@out0 : 2*chan_lp+MZIS_l-
5*port_d+9*radius+7*width+3*bb, width) S13;
   ml::demofabArc(in0->S13@out0 : 90,radius,width) A11;
   ml::demofabStraight(in0->A11@out0 : 3*width+3*radius, width) S14;
   ml::demofabFiberCoupler(out0->S14@out0+[0,0,180] : ) FC4;
   // From the output L-band CDC #1 to output port #1
   ml::demofabStraight(in0->B1@O1 : 3*radius-6*width+2.5*bb, width) S15;
   ml::demofabArc(in0->S15@out0 : -90,radius,width) A12;
   ml::demofabStraight(in0->A12@out0 : 8*ms_x+13*radius+7.5*width+2*W+2*bb,
width) S16;
   ml::demofabArc(in0->S16@out0 : -90,radius,width) A13;
   ml::demofabStraight(in0->A13@out0 : 2*chan_lp+MZIS_l-
6*port_d+3.5*bb+11*radius+7*width, width) S17;
   ml::demofabArc(in0->S17@out0 : 90,radius,width) A14;
   ml::demofabStraight(in0->A14@out0 : 2*width+2*radius, width) S18;
   ml::demofabFiberCoupler(out0->S18@out0+[0,0,180] : ) FC5;
   // From the last input Band CDC (L-band) to Testing port #4
   ml::demofabStraight(in0->B3@TB : radius+0.5*width, width) S19;
   ml::demofabArc(in0->S19@out0 : 90,radius,width) A15;
   ml::demofabStraight(in0->A15@out0 : 2*bs_x+6*radius+6.5*width, width) S20;
   ml::demofabArc(in0->S20@out0 : 90,radius,width) A16;
   ml::demofabStraight(in0->A16@out0 : port_d-2*radius-width/2, width) S21;
   ml::demofabArc(in0->S21@out0 : -90,radius,width) A17;
   ml::demofabFiberCoupler(out0->A17@out0+[0,0,180] : ) FC6;
   // From the last input channel CDC of the middle band block (C-Band) to
Testing port #2
   ml::demofabStraight(in0->B2@T : 2.5*bb+2*radius+2.5*width, width) S22;
   ml::demofabArc(in0->S22@out0 : 90,radius,width) A18;
   ml::demofabStraight(in0->A18@out0 : bs_x+7*ms_x+8*radius+W+4.5*width, width)
S23;
   ml::demofabArc(in0->S23@out0 : 90,radius,width) A19;
   ml::demofabStraight(in0->A19@out0 : MZIS_l+2*chan_lp-
7*port_d+10*radius+7*width+4.5*bb, width) S24;
   ml::demofabArc(in0->S24@out0 : -90,radius,width) A20;
   ml::demofabStraight(in0->A20@out0 : width+radius, width) S25;
   ml::demofabFiberCoupler(out0->S25@out0+[0,0,180] : ) FC7;
   // From the last input channel CDC of the final band block (L-Band) to Testing
port #3
   ml::demofabStraight(in0->B3@TC : 2.5*bb+4*radius+4.5*width, width) S26;
   ml::demofabArc(in0->S26@out0 : 90,radius,width) A21;
   ml::demofabStraight(in0->A21@out0 : 2*bs_x+7*ms_x+9*radius+W+5.5*width, width)
S27;
   ml::demofabArc(in0->S27@out0 : 90,radius,width) A22;
   ml::demofabStraight(in0->A22@out0 : MZIS_l+2*chan_lp-
8*port_d+11*radius+8*width+4.5*bb, width) S28;
   ml::demofabArc(in0->S28@out0 : -90,radius,width) A23;
   ml::demofabFiberCoupler(out0->A23@out0+[0,0,180] : ) FC8;


   // To see the northernmost and southermost points regarding the optical layout
   ml::setPort(this : south-> S20@in0+[0,-width/2,0]);
   ml::setPort(this : north->S27@in0+[0,width/2,0]);
```

```
   // Electrical connections and pads if specified by the input e_pads
   if (e_pads == 1){

   var pads [n_mzi+1];     // Electrical pads (The number of MZIs used plus a
ground line)
   var ruler;              // Ruler for the distances from the electrical pads to
the MZI ports
   var v_lines[n_mzi+1];   // Vertical electrical lines exiting from the DC Pads
(i+1 so that it coincides with pads)
   double y;               // Length from the upper to the lower part of the
optical layout
   double y1;              // Vertical distance from the 1st pad to the IN port
of the output channel CDCs #3

   ruler = ml::diffPort(this@north,B2@northeast);
   y = abs(ruler.y); y = y+width;


   /* It has been decided to create the 49 electrical pads as an array of 7x7.
This value has been taken because
      it seems to be a good one to have the electrical lines at the bottom and
the GND line on top.
      Anyway, other configurations are possible.
      The [0,0] pad corresponds to the ground
   */
   // Creating the electrical pads and their main lines so that they do not
coincide vertically
   for (int j = 0; j < nc; j++){
      for (int i = 0; i < nr; i++){
         pads[nr*j+i] = ml::demofabDCPad_bidir(dc1->B2@northeast-[(1.5*nr-
0.5)*w_pad*(j+1)+pad_bb*j,y-pad_bb/2-i*(pad_bb),0] : );
         mask::CSselect("mcsVia_metal");

         // For the next vertical lines (dont change and have pads[0]@dc1)
         if (i == 0 && j == 0){
            ruler = ml::diffPort(pads[0]@dc1,B3@LC);
            // Vertica distance from the first pad to the IN port of the last
output channel CDC#3 (last channel of the last band)
            y1 = abs(ruler.y);
         }
         else{
            ml::demofabStraight(in0->pads[nr*j+i]@dc1 : 1.5*w_pad*(nr-1-i),
w_pad);
            v_lines[nr*j+i] = ml::demofabStraight(last+[w_pad/2,-w_pad/2,90] :
y1-i*pad_bb-w_pad+i*1.5*w_pad +j*(nr*1.5*w_pad) , w_pad);
         }
      }
   }


   // Voltage lines to every MZI
   // To the final band block
   ruler = ml::diffPort(v_lines[1]@out0, B3@VI_16);
   ml::demofabStraight(in0->v_lines[1]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[2]@out0, B3@VO_15);
```

```
   ml::demofabStraight(in0->v_lines[2]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[3]@out0, B3@VI_14);
   ml::demofabStraight(in0->v_lines[3]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[4]@out0, B3@VO_13);
   ml::demofabStraight(in0->v_lines[4]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[5]@out0, B3@VI_12);
   ml::demofabStraight(in0->v_lines[5]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[6]@out0, B3@VO_11);
   ml::demofabStraight(in0->v_lines[6]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[7]@out0, B3@VI_10);
   ml::demofabStraight(in0->v_lines[7]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[8]@out0, B3@VO_9);
   ml::demofabStraight(in0->v_lines[8]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[9]@out0, B3@VI_8);
   ml::demofabStraight(in0->v_lines[9]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[10]@out0, B3@VO_7);
   ml::demofabStraight(in0->v_lines[10]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[11]@out0, B3@VI_6);
   ml::demofabStraight(in0->v_lines[11]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[12]@out0, B3@VO_5);
   ml::demofabStraight(in0->v_lines[12]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[13]@out0, B3@VI_4);
   ml::demofabStraight(in0->v_lines[13]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
```

```
   ruler = ml::diffPort(v_lines[14]@out0, B3@VO_3);
   ml::demofabStraight(in0->v_lines[14]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[15]@out0, B3@VI_2);
   ml::demofabStraight(in0->v_lines[15]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[16]@out0, B3@VO_1);
   ml::demofabStraight(in0->v_lines[16]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   // To the middle band block
   ruler = ml::diffPort(v_lines[17]@out0, B2@VI_16);
   ml::demofabStraight(in0->v_lines[17]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[18]@out0, B2@VO_15);
   ml::demofabStraight(in0->v_lines[18]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[19]@out0, B2@VI_14);
   ml::demofabStraight(in0->v_lines[19]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[20]@out0, B2@VO_13);
   ml::demofabStraight(in0->v_lines[20]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[21]@out0, B2@VI_12);
   ml::demofabStraight(in0->v_lines[21]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[22]@out0, B2@VO_11);
   ml::demofabStraight(in0->v_lines[22]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[23]@out0, B2@VI_10);
   ml::demofabStraight(in0->v_lines[23]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[24]@out0, B2@VO_9);
   ml::demofabStraight(in0->v_lines[24]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[25]@out0, B2@VI_8);
   ml::demofabStraight(in0->v_lines[25]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
```

```
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[26]@out0, B2@VO_7);
   ml::demofabStraight(in0->v_lines[26]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[27]@out0, B2@VI_6);
   ml::demofabStraight(in0->v_lines[27]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[28]@out0, B2@VO_5);
   ml::demofabStraight(in0->v_lines[28]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[29]@out0, B2@VI_4);
   ml::demofabStraight(in0->v_lines[29]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[30]@out0, B2@VO_3);
   ml::demofabStraight(in0->v_lines[30]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[31]@out0, B2@VI_2);
   ml::demofabStraight(in0->v_lines[31]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[32]@out0, B2@VO_1);
   ml::demofabStraight(in0->v_lines[32]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   // To the initial band block
   ruler = ml::diffPort(v_lines[33]@out0, B1@VI_16);
   ml::demofabStraight(in0->v_lines[33]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[34]@out0, B1@VO_15);
   ml::demofabStraight(in0->v_lines[34]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[35]@out0, B1@VI_14);
   ml::demofabStraight(in0->v_lines[35]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[36]@out0, B1@VO_13);
   ml::demofabStraight(in0->v_lines[36]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[37]@out0, B1@VI_12);
```

```
   ml::demofabStraight(in0->v_lines[37]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[38]@out0, B1@VO_11);
   ml::demofabStraight(in0->v_lines[38]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[39]@out0, B1@VI_10);
   ml::demofabStraight(in0->v_lines[39]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[40]@out0, B1@VO_9);
   ml::demofabStraight(in0->v_lines[40]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[41]@out0, B1@VI_8);
   ml::demofabStraight(in0->v_lines[41]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[42]@out0, B1@VO_7);
   ml::demofabStraight(in0->v_lines[42]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[43]@out0, B1@VI_6);
   ml::demofabStraight(in0->v_lines[43]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[44]@out0, B1@VO_5);
   ml::demofabStraight(in0->v_lines[44]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[45]@out0, B1@VI_4);
   ml::demofabStraight(in0->v_lines[45]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[46]@out0, B1@VO_3);
   ml::demofabStraight(in0->v_lines[46]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[47]@out0, B1@VI_2);
   ml::demofabStraight(in0->v_lines[47]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
   ruler = ml::diffPort(v_lines[48]@out0, B1@VO_1);
   ml::demofabStraight(in0->v_lines[48]@out0+[w_pad/2,w_pad/2,-90] :
abs(ruler.y)+0.5*w_pad, w_pad);
   ml::demofabStraight(last+[w_pad/2,w_pad/2,-90] : abs(ruler.x)+1.5*w_pad,
w_pad);
```

```
   // Ground lines to the MZIs
   // Common Ground line
   ml::demofabStraight(in0->pads[0]@dc1 : 1.5*w_pad*(nr-1), w_pad) GND_1;
   // Distance from the last wvg to the TH port of channel CDC#2 (last channel of
the last band)
   ruler = ml::diffPort(GND_1@out0, B3@L2);
   v_lines[0] = ml::demofabStraight(last+[w_pad/2,-w_pad/2,90] : abs(ruler.y)-
w_pad, w_pad);
   ruler = ml::diffPort(v_lines[0]@out0, B1@VI_1);  // Distance to the first MZI
(VI port)
   ml::demofabStraight(last+[-w_pad/2,w_pad/2,-90] : abs(ruler.y)+1.5*w_pad,
w_pad) GND_L;
   // GND lines to the initial band MZIs
   ml::demofabStraight(last+[-w_pad/2,w_pad/2,90] : abs(ruler.x)+0.5*w_pad,
w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VO_2);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VI_3);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VO_4);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VI_5);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VO_6);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VI_7);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VO_8);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VI_9);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VO_10);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VI_11);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VO_12);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VI_13);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VO_14);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B1@VI_15);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
```

```
   ruler = ml::diffPort(GND_L@out0, B1@VO_16);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   // GND lines to the middle band MZIs
   ruler = ml::diffPort(GND_L@out0, B2@VI_1);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VO_2);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VI_3);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VO_4);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VI_5);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VO_6);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VI_7);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VO_8);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VI_9);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VO_10);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VI_11);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VO_12);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VI_13);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VO_14);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VI_15);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B2@VO_16);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   // GND lines to the final band MZIs
   ruler = ml::diffPort(GND_L@out0, B3@VI_1);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VO_2);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
```

```
   ruler = ml::diffPort(GND_L@out0, B3@VI_3);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VO_4);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VI_5);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VO_6);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VI_7);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VO_8);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VI_9);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VO_10);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VI_11);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VO_12);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VI_13);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VO_14);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VI_15);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);
   ruler = ml::diffPort(GND_L@out0, B3@VO_16);
   ml::demofabStraight(in0->GND_L@out0+[w_pad/2+ruler.x,w_pad/2,90] :
abs(ruler.y), w_pad);

   }

   // Setting the ports
   ml::setPort(this : IBB->B1@IB);  // Port at the Input port of the Input Band
CDC of the S-Band
   ml::setPort(this : I1->B1@CDC1); // Port to the Input of the first input
channel CDC


   this = Tech.populateAutoRouterInformation(this);
   // Set the promoted I/O ports domain
   mask::elementPortRegexSetDomain(this, "(in|out)([[:int:]])", OpticsDomain);
   mask::port2layout(&this);

}
```

```
/*****************************************************************************
Trials to implement the channel blocks and band blocks

** Important remark ** The sentence at the beginning of the function layouts
chann_block() and band_block() (var RoutingGrid) must be uncommented to implement
these trials
*****************************************************************************/


// Trial 1 for the initial channel block
//ml::chan_block(IP->[0,0,90] : 0, MZIS_l, pitchS1, pitchS1, pitchS1,
pitchS1,mask_MZI_S,mask_chanS,mask_chanS) C1;

// Trial 2 for a middle channel block
//ml::chan_block(IP->[0,0,90] : 1, MZIS_l, pitchS2, pitchS2, pitchS2,
pitchS2,mask_MZI_S,mask_chanS,mask_chanS) C2;

// Trial 3 for the final channel block
//ml::chan_block(IP->[0,0,90] : 2, MZIS_l, pitchS8, pitchS8, pitchS8,
pitchS8,mask_MZI_S,mask_chanS,mask_chanS) C8;

//Trial 4 for the initial band block
//ml::band_block(CDC1->[0,0,90] : 0) B1;

// Trial 5 for the middle band block
//ml::band_block(CDC1->[0,0,90] : 1) B2;

// Trial 6 for the final band block
//ml::band_block(CDC1->[0,0,90] : 2) B3;



/*****************************************************************************
Trials to implement the WSS layout

** Important Remark ** The sentences at the beginning of the function layouts
chann_block(), band_block(), MZI() and CDC() (var RoutingGrid) must all be
commented to implement these trials
*****************************************************************************/

// Trial 1 to implement the WSS optical layout
//ml::WSS(I1->[0,0,90] : );

// Trial 2 to implement the WSS full layout (optical + electrical)
ml::WSS(I1->[0,0,90] : 1);
```

Appendix F. Statement of non-plagiarism

I hereby declare that all information in this report has been obtained and presented in accordance with academic rules and ethical conduct and the work I am submitting in this report, except where I have indicated, is my own work.

Appendix G. Supervisor approval

I, the undersigned, Paolo Bardella, supervisor of Alberto Otero Casado, student of the PSRS EMJMD, during his master thesis at Politecnico di Torino certify that I approve the content of this master thesis report entitled “Silicon Photonics Chip for Telecom applications”.

Appendix H. Copyright of the figures

Figure 1

The presented figure is a modification of the 5th figure on [7], and it belongs to its authors: Near Margalit, Chao Xiang, Steven M. Bowers, Alexis Bjorlin, Robert Blum and John E. Bowers.

The image has been modified, so that it only includes (a).

Figure 2

Figure 3

Figure 4

The presented figure has been created by Lorenzo Tunesi, PhD student at Politecnico di Torino. I have been given permission to use it for the purpose of this report.

Figure 5

The presented figure is a modification of the one created by Ansgar Hellwig, which has been released into the public domain. Link to the original file and public domain statement

Figure 6

Figure 7

Figure 8

Figure 9

The image has been modified, with the addition of the headings “Output CDCs” and “Output CDCs”.

Figure 10

Figure 11

Figure 12

Figure 13

Figure 14

Figure 15

Figure 16

Figure 17

Figure 18

Figure 19

Figure 20

Figure 21

The image has been modified. The numerical values of the different parameters of the MZI have been changed by their symbolic representations.

Figure 22

Figure 23

This image was generated using OptoDesigner software. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 24

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 25

This image was generated using OptoDesigner software. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 26

This image was generated using OptoDesigner software. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 27

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 28

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 29

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 30

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 31

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 32

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 33

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 34

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 35

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 36

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 37

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.

Figure 38

This image was generated using OptoDesigner software and has been modified by Alberto Otero Casado. OptoDesigner is a trademark of Synopsys, Inc.

This image is provided for educational and research purposes only. It is not warranted to be accurate or complete, and Synopsys assumes no liability for its use.

The modifications made to this image consists of adding labels.

For more information about OptoDesigner, please visit the Synopsys website at http://www.synopsys.com.