ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Sophie Cavallini

Photonics Building Blocks in Alumina

Semester Thesis

Master's Degree Programme in Quantum Engineering
Trapped Ion Quantum Information Group
Institute for Quantum Electronics
Swiss Federal Institute of Technology (ETH) Zurich

Supervisors

Gillenhaal Beck

Prof. Dr. Jonathan Home

July, 2023

Abstract

Photonic integrated circuits (PIC) can be exploited to control trapped ions more robustly than free-space optics. Most of the work done in this area used silicon nitride (Si_3N_4), however, it is incompatible with UV wavelengths. Instead, aluminium oxide (Al_2O_3) proves to be a suitable waveguide material. This project focuses on designing and optimizing several integrated optics structures using alumina. All the simulations have been carried out using the finite difference time domain method implemented in Lumerical.
To improve the transmission of a bend, a partial Euler bend can be implemented. Using a portion of an Euler spiral decreases the connection loss thanks to its linearly varying curvature, however, it also increases the bending loss due to the decreased minimum radius of curvature. Therefore it needs to be optimized to achieve the right balance between the two parts. A valuable component for PICs is the power splitter. In this work, it has been implemented using a directional coupler since it allows a design with arbitrary splitting ratios. The designs were conducted using two distinct sets of oxides and waveguide thicknesses.

Contents

Chapter 1

Introduction

1.1 Motivation

Integrated optics can be used to control trapped ions more robustly than free-space optics [1]. In this way, it is possible to achieve higher fidelity quantum operations. Moreover, when increasing the number of ions, controlling a free-space optics setup can become a complex task, and integrated optics devices present themselves as an ideal solution to this problem [2], paving the way to scaling up trapped ions as a quantum information processing platform.

Much work has been done for telecom and visible wavelength integrated optics structures, especially using silicon nitride (Si_3N_4). However, silicon nitride is not a suitable waveguide material for UV wavelengths due to the high material loss. A valid solution is presented by alumina (Al_2O_3) waveguides. In this work, we present the designs and simulations for several integrated photonic components, tailored for UV wavelengths, namely partial Euler bends and arbitrary splitting ratio power splitters.

Improving the bend loss performance for small bend radii is a challenge for Al_2O_3 integrated optics devices. The partial Euler bend has already been demonstrated as a valuable technique to reduce losses when using silicon nitride [3], and using more advanced approaches like using photonic crystals are not suitable for UV light due to its short wavelength. Therefore the first part of this project was focused on optimizing the partial Euler bend for two different waveguides thickness and oxide materials.

Another useful component in photonic integrated circuits is the directional coupler that acts as a power splitter. Usually, different waveguides are far apart throughout the device to inhibit cross-talk, but when it can also be fine-tuned to allow power coupling from one waveguide to another. One of the key properties of the directional coupler is that is possible to design it with arbitrary splitting ratios. This is not possible with components such as MMI (Multi-Mode Interferometer) couplers, where light can only be split evenly between the different outgoing waveguides. Integrated power splitters can be particularly useful in experiments involving trapped ions since they are both need in the

integrated optics below the ion trap, which often require UV laser light and now there are not any commercially available chip- or fiber-integrated beam splitters for UV light. Targeted devices provide unbalanced splitting ratios (99:1, 90:10, 75:25) which provide various experimental capabilities including frequency and power monitoring via pick-off lines of the light before it is directed to the trap.

1.2 Partial Euler bend

The allowed modes of a straight waveguide and a bent waveguide do not correspond, therefore when the fundamental mode in the straight waveguide transitions to the bent part, it will exites higher bend modes [4]. Using an Euler spiral where the curvature varies linearly, should adiabatically convert the straight waveguide mode profile into the bent ones, reducing so the excitations [3] and consequently the connection loss. In this section, the Euler spiral is first presented and afterwards, the partial Euler bend, made by a combination of an Euler spiral and a circle, is described. The mathematical formulation used is the same one as shown in [3], with minor corrections to some formulas.

1.2.1 Euler spiral

In a two-dimensional plane, the Euler spiral, also called clothoid, is described by the following Fresnel integrals:

$$x(s) = \int_0^s \cos\left(\frac{t^2}{2R_0^2}\right) dt, \tag{1.1a}$$

$$y(s) = \int_0^s \sin\left(\frac{t^2}{2R_0^2}\right) dt, \tag{1.1b}$$

where R_0 is a free parameter radius in this formulation, and it is usually set to $1/\sqrt{2}$. The Euler spiral curve is then defined by the function:

$$\mathbf{r}(s) = \begin{pmatrix} x(s) \\ y(s) \end{pmatrix}. \tag{1.2}$$

A useful property of this formulation of the Euler curve is that its length is exactly equal to s. Being the curve regular, it is possible to calculate its curvature as follows:

$$k(s) = \frac{|\mathbf{r}'(s) \times \mathbf{r}''(s)|}{|\mathbf{r}'|^3} = \frac{s}{R_0^2}, \tag{1.3}$$

Thus, the radius of the curvature is given by $R(s) = 1/k(s) = R_0^2/s$. Therefore the radius changes linearly with the curve parameter s, and not abruptly like in a circle bend, as shown in figure 1.2b.

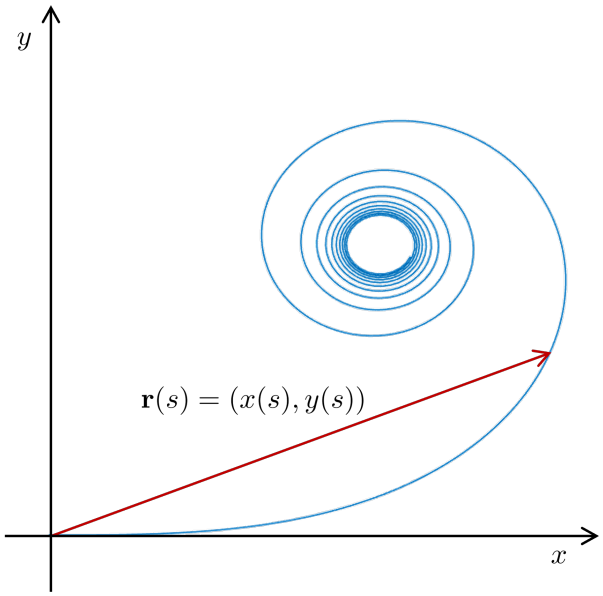


Figure 1.1: Euler spiral curve described by (1.2), with s going from 0 to 500.

It is useful, to perform a change of variable inside the Fresnel integral: $u = t/(R_0\sqrt{2})$ to obtain a direct relationship with the angle α spanned by the curve, obtaining:

$$x(s) = \sqrt{2R_0t_0}\int_0^{s/\sqrt{2R_0t_0}} \cos(u^2)du, \tag{1.4a}$$

$$y(s) = \sqrt{2R_0t_0}\int_0^{s/\sqrt{2R_0t_0}} \sin(u^2)du, \tag{1.4b}$$

where $t_0 = R_0 \cdot \alpha$, this comes directly from the fact that the parametric curve describing a circle has the same property as the Euler spiral, namely that the curve length at a point is exactly equal to the curve parameter at that point. So, to get to the angle α with radius R_0 it is needed to span a length:

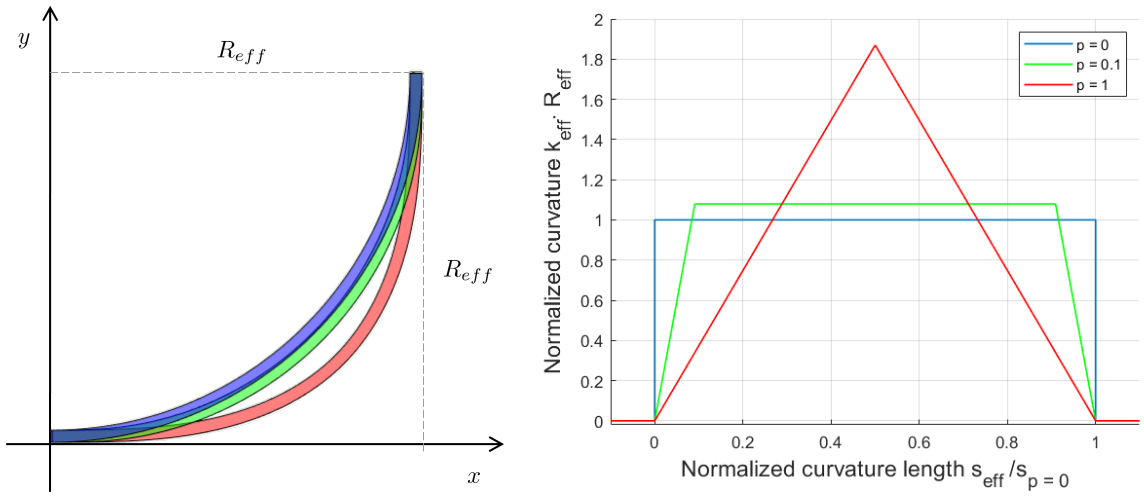
$$s(\alpha) = \sqrt{2R_0t_0(\alpha)} = R_0\sqrt{2\alpha}. \tag{1.5}$$

Thus substituting (1.5) into (1.3), the following relation between the curvature and the angle is obtained:

$$k(\alpha) = \frac{\sqrt{2\alpha}}{R_0}. \tag{1.6}$$

1.2.2 Constructing the Partial Euler Bend

Using a pure Euler bend increases the curvature by 87% with respect to the corresponding circular bend, as it is shown in figure 1.2b. Therefore a combination of both can be used to balance the connection and bending losses. It is possible to design a bend with a desired



(a) **(b)**

Figure 1.2: **(a)** Comparison between a 90° classical circular bend (in blue), a partial Euler bend (in green) with parameter $p = 0.3$, where p is the ratio between the Euler and circular parts, and pure Euler bend (in red), all with radius R_{eff}. **(b)** Normalized curvature along the bend for $p = 0, 0.1, 1$, where $p = 0$, is the classical circular bend and $p = 1$, is the pure Euler bend. Normalization is needed since bends with different parameters p have different effective radii and different curvature lengths. The partial Euler bend has a higher maximum curvature compared to the circular bend. However, the transition to this point is linear, rather than abrupt like for the circular case.

percentage p of α, the bend angle, using the Euler spiral and the remaining part using a circular arc. The idea is to use first an Euler spiral of angle $p/2\alpha$ to get the curvature linearly from 0 to $1/R_p$, then to insert a circular arc with radius R_p and angle $(1-p)\alpha$, and complete the remaining part with another Euler spiral, so that the curvature can return, again linearly, to 0, as shown in figure 1.3. Since the bend is symmetric with respect to the curve perpendicular line positioned at $\alpha/2$, the description of the first half of the bend only is given, it is, then, possible to obtain the second half, simply, by reflection of the first part. The coordinates (x_p, y_p) of the transition point between the Euler and circular part, can easily be obtained using (1.1), i.e. $(x_p, y_p) = (x(s_p), y(s_p))$, where $s_p = R_0\sqrt{p\alpha}$, and the curvature at this point is given by $k_p = \sqrt{p\alpha}/R_0 = 1/R_p$. As can be seen from figure 1.2a the circular part of the partial Euler bend and the circular bend with the same effective radius do not overlap, thus it is important to calculate the displacement of the centre of the circle used in the partial Euler bend.

$$\Delta x = x_p - R_p \sin\left(p\frac{\alpha}{2}\right), \tag{1.7a}$$

$$\Delta y = y_p - R_p \left(1 - \cos\left(p\frac{\alpha}{2}\right)\right). \tag{1.7b}$$

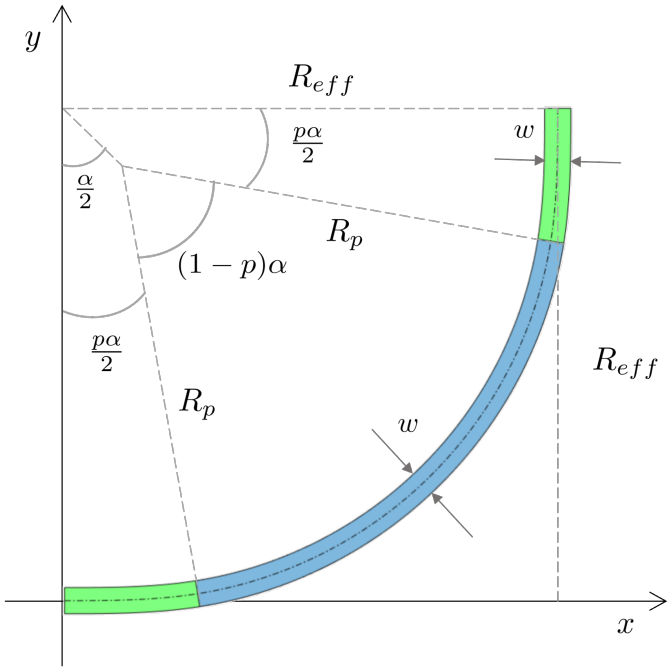


Figure 1.3: Geometry for a 90° partial Euler bend with $p = 0.2$ and radius R_{eff}. The section in blue, which spans an angle $(1-p)\alpha$, is the circular part, while the green sections are the Euler parts, and both of them span an angle $p\alpha/2$. The width is constant in the entire bend and equal to w.

While the total length of the bend is given by: $s_0 = 2s_p + R_p\alpha(1-p)$. So, the coordinates of the midpoint are:

$$x_{bend}\left(\frac{s_0}{2}\right) = R_p \cdot \sin\left(\frac{s_0/2 - s_p}{R_p} + p\frac{\alpha}{2}\right) + \Delta x, \tag{1.8a}$$

$$y_{bend}\left(\frac{s_0}{2}\right) = R_p \cdot \left(1 - \cos\left(\frac{s_0/2 - s_p}{R_p} + p\frac{\alpha}{2}\right)\right) + \Delta y. \tag{1.8b}$$

To this point all the calculations have been done without considering the effective radius R_{eff}, which is the desired radius of the bend. To design a bend with radius R_{eff}, it is useful to introduce the rescaling factor η:

$$\eta = \frac{R_{eff}}{y_{bend}(s_0/2) + x_{bend}(s_0/2)/\tan(\alpha/2)}. \tag{1.9}$$

The parametric equations describing the different parts of the rescaled bend are:

$$x_{bend}(s) = \begin{cases} \eta \cdot x(s/\eta) & \text{for } 0 \le s \le \eta \cdot s_p \\ \eta\left[R_p \sin\left(\frac{s/\eta - s_p}{R_p} + p\frac{\alpha}{2}\right) + \Delta x\right] & \text{for } \eta \cdot s_p \le s \le \eta \cdot \frac{s_0}{2} \end{cases}, \tag{1.10a}$$

$$y_{bend}(s) = \begin{cases} \eta \cdot y(s/\eta) & \text{for } 0 \leq s \leq \eta \cdot s_p \\ \eta \left\{ R_p \left[1 - \cos\left(\frac{s/\eta - s_p}{R_p} + p\frac{\alpha}{2} \right) \right] + \Delta x \right\} & \text{for } \eta \cdot s_p \leq s \leq \eta \cdot \frac{s_0}{2} \end{cases} . \quad (1.10b)$$

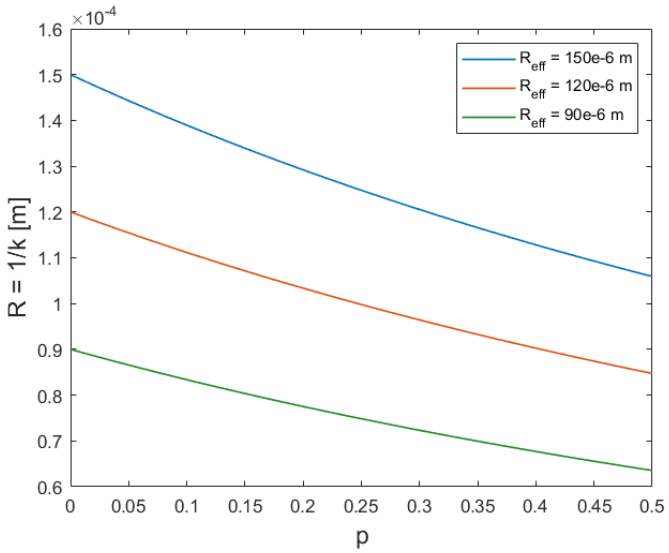


Figure 1.4: Minimum radii of curvature for three different effective radius: 90 µm, 120 µm, 150 µm, for a range of p going from 0 to 0.5. The points at $p = 0$ indicate the radius of curvature for the corresponding standard circular bends, in this case, the radius of curvature equals the effective radius of the circle.

These coordinates are the ones of the midline of the bend, the dotted line shown in figure 1.3. The coordinates of the inner and outer parts of the bend can be obtained by taking the perpendicular line to each point of the midline and then taking the two points which are distant $w/2$ to the corresponding midpoint, where w is the width of the waveguide. The MATLAB code used to generate this geometry can be found in Appendix B.1.

It is important to optimize the bend for p, the ratio of the bend that is a clothoid, since using an Euler bend decreases the connection loss between the straight waveguide and the bent waveguide [5], thanks to the linear change of curvature, but it also increases its maximum curvature, as shown in figures 1.2b and 1.4, thus increasing the radiation loss. Therefore the optimal balance between the Euler and circular parts needs to be found to minimize the overall bending loss.

1.3 Arbitrary splitting ratio power splitter

A standard directional coupler is symmetric [6] and two S-bends are used to bring the waveguides closer together, so to increase the cross-talk, and another two S-bends are

needed to bring the waveguides apart after the coupling region. The main difference between a standard directional coupler and the adiabatic directional coupler, here presented, is that for the former the width of the waveguides is constant throughout the structure [7]. By changing the widths before and after the S-bends it is possible to adiabatic transition light in the device [8]. The work done here took inspiration from [8], where an adiabatic directional coupler was designed for silicon waveguides at telecom wavelengths.

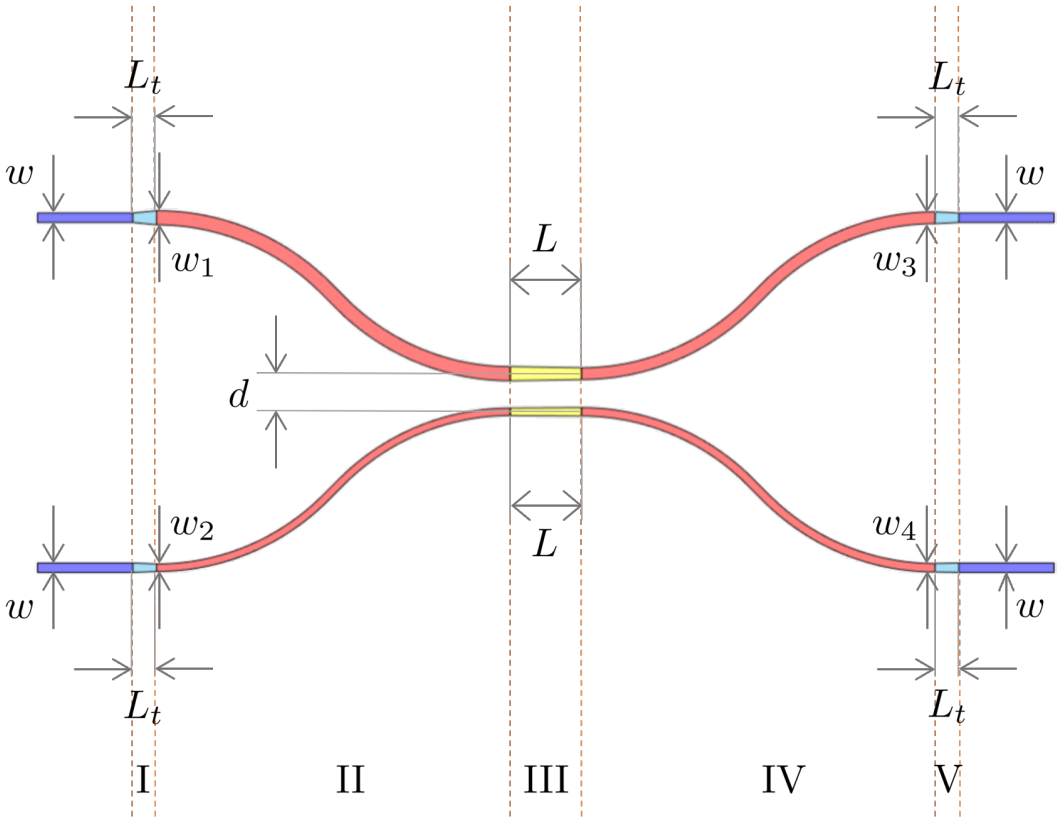


Figure 1.5: Schematic representation of the adiabatic directional coupler. Depicted in blue are the standard straight waveguides, while the light blue ones are the tapers in and out of the structure, in red are represented the S-bends, and finally in yellow are the tapered central coupling waveguides.

As shown in 1.5, the adiabatic directional coupler can be divided into five different regions:

I. Both waveguides are tapered to reach the desired wavelength at the input of the S-bends. The length of the waveguides is L_t for both. Both the top and bottom waveguides, beginning at width w, are tapered over length L_t to widths of w_1 and w_2, respectively.

II. Two adiabatic S-bends, with constant width, are used to get the waveguides close to each other.

III. This is the coupling region of the device, where, instead, of having two straight parallel

waveguides of constant width, the waveguides are tapered to increase the coupling rate. The coupling length is L and the separation between the midlines of the two waveguides is given by d.

IV. As in Region II two adiabatic S-bends are used to separate the waveguides.

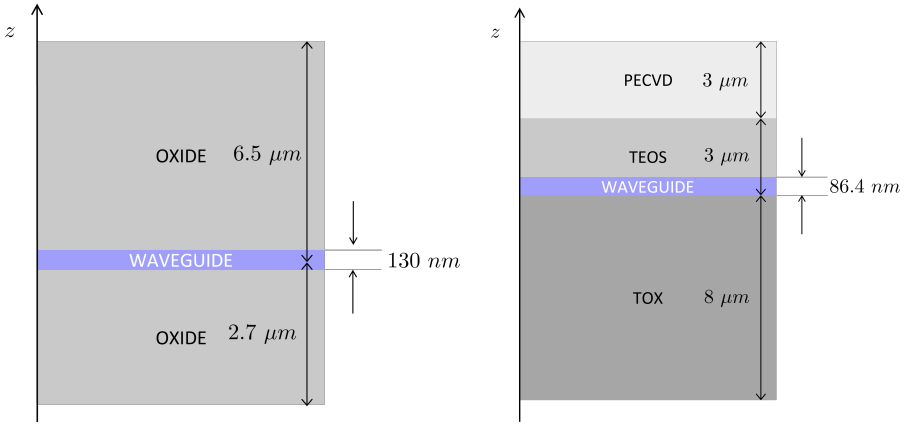
V. The waveguides are again being tapered, here, to get them to the original width. So, the widths go from w_3 to w and from w_4 to w, respectively for the top and bottom waveguide.

It is then important to simulate and optimize each region before optimizing the whole structure. To assure that the structure is truly adiabatic the most critical parts are the S-bends since if not taken care of a significant amount of power will be lost there.

Chapter 2

Results and Discussion

This chapter focuses on the simulation and optimization of the photonic integrated circuit components first described in Chapter 1. Two different material sets have been considered. One assumes a single oxide material above and below the waveguide layer, with 6.5 µm of it above the waveguide and 2.7 µm below. This corresponds to the material stack of a new generation of ion trap chips fabricated by LioniX. The second one is composed of three different types of oxides, namely: TOX (Thermal Oxide), TEOS (tetraethoxysilane) oxide and PECVD (Plasma Enhanced Chemical Vapour Deposited) oxide. However, the most relevant difference between the two is the waveguide thickness of 130 nm and 86.4 nm, respectively. The vertical stacks of the two setups are shown in figure 2.1.
All the simulations and fine-tuned optimizations have been first conducted for the LioniX setup, using the techniques presented in appendix A. The second setup was developed in the framework of the ALUVia project, and due to tight time constraints it was not possible to run all the optimization, therefore it was assumed that the results obtained for the first setup are still valuable for this layout too, but the correctness of this assumption should be verified at a later moment. Therefore the results for this setup are presented in a dedicated section (section 2.3).
Everything has been optimized for $\lambda = 397$ nm, being it the central wavelength, and then the results were checked for 375 nm and 423 nm, being these the relevant UV wavelengths for calcium trapped ions. Part of the calculations were carried out on the ETH Zurich Euler cluster.
As discussed in appendix A, when dealing with large simulation regions compared to the wavelength the computational cost for the full FTDT simulation can be prohibitive. The components presented here are no exception, since the RAM required for the partial Euler bend can be up to 1 TB, and this far exceeds even the memory available per user on the Euler cluster. Therefore, only the varFDTD solver has been used. Even though looking at the electric field mode profile slice, shown in figure 2.2, we see that a considerable amount of the mode lies outside the waveguide, especially for the ALUVia case. Therefore other full 3D simulation techniques, such as the beam propagation method or the full FDTD



(a) LioniX **(b)** ALUVia

Figure 2.1: Oxide vertical stack for the two considered setups.

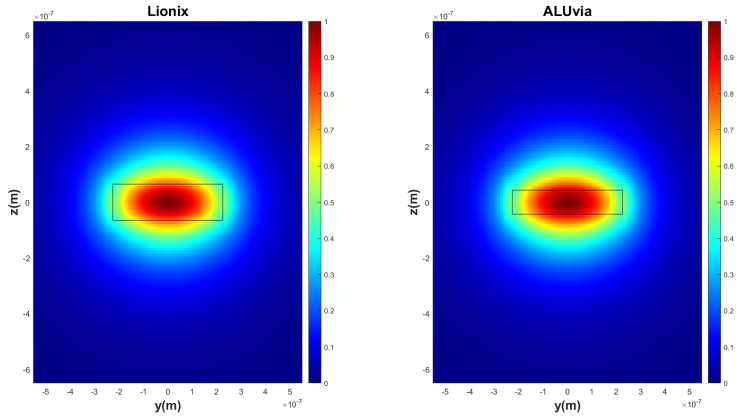


Figure 2.2: Electric field mode profile slice of the source for the two different oxide sets.

with the scattering matrix approach, may be considered for future works.

2.1 Partial Euler bend

Increasing the radius of the bend trivially reduces the bend loss. The simulations have been conducted for a range going from 100 µm to 150 µm. Considering the waveguide width as shown by figure 2.2 for $d = 450$ nm gives good lateral mode confinement, and increasing it even more would allow higher order modes inside the waveguide. An example of the electric field monitor is shown in figure 2.3, for $R = 120$ µm and $p = 0.1$.
The optimization for the parameter p has been conducted sweeping it from 0 to 0.2 using 21 test points, for the radii range considered before, the best radius was identified by looking at the minimum of the loss. The results are shown in figure 2.4, while the loss, for a selection of radii, is shown in figure 2.5. For $R \geq 120$ µm, the optimal p parameter lies between 0.09 and 0.11, in particular for $R = 150$ µm it is equal to 0.0995, and the

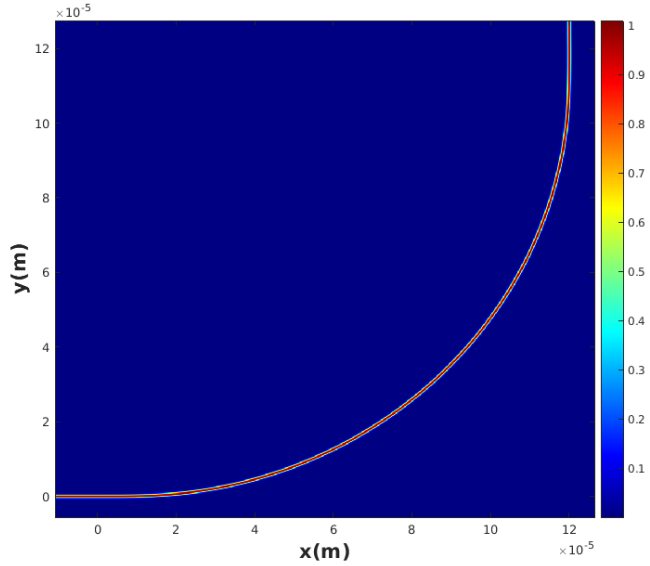


Figure 2.3: Electric field in plane 2D monitor for a partial Euler bend with radius $R = 120\,\mu m$, parameter $p = 0.1$, waveguide width $d = 450\,nm$.

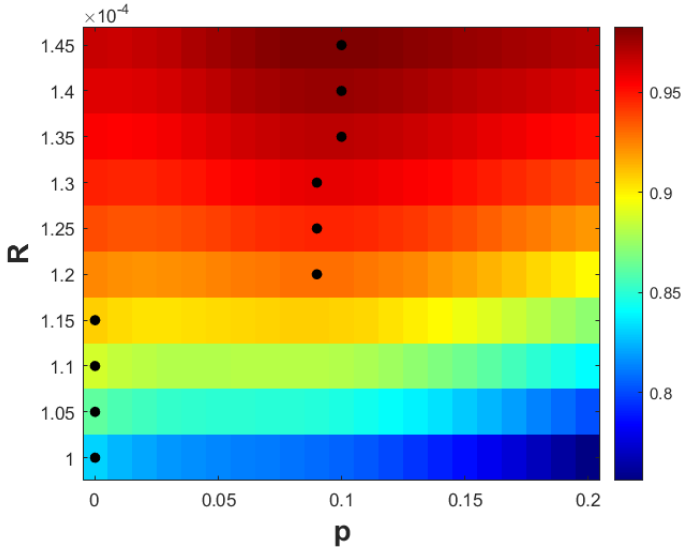


Figure 2.4: Partial Euler bend transmission for the parameter p sweep from 0 to 0.2, for a range of radius going from 100 µm to 145 µm. The black dots indicate the points with maximum transmission loss.

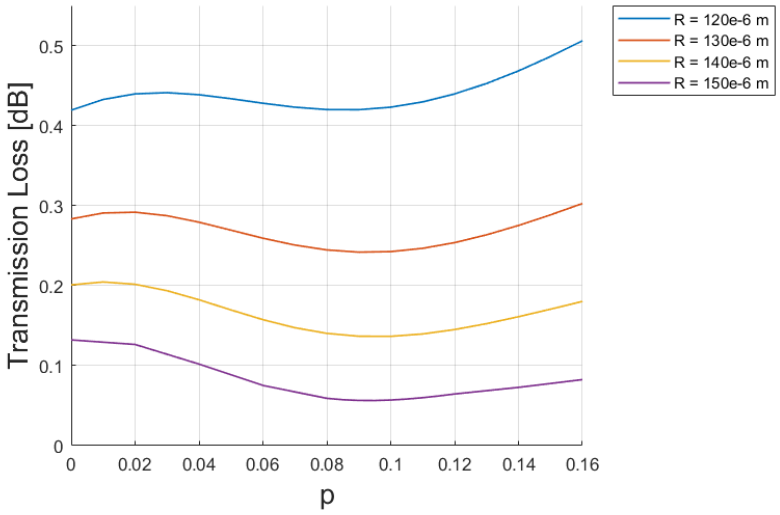


Figure 2.5: Partial Euler bend loss for a selection of radii. For $R = 150\,\mu m$ the sweeping points around the optimal point are double the ones for the other radii.

λ [µm]	T [-]	Loss [dB]
375	0.998371	0.0071
397	0.999516	0.0021
423	0.998700	0.0056

Table 2.1: Partial Euler bend transmission T and loss for radius $R = 150\,\mu m$, angle $\alpha = \pi/2$ and parameter $p = 0.0995$. The loss for $\lambda = 397\,nm$ is different from the one shown in figure 2.5 at $p = 0.0995$, due to different meshing accuracies.

transmissions for different wavelengths are shown in table 2.1. While for $R < 120\,\mu m$, from figure 2.4 it is clear that there is no benefit in using the Euler bend. This can be interpreted as the reduced connection loss is not enough to counterbalance the increased bend loss due to a lower effective radius.

2.2 Arbitrary splitting ratio power splitter

Figure 1.5 shows that the adiabatic directional coupler (ADC) can be divided into 5 different regions: the input tapers and S-bends, the central coupling region and the output tapers and S-bends. However, simulation-wise it is better to first identify and optimize the sub-structures that are relevant for the total transmission, instead of simulating each region individually. These components are the taper, S-bend and parallel waveguides. After this has been done, the central coupling region can be tuned to obtain the desired splitting ratios. As for the partial Euler bend, everything has been optimized for $\lambda = 397\,nm$, and then checked for 375 nm and 423 nm.

2.2.1 Taper

For the taper optimization, we aim to determine the smallest length L_t where it is working adiabatically across a range of output width w_1 going from 250 nm to 650 nm. The input width is kept fixed at $w = 450$ nm since this is the chosen width for the rest of the device. Thus, the structure can be simulated varying w_1 and L_t, as shown in 2.6. The results are

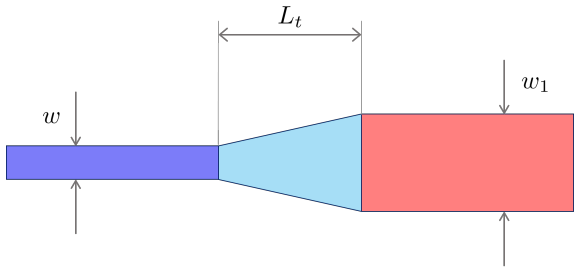
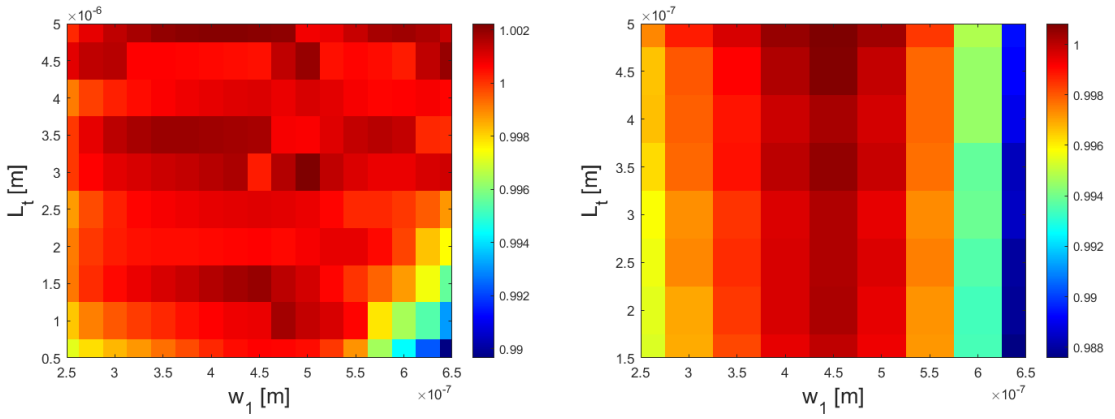


Figure 2.6: Taper simulation setup, w is fixed and w_1 and L_t are swept.

shown in figure 2.7. From 2.7b, it is clear that, as expected, the transmission is higher for $w_1 = 450$ nm, since in this case, being the output width the same as the input one, it is just a normal straight waveguide. In general, the transmission is excellent for all considered values, but looking at 2.7a to ensure a transmission higher than 0.99 for the entire range of w_1, the length L_t should be at least 1 µm.

2.2.2 Parallel waveguides

Even though the simulation of two parallel waveguides may be considered trivial it is still important to verify that before and after the ADC the waveguides are distant enough to be sure that they are not coupled. Moreover, it is also interesting to have a rough estimation of the distance between the waveguides where the crosstalk becomes important. The device can be simulated as shown in figure 2.8, where the length of the waveguides remains fixed, while the position of the field monitor, where the transmission is being measured, is varied. Therefore, the distance L between the source and the monitor and the distance d between the centres of the waveguides are swept. The results for the 2D sweep of L going from 1 to 10 µm and Δd, also, going 1 to 10 µm, where the true separation between the centres of the waveguides is defined as $d = \Delta d + w$ as shown in 2.9. The transmissions for the top and bottom waveguides are shown in figure 2.9. As expected better results are obtained for larger separations, while the results worsen for large L. As it will be seen in section 2.2.3 the S-bend we chose has radius R equal to 70 µm and angle α equal to $\pi/12$, thus using (1.10) we get that the separation between the input waveguides is 14.124 µm, larger than the ones considered here. Nevertheless, we can look at the results for $L = 10$ µm being the top transmission obtained here as a lower bound for our case, so the transmission is always larger than 0.98 for the top waveguide and less than 2% for the bottom one, hence there won't be any relevant crosstalk before and after the coupler. The considerations made for



(a) **(b)**

Figure 2.7: Taper transmission for a 2D sweep of w_1, output width, and L_t, taper length. In (b) a narrower range of values has been used to allow a higher meshing precision. Trivially the higher transmission is reached for $w_1 = w = 450\,\text{nm}$, overall the transmission is above 0.99 for $L_t \geq 1\,\mu\text{m}$ for all values of w_1 considered.

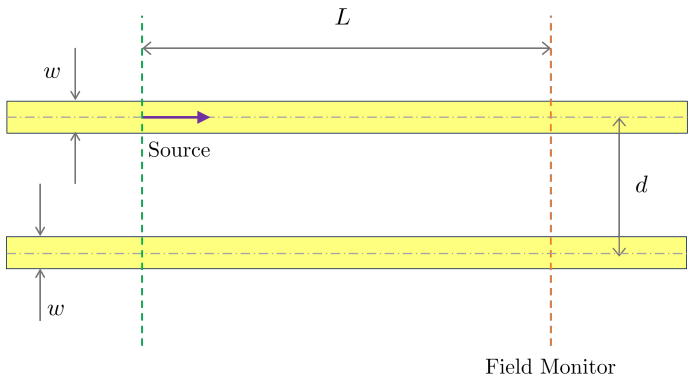
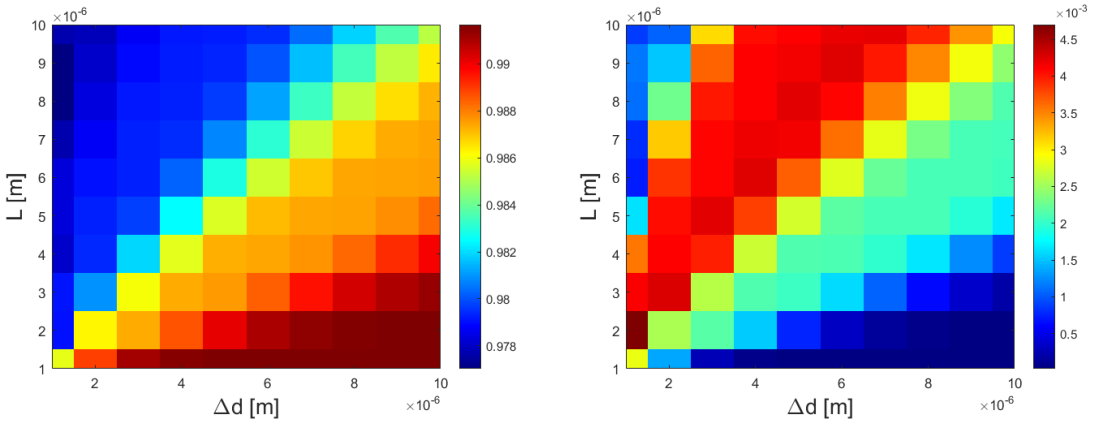


Figure 2.8: Parallel waveguides simulation setup, w is fixed, and L and d are swept.

$\lambda = 397\,\text{nm}$ are also valuable for the other two wavelengths considered, as shown in 2.10 where L goes from $1\,\mu\text{m}$ to $10\,\mu\text{m}$, and d is fixed at $10\,\mu\text{m}$. To guarantee that there is no crosstalk between the waveguides additional S-bends with large radii can be used to separate the waveguides further apart.

2.2.3 S-Bend

The S-bend is formed by a circular bend followed by another circular bend with opposite concavity, and to limit the losses a straight waveguide can be inserted in between the two bends (the MATLAB code used to generate the geometry can be found in Appendix B.2). In this context, the bend angle α is not fixed at 90°, therefore together with the bend radius R, it can be optimized to achieve low transmission. Therefore the radius of the bend is limited up to $70\,\mu\text{m}$ to keep a compact design. Being all the considered radii smaller than $120\,\mu\text{m}$ there is no benefit in using the partial Euler bend, as already shown in section 2.1.



(a) Top waveguide **(b)** Bottom waveguide

Figure 2.9: Parallel waveguides transmission for a 2D sweep of L, the distance between source and field monitor, and Δd, defined such that the separation between the centres of the waveguides is $d = \Delta d + w$, where w is the waveguide width.

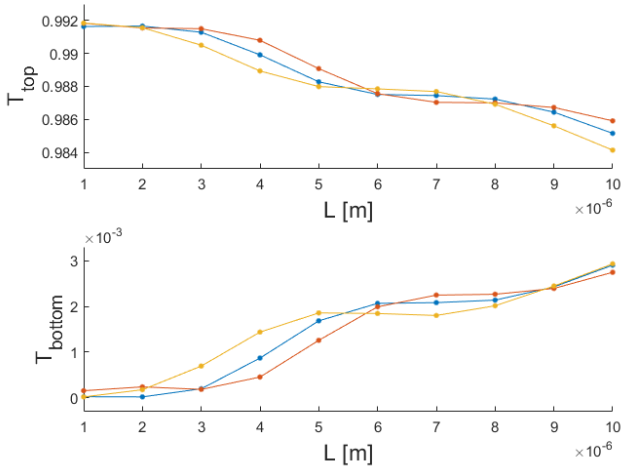


Figure 2.10: Transmissions for the top and bottom parallel straight waveguides for $d = 10\,\mu\text{m}$, in orange $\lambda = 375\,\text{nm}$, in blue $\lambda = 397\,\text{nm}$, and in yellow $\lambda = 423\,\text{nm}$.

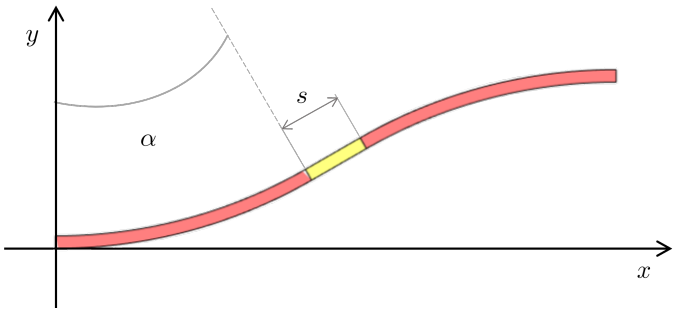


Figure 2.11: S-bend simulation setup, where s, length of the middle straight section, α, bend angle, and R, bend radius (not shown in the picture), are optimized to minimize the bend loss.

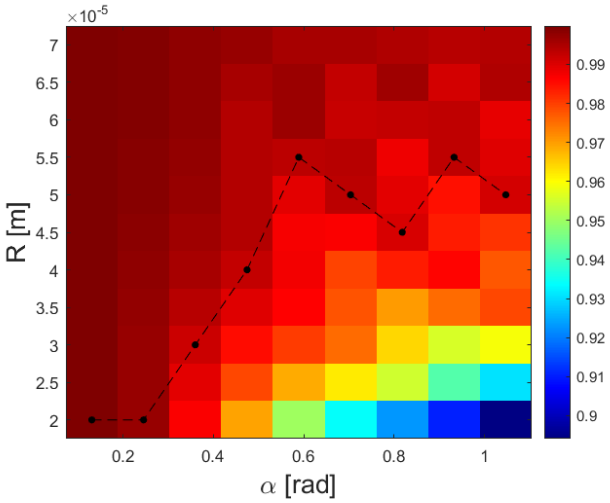


Figure 2.12: S-bend transmission for a 2D sweep of α and R, respectively bend angle and radius. To get a total transmission above 0.99, a pair (α, R) positioned above the black dotted line should be chosen.

The simulation setup is shown in figure 2.15.

The results obtained as a function of R going from 20 µm to 70 µm, and of α from $\pi/24$ to $\pi/3$, at $\lambda = 397$ nm, are shown in figure 2.12, for the other wavelengths the results are similar. Thus, to get a total transmission above 0.99, a pair (α, R) positioned above the black dotted line should be chosen. To ensure the realization of an adiabatic bend, an S-Bend with $R = 70$ µm and $\alpha = \pi/12$ has been chosen. Now, that these parameters have been fixed, the length s of the straight waveguide in the centre can be optimized. Sweeping s from 0 to 8 µm, the transmissions shown in figure 2.13 are obtained. The maximum is reached at $s = 4.5$ µm for $\lambda = 397, 423$nm. The transmission results for the three different wavelengths, with this tern of parameters $(R, \alpha, s) = (70\,\mu\text{m}, \pi/12, 4.5\,\mu\text{m})$ are shown in table 2.2, while the in-plane electric field is shown in figure 2.14.

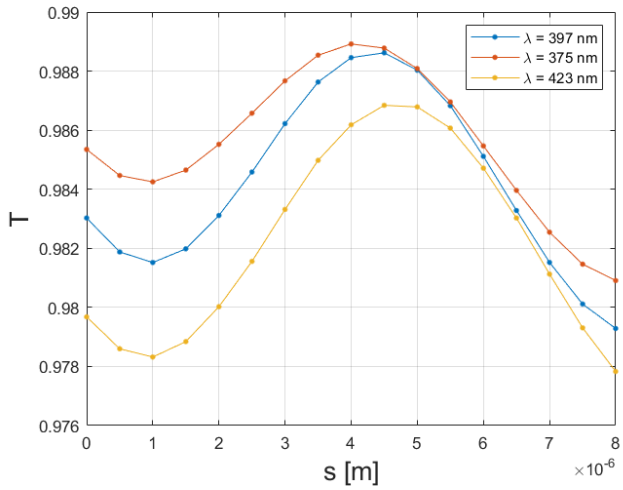


Figure 2.13: S-bend transmission with $R = 70\,\mu m$ and $\alpha = \pi/12$, for s going from 0 to $8\,\mu m$. The maximum is reached at $s = 4.5\,\mu m$ for $\lambda = 397, 423 nm$, and at $5\,\mu m$ for $\lambda = 423\,\mu m$

λ [µm]	T [-]	Loss [dB]
375	0.995156	0.0211
397	0.99846	0.0067
423	0.996664	0.0145

Table 2.2: S-bend transmission T and loss for radius $R = 70\,\mu m$, angle $\alpha = \pi/12$ and middle straight waveguide length $s = 4.5\,\mu m$.

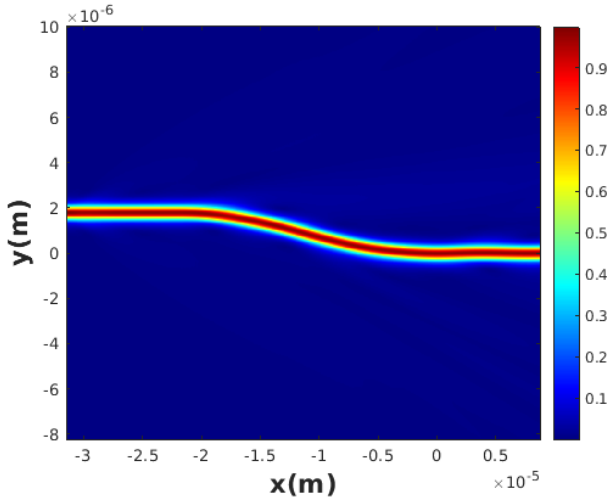


Figure 2.14: Electric field in plane 2D monitor for an S-bend with $R = 70\,\mu m$, $\alpha = \pi/12$, and $s = 4.5\,\mu m$.

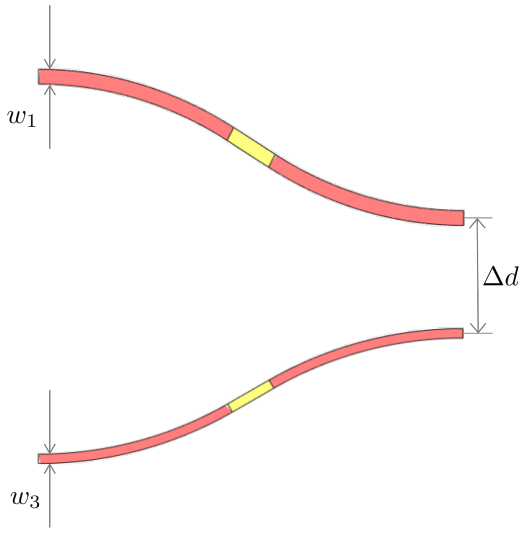


Figure 2.15: Two S-bends simulation setup, where Δw that is defined such that the top waveguide has width $w_1 = 450\,\text{nm} + \Delta w$ and the bottom waveguide has width $w_3 = 450\,\text{nm}$. The single S-bend specifications are the ones illustrated in section 2.2.3.

2.2.4 Two S-bends

To further improve the total transmission the waveguide width of the S-bends can be improved. As already discussed in section 2.1, increasing the width implies that the mode is more confined inside the waveguide. If the lost power in one waveguide gets confined in the other one it does not affect the total component transmission. Therefore, to analyze how changing the width influences the transmission the simulation region should include both S-bends. The simulation setup is shown in figure 2.15.

As shown in figure 2.16, the total transmission increases with the difference in waveguide width Δw. Therefore it is recommended to choose Δw greater than 0.

2.2.5 Complete device simulation

The parameters that characterize the coupling region are the widths of the waveguide w_1, w_2, w_3, w_4, the length L and the separation d between the two waveguides. However, there is no need to optimize all of them in this section, since the computational cost for a multiple-dimensional sweep with as many parameters will be huge, and we already have obtained some information regarding the widths from section 2.2.4. Therefore it has been chosen to have $w_1 = 500\,\text{nm}$, $w_2 = 300\,\text{nm}$, $w_3 = 425\,\text{nm}$ and $w_4 = 500\,\text{nm}$. Now the focus of the optimization is to first find a suitable L and then to fine-tune d. For highly unbalanced splitting ratios, i.e. 99:01 and 90:10 it is better to use short coupling length since, otherwise, the exchanged power would be too much, so $L = 2\,\mu\text{m}$ is effective. But for 75:25, this short length means that the coupling constant needs to be high, in fact so high that the distance d between the centre of the waveguides would be so short that the waveguides would have to overlap, thus a larger length has been used: $L = 10\,\mu\text{m}$. Now,

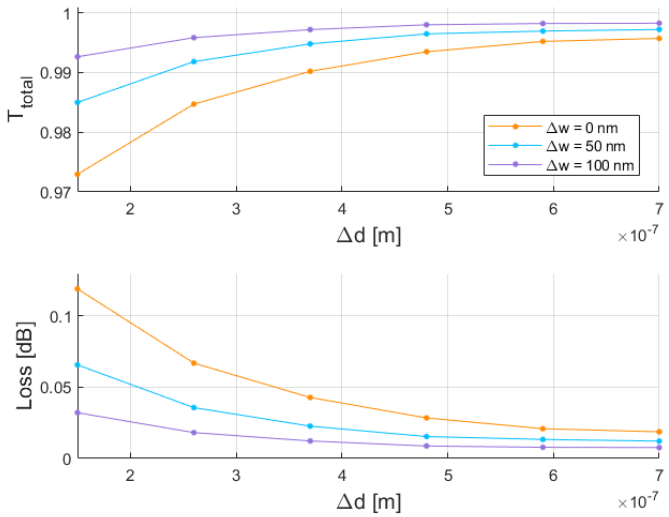
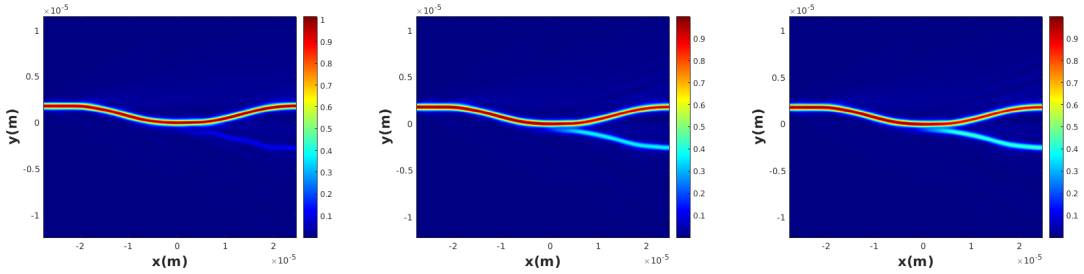


Figure 2.16: Two S-bends transmission as a function of of the waveguide spacing Δd for $\lambda = 397\,\text{nm}$ at three different Δw.



(a) SR = 99:01 **(b)** SR = 90:10 **(c)** SR = 75:25

Figure 2.17: Electric field in plane 2D monitor for different splitting ratios at $\lambda = 397\,\text{nm}$.

by sweeping d, or better Δd, which has been defined as $\Delta d = d - \max[(w_1, w_2, w_3, w_4)]$, we can find the optimal value for each splitting ratio. In conclusion, we have been able to design low-loss power splitters with the desired splitting ratios.

2.2.6 Standard directional coupler

We can now compare the results obtained for the adiabatic directional coupler with the standard version. Overall, for the same coupling length to obtain the same splitting ratio the distance between the waveguides needs to be smaller. This can be easily interpreted by looking at figure 2.7, where it showed that when a waveguide is tapered part of the mode gets squeezed out, and can so be confined in the other one. This is also why it is important that not only there is a difference in width between the corresponding waveguides in the same region as shown in 2.2.4, but also there is a difference between the input and output of the coupling region for the same waveguide. By reformulating this result we obtain that for the same waveguide separation and splitting ratio the coupling length for the adiabatic

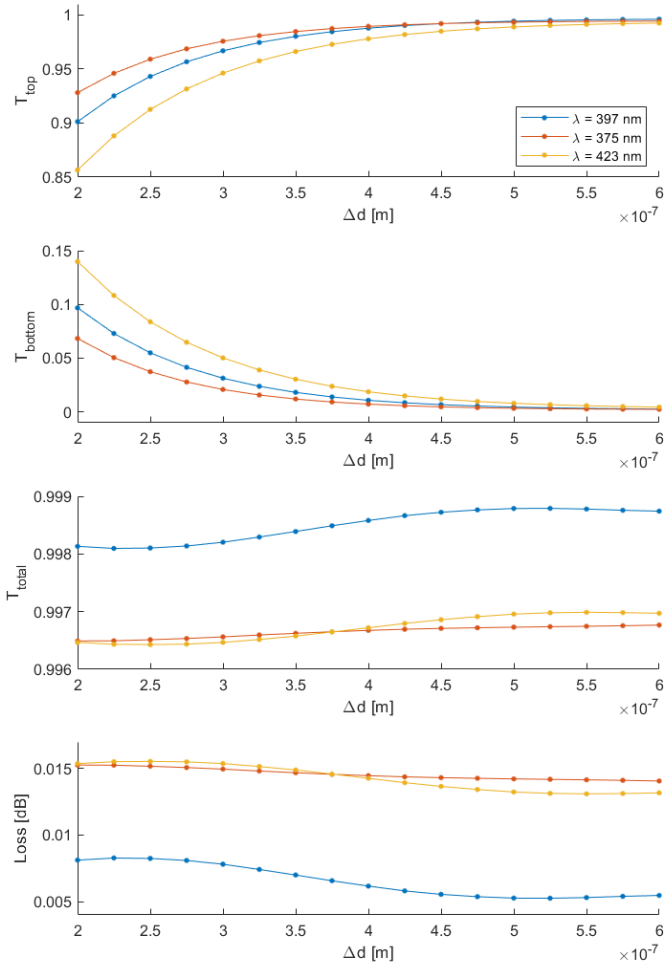


Figure 2.18: ADC transmission as a function of the waveguides separations Δd, for three different wavelengths, at $L = 2\,\mu\text{m}$.

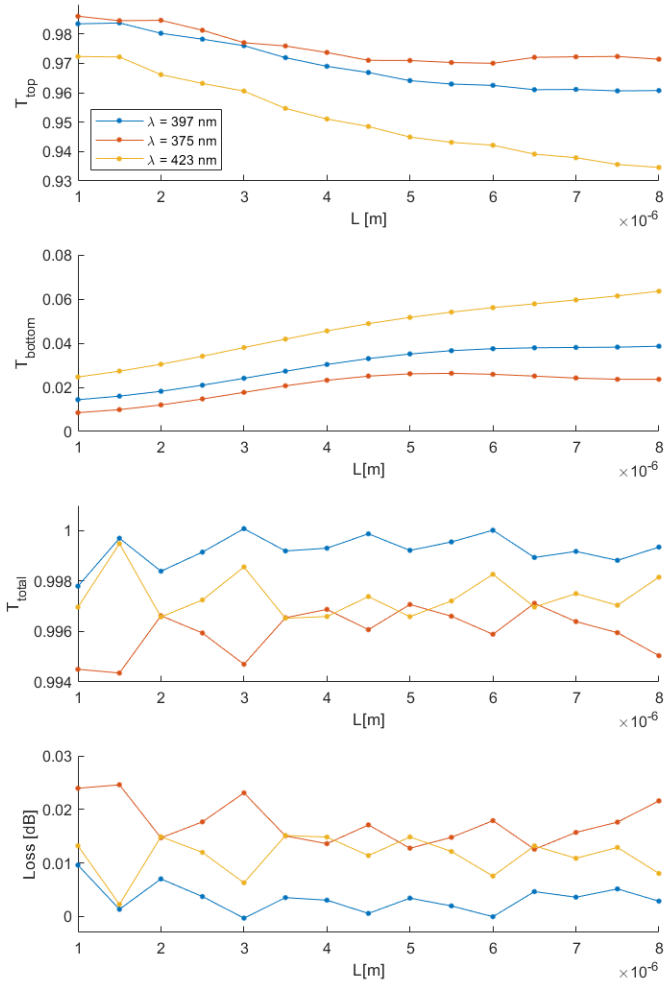


Figure 2.19: ADC transmission as function of coupling length L, for the three different wavelengths, at $d = 900\,\text{nm}$.

λ [µm]	T_{top}	T_{bottom}	T_{total}	Loss [dB]
375	0.993910	0.006090	0.996263	0.0163
397	0.9903581	0.009642	0.999585	0.0018
423	0.985618	0.014382	0.997734	0.0099

(a) SR = 99:01, with $L = 2\,\mu\text{m}$ and $d = 1200\,\text{nm}$.

λ [µm]	T_{top}	T_{bottom}	T_{total}	Loss [dB]
375	0.928361	0.071639	0.996007	0.0173
397	0.898615	0.101385	0.999253	0.0032
423	0.856439	0.143561	0.997267	0.0119

(b) SR = 90:10, with $L = 2\,\mu\text{m}$ and $d = 970\,\text{nm}$.

λ [µm]	T_{top}	T_{bottom}	T_{total}	Loss [dB]
375	0.816589	0.183411	0.996291	0.0161
397	0.749800	0.250200	0.999054	0.0041
423	0.656136	0.343864	0.995856	0.0180

(c) SR = 75:25, with $L = 10\,\mu\text{m}$ and $d = 740\,\text{nm}$.

Table 2.3: Transimission of the ADC for different splitting ratios.

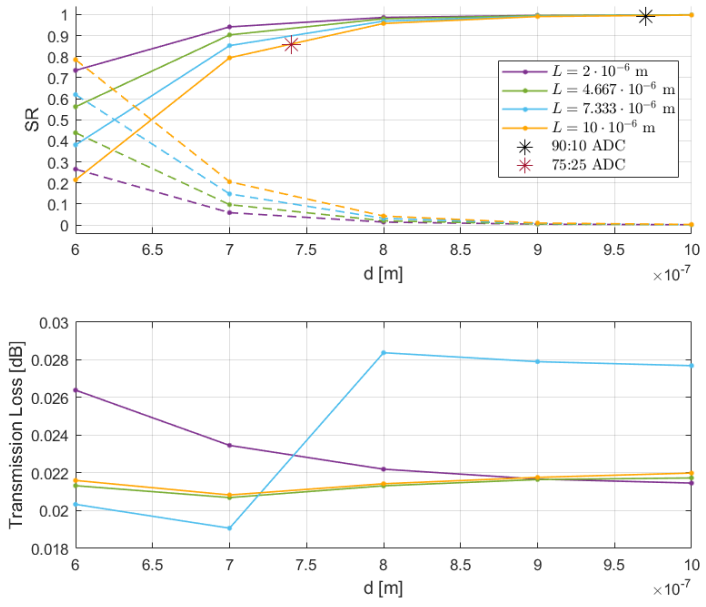


Figure 2.20: Standard directional coupler splitting ratio and loss as a function of the waveguide separation d, for different coupling lengths L at $\lambda = 397\,\text{nm}$. The solid lines indicate the transmission of the top waveguide, while the dotted lines of the bottom waveguide. The two stars indicate the point (d, L) for the corresponding adiabatic directional coupler.

λ [µm]	T_{top}	T_{bottom}	T_{total}	ΔT_{total}
375	0.993941	0.006059	0.996393	1.30×10^{-4}
397	0.990398	0.009602	0.999278	3.06×10^{-4}
423	0.985693	0.014307	0.997195	5.39×10^{-4}

Table 2.4: Transmission of the ADC with SR = 99:01 and mesh accuracy 8. In the last column, it is shown the difference with the total transmission for mesh accuracy 6 (table 2.3a).

version is smaller than for the standard version. Now looking at the losses, shown in the bottom plot of figure 2.20, and comparing them with the ones for $\lambda = 397\,\text{nm}$ in table 2.3, the results obtained for the former are better than ones obtained for the latter. Therefore using the adiabatic directional coupler has achieved the objective of reducing the total loss with respect to the standard directional coupler. It also allows us to make more compact designs.

2.2.7 Numerical errors

All the previous simulations have been done with mesh accuracy 6. To check that this is sufficient we can try to run one of the previous simulations with a higher mesh accuracy and compare the results. For example, the results for the 99:01 power splitter with mesh accuracy 8, the highest possible in Lumerical, are shown in table 2.4, and they are in good agreement with numbers shown in table 2.3a. Therefore, the results shown in table 2.3 are accurate enough.

2.3 ALUVia

As already discussed at the beginning of the current chapter, for the ALUVia project different oxides were used.

It is computationally expensive to simulate the partial Euler bend because the dimensions of the structure are large compared to the wavelength [9]. Therefore the same parameter p has been used, so equal to 0.1, to be certain that is still the optimal ratio the sweep for p should also be run for these new differnt characteristics of the structure. It has been decided to test the fabrication of the bends with the maximum radius here considered, since for this value the results are the best one, so with radius equal to 150 µm. The results are shown in table 2.5.

The central coupling region has been optimized in the same manner as in section 2.2.5 and the optimal values are shown in table 2.6.

For the 99:01, 90:10 the ADC have dimensions comparable to LioniX ones, while for SR = 75 : 25, the ALUVia structure is larger. While regarding the total loss for the first setup are lower, but this can be easily explained since in that case all the subcomponents

λ [µm]	T	Loss [dB]
375	0.997977	0.0088
397	0.998925	0.0047
423	0.999067	0.0041

(a) $w = 440\,\text{nm}$

λ [µm]	T	Loss [dB]
375	0.997128	0.0125
397	0.998826	0.0051
423	0.998952	0.0046

(b) $w = 450\,\text{nm}$

λ [µm]	T	Loss [dB]
375	0.997299	0.0117
397	0.998750	0.0054
423	0.999028	0.0042

(c) $w = 460\,\text{nm}$

Table 2.5: Partial Euler bend transmission for different wavelengths and waveguide widths.

λ [µm]	T_{top}	T_{bottom}	T_{total}	Loss [dB]
375	0.987473	0.0125273	0.995516	0.0195
397	0.991083	0.00891666	0.997120	0.0125
423	0.993612	0.00638772	0.994768	0.0228

(a) SR = 99:01, with $L = 1\,\mu\text{m}$ and $\Delta d = 2000\,\text{nm}$.

λ [µm]	T_{top}	T_{bottom}	T_{total}	Loss [dB]
375	0.929014	0.0709860	0.992686	0.0319
397	0.904214	0.0957862	0.995261	0.0206
423	0.872085	0.127915	0.990651	0.0408

(b) SR = 90:10, with $L = 2\,\mu\text{m}$ and $\Delta d = 350\,\text{nm}$.

λ [µm]	T_{top}	T_{bottom}	T_{total}	Loss [dB]
375	0.816233	0.183767	0.993010	0.0305
397	0.746200	0.253800	0.992241	0.0338
423	0.661563	0.338437	0.987781	0.0534

(c) SR = 75:25, with $L = 21\,\mu\text{m}$ and $\Delta d = 325\,\text{nm}$.

Table 2.6: Transimission of the ADC for different splitting ratios.

were optimized, while for the current case, only the coupling region was optimised.

The structures designed in this section are currently being fabricated (as of July 2023), and they will afterwards be characterized to test whether the results obtained through the simulations actually match their physical realization.

Chapter 3

Conclusion and Outlook

The goal of this semester project was to design and optimize different photonic components in alumina for UV wavelength, in particular for $\lambda = 375,\ 397,\ 423\,\text{nm}$. To simulate the structures the finite difference time domain method was used with the software Lumerical, and in particular its solver varFDTD, where the three space dimensions are collapsed to two to decrease the computational cost.

In the first part of the project, to improve the transmission of a 90° bend it was investigated the implementation of a partial Euler bend, made by a combination of Euler and circular bends. Using a portion of an Euler spiral decreases the connection loss thanks to its linearly varying curvature, but it also increases the bending loss due to the decreased minimum radius of curvature. Therefore the right balance between the Euler and circular part had be found to minimize the overall loss. The main result obtained was that for radii below 120 µm using a partial Euler bend does not give any advantage compared to a standard circular bend, while for R in the range from 120 to 150 µm the optimal parameter p, the ratio between the Euler and circular part of the bend, is circa 0.1.

Another useful component in integrated photonics is the power splitter. This can be implemented through different structures, each one with its own characteristic properties. The chosen structure is the directional coupler, its main property is the possibility to design it with arbitrary splitting ratios. To minimize the total transmission loss its adiabatic version was used. At first, the S-bends and tapers, sub-components of the structure, were optimized to reduce their loss and to study the cross-talk two straight parallel waveguides were simulated for different separations and lengths. Afterwards, the entire device was simulated to optimize the waveguide length and distance in the coupling region to achieve the desired splitting ratios, namely 99:01, 90:10 and 75:25, for $\lambda = 397\,\text{nm}$.

Two different material sets were used, one with only one type of oxide and with a waveguide thickness equal to 130 nm, while the other one used three different types of oxides and a thinner waveguide ($t = 86.4\,\text{nm}$), and was developed in the framework of the ALUVia project. All the optimizations were conducted for the first one, and the results were used also for the second one, with the exception of the optimizations regarding the

splitting ratios, which were also performed for this other setup.

The next natural step is to fabricate and characterize these structures to verify that the results obtained through the simulations actually match the physical devices. Fabrication is currently in process (as of July 2023) for the ALUVia structures. Future structures of interest for alumina photonic integrated circuits can include power coupler and waveguide crossing. For the former, the implementation should be straightforward by using the same design as the adiabatic directional coupler, designed here as a power splitter, but by simulating it with a source at each of the input ports. Regarding the simulation techniques, the beam propagation method or the full FDTD method using the scattering matrix approach could be investigated to implement full 3D simulations also for large components.

Appendix A

Simulation Techniques

The waveguides used in this project have dimensions comparable to the light wavelength, therefore is not possible to use ray optics to study the light propagation through the rectangular dielectric waveguide. Therefore other techniques need to be investigated. The most common algorithms are:

- **Eigenmode Expansion Method (EME):** It solves Maxwell's Equations using a finite basis set of local modes, giving as output the scattering matrix of the structure [10].
- **Beam Propagation Method (BPM):** It used to numerically solve the Helmholtz Equation relying on the Slowly Varying Envelope Approximation [11].
- **Finite Difference Time Domain (FDTD):** It solves Maxwell's Equation by discretizing both space and time domains [11].

The Eigenmode Expansion Method is useful to simulate large devices since if they can be divided into subparts, it is possible to simulate each part at the time, and then the single scattering matrix can be multiplied together to obtain the total scattering matrix of the device, reducing so, the resources needed to run each simulation. It is particularly suited for periodic structures, where the scattering matrix of each element is computed only once. This algorithm can be adapted for bends, by considering it a set of straight waveguides with a slightly different orientation between each section, in this way it becomes a periodic structure. This is only valid if the curvature is constant, but one of the objectives of this work is to investigate the Partial Euler Bend, which has as its main characteristic a linear varying curvature, therefore the EME cannot be used to simulate it[10].

While the Beam Propagation Method does not take into consideration reflections, since it assumes that the light is travelling in only one direction[11].

In contrast, the FDTD Method can deal well with any geometry, by using a fine rectangular grid and by rigorously solving the Maxwell Equations it intrinsically takes into account the polarization, and it is generally more precise than the BPM. Therefore the

FDTD method is the simulation technique that has been used in this project, and it has been implemented using Lumerical .

A.1 Finite-Difference Time-Domain (FDTD) Method

The FDTD Method is based on solving Maxwell's equation in each point of the discretized space and time domains. The given description of this method follows the one in [11]. Given Maxwell's equations in a homogenous and non-dispersive medium:

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \tag{A.1a}$$

$$\nabla \times \vec{H} = -\frac{\partial \vec{D}}{\partial t} + \vec{J} \tag{A.1b}$$

and substituting the relation for the electric field $\vec{E}$, magnetic field $\vec{H}$, electric displacement field $\vec{D}$, and magnetic flux density $\vec{B}$, as

$$\vec{B} = \mu \vec{H} \tag{A.2a}$$

$$\vec{D} = \varepsilon \vec{E} \tag{A.2b}$$

$$\vec{J} = \sigma \vec{E} \tag{A.2c}$$

Substituting these in (A.1), and reordering the terms, such that the derivatives are on the l.h.s., the following equations are obtained:

$$\frac{\partial \vec{E}}{\partial t} = -\frac{\sigma}{\varepsilon}\vec{E} + \frac{1}{\varepsilon}\nabla \times \vec{H} \tag{A.3a}$$

$$\frac{\partial H}{\partial t} = -\frac{1}{\mu}\nabla \times \vec{E} \tag{A.3b}$$

To numerically solve these equations, they need first to be discretized, and the center difference forms can be used, as

$$\frac{\vec{E}^n - \vec{E}^{n-1}}{\Delta t} = -\frac{\sigma}{\varepsilon}\vec{E}^{n-\frac{1}{2}} + \frac{1}{\varepsilon}\nabla \times \vec{H}^{n-\frac{1}{2}} \tag{A.4a}$$

$$\frac{\vec{H}^{n+\frac{1}{2}} - \vec{H}^{n-\frac{1}{2}}}{\Delta t} = -\frac{1}{\mu}\nabla \times \vec{E}^{n} \tag{A.4b}$$

where Δt is the increment in time ($t = (n-1)\Delta t$), and $\vec{E}^{n-\frac{1}{2}}$ can be approximated by:

$$\vec{E}^{n-\frac{1}{2}} = \frac{\vec{E}^n + \vec{E}^{n-1}}{2} \tag{A.5}$$

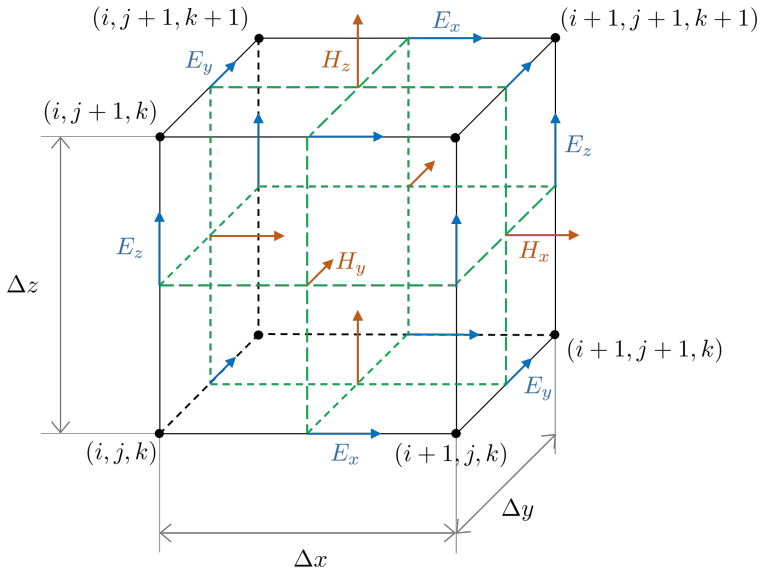


Figure A.1: Yee's unit cell

Thus, substituting (A.5) into (A.4), and solving this for $\vec{E}^n$ and $\vec{H}^{n+\frac{1}{2}}$, the following results are obtained:

$$\vec{E}^n = \frac{1 - \frac{\sigma\Delta t}{2\varepsilon}}{1 + \frac{\sigma\Delta t}{2\varepsilon}}\vec{E}^{n-1} + \frac{\frac{\Delta t}{\varepsilon}}{1 + \frac{\sigma\Delta t}{2\varepsilon}}\nabla \times \vec{H}^{n-\frac{1}{2}} \tag{A.6a}$$

$$\vec{H}^{n+\frac{1}{2}} = \vec{H}^{n-\frac{1}{2}} - \frac{\Delta t}{\mu}\nabla \times \vec{E}^2 \tag{A.6b}$$

where, for example, the x-component of $\vec{E}^n$ is equal to:

$$E_x^n = \frac{1 - \frac{\sigma\Delta t}{2\varepsilon}}{1 + \frac{\sigma\Delta t}{2\varepsilon}}E_x^{n-1} + \frac{\frac{\Delta t}{\varepsilon}}{1 + \frac{\sigma\Delta t}{2\varepsilon}}\left(\frac{\partial H_z^{n-\frac{1}{2}}}{\partial y} - \frac{\partial H_y^{n-\frac{1}{2}}}{\partial z}\right) \tag{A.7}$$

It is possible to obtain all the other fields components in the space using the leap-frog scheme introduced by Yee, with the cubic cell as shown in figure A.1. From which it can be seen that E_x^n is positioned at $(i + 1/2, j, k)$. So:

$$\frac{\partial H_z^{n-1/2}}{\partial y} = \frac{H_z^{n-1/2}(i + 1/2, j + 1/2, k) - H_z^{n-1/2}(i + 1/2, j - 1/2, k)}{\Delta y} \tag{A.8a}$$

$$\frac{\partial H_y^{n-1/2}}{\partial y} = \frac{H_y^{n-1/2}(i + 1/2, j, k + 1/2) - H_y^{n-1/2}(i + 1/2, j, k - 1/2)}{\Delta z} \tag{A.8b}$$

Then, substituting (A.8) into (A.7), the finite difference equation for E_x is obtained. All the other finite difference equations are obtained in a similar manner. The only requirement on Δt, for convergence, is that it must satisfy the Courant-Friedrichs-Lewy (CFL)

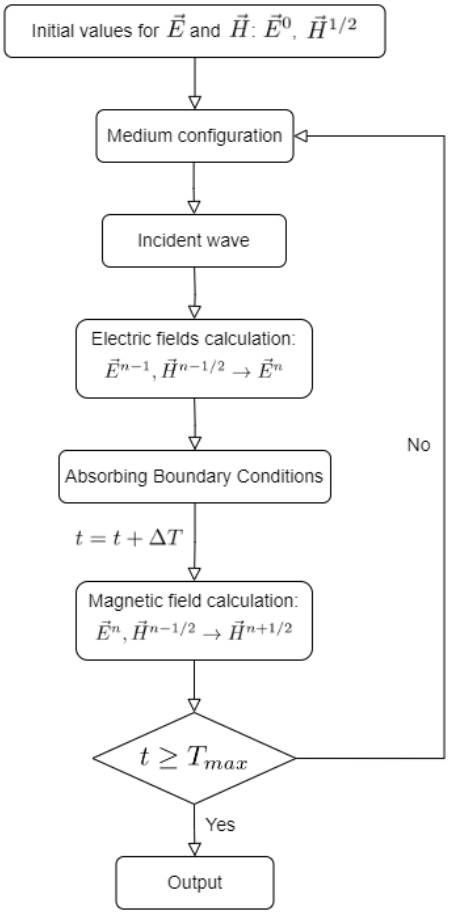


Figure A.2: FDTD algorithm flowchart (adapted from [11])

condition:

$$c\Delta t \leq \frac{1}{\sqrt{\left(\frac{1}{\Delta x}\right)^2 + \left(\frac{1}{\Delta y}\right)^2 + \left(\frac{1}{\Delta x}\right)^2}} \tag{A.9}$$

where: c is the speed of light inside the material. The general flowchart for FDTD is shown in figure A.2. Where the Absorbing Boundary Condition (ABC) are there to enforce the complete transmission at the boundaries of the simulation region, since any reflections there would not be physical. The best ABC that can be implemented in Lumerical for these kinds of structures, since they are neither periodic nor symmetric, is the Perfectly Matched Layer (PML).

A.1.1 Perfectly Matched Layer

The idea behind the PML is to put a perfectly Absorbing Boundary Layer instead of enforcing absorbing boundary conditions. This layer is made by an artificial absorbing material put around the simulation region. When the light enters this region is attenuated

by the material, so it will decay exponentially and if this region is wide enough it will be completely absorbed. But, the only fact that a different artificial material has been placed near the material inside the simulation region, creates reflections, since the translational symmetry has been broken. But a special artificial absorbing medium can be created to ensure that waves do not reflect at the interface [12].

A.1.2 Meshing

Usually, the mesh is just a Cartesian mesh, so made of a rectangular grid. But, there are some challenges related to this method.
First, numerical errors arise due to the finite mesh, since the speed of light depends on Δx, Δy, Δz, Δt, and thus is not the same as in real space. An easy solution to this problem exists: adjusting the mesh to the refracting index of the material, therefore different mesh sizes are needed for different refracting indexes [13].
Second, it is not possible to resolve interfaces to higher precision than the size of the mesh used, and this is particularly important in the case of non-linear edges, such as in bends. The first straightforward solution to this problem is to use a mesh override near the interface with a small spatial mesh, but obviously, this will increase the computational resources needed, as shown by table A.1. The second one was introduced by Yu and Mittra [14]: the Conformal Meshing Technique, where other grids than rectangular are used, for example triangular, or even curved, so that the nodes at the interface are in a one-to-one match. This method with a few modifications is the one implemented in Lumerical.

	3D	**2D**
Memory requirements	$\sim V(\lambda/dx)^3$	$\sim A(\lambda/dx)^2$
Simulation Time	$\sim V(\lambda/dx)^4$	$\sim A(\lambda/dx)^3$

Table A.1: FDTD computational costs in terms of simulation time and memory requirements, both for the 2D and 3D cases.

A.1.3 Source

The best source for this application is the mode source whereby defining the geometry: center location and span, Lumerical will calculate the possible guided modes. It is then possible to choose the preferred mode to be injected, which for the case of the structures here shown is always the fundamental TE mode. To obtain the available modes the same discretization mesh as for the FDTD is used, thus it will be interfaced with the underlying FDTD grid, therefore reducing the back reflections.

A.2 varFDTD

Since the computational cost is high for small wavelength and large simulation regions, in these cases is more efficient to use varFDTD method instead of the standard 3D FDTD method.
The idea behind the varFDTD is to collapse the 3 dimensions to 2, thus getting a 2D set of refracting indexes. The requirement for this is to not have too many vertical slab modes and the steps are:

1. Identify the vertical slab modes over the desired wavelength range.
2. Mesh and collapse to 2D by calculating the effective 2D refracting indexes. This can be done using either a variational procedure or using the reciprocity theorem.

A.3 MATLAB Integration

In Lumerical, standard mathematical functions can be used to create the geometry of components. However, to create more complicated structures it is helpful to implement their geometry first in MATLAB inside a custom function and then to use it inside Lumerical. First of all, it is needed to add the path of the folder containing these functions if they are not in the same folder as the Lumerical project. Since the workspaces of Lumerical and MATLAB are different it is important to move all the necessary variables from one to the other using `matlabput(<list of variables>)` to put variable inside the MATLAB workspace, and `matlabget(<list of variables>)` to move the results obtained in MATLAB to the Lumerical workspace. An example of the routine is shown below:

```
matlab('addpath("/AluminaPhotonicComponents/MATLAB_functions")');
matlabput(alpha_rad, p, R_eff, n_steps, d);
matlab("bend = PartialEulerBend(p, alpha_rad, R_eff, d, n_steps)");
matlabget(bend);
```

A small note regarding their uses inside a setup script is to save the folder path inside a string and then use this for the `addpath` command, instead of directly using the path there, since errors with strings may arise when running the setup script.

Appendix B

Code

B.1 Partial Euler Bend

```
function [bend, bend_in, bend_out] = PartialEulerBend(p,
    alpha, R_eff, d, n_steps)
% This function creates a Partial Euler Bends using the order
    of
% coordinates needed to create the correct structure in
    Lumerical
% p: indicates the percentage of the bend that is an Euler
    Bend instead of
% a circular bend (0 <= p <= 1)
% alpha: is the angle of the bend
% R_eff: is radius of the bend in the center
% d: width of the bend
% n_steps: number of points of each part, the total number of
    steps is
% 4*n_steps

R_0 = 1 / sqrt(2);

d = d / 2;

% circular bend case (for speed-up reasons it is better to
    separete it from
% the partial euler bend case)
if p == 0

    ivet = [0:(4*n_steps-1)];
```

```
21      % midline parametric equation
22      x_circle_center = (R_eff*sin((ivet-1)*alpha/(4*n_steps -
           1)))';
23      y_circle_center = (R_eff*(1 -
           cos((ivet-1)*alpha/(4*n_steps - 1))))';
24
25      % perpendicular lines characteristics
26      ivet = [1:4*n_steps] / (4*n_steps);
27      mc = (- 1 ./ tan(ivet * alpha))';
28      qc = y_circle_center- mc .* x_circle_center;
29      a = (1 + mc.^2);
30      b = 2*(mc.*(qc - y_circle_center) - x_circle_center);
31      c = x_circle_center.^2 + (qc - y_circle_center).^2 - d^2;
32
33      % solutions for the outer and inner edges of the bend
34      x1sol = [(-b-sqrt(b.^2 - 4.*a.*c))./(2*a), (-b+sqrt(b.^2
           - 4.*a.*c))./(2*a)];
35      x_circle_in = x1sol(x1sol < x_circle_center);
36      y_circle_in = mc.*x_circle_in + qc;
37      x_circle_out = x1sol(x1sol > x_circle_center);
38      y_circle_out = mc.*x_circle_out + qc;
39
40      % glue everything together, in a proper way for Lumerical
41      bend_in = [x_circle_in, y_circle_in];
42      bend_out = [x_circle_out, y_circle_out];
43      bend = [bend_in; flip(bend_out)];
44
45  else
46      % switching point from Euler spiral to circular arc
47      s_p = sf(p * alpha /2);
48      R_p = 1 / kf(p * alpha / 2);
49      x_p = xf(s_p);
50      y_p = yf(s_p);
51
52      deltaX = x_p - R_p*sin(p * alpha / 2);
53      deltaY = y_p - R_p*(1 - cos(p * alpha / 2));
54
55      s_0 = 2*s_p + R_p * alpha * (1 - p); % total bend length
56
57      % mid point coordinates
```

```
58      x_bend_half = R_p*sin((s_0 / 2 - s_p) / R_p + p * alpha /
           2) + deltaX;
59      y_bend_half = R_p*(1 - cos((s_0 / 2 - s_p) / R_p + p *
           alpha / 2)) + deltaY;
60
61      eta = R_eff / (y_bend_half + x_bend_half/ tan(alpha /
           2)); %rescaling factor
62
63      % first half of the midline
64      s_steps_euler_1 = linspace(0, eta * s_p, n_steps);
65      s_steps_circle_1 = linspace(eta * s_p, eta * s_0 / 2 ,
           n_steps);
66
67      x_euler_center_1 = zeros(n_steps, 1);
68      y_euler_center_1 = zeros(n_steps, 1);
69
70      x_circle_center_1 = zeros(n_steps, 1);
71      y_circle_center_1 = zeros(n_steps, 1);
72
73      for i=1:n_steps
74          x_euler_center_1(i) = eta * xf(s_steps_euler_1(i) /
               eta);
75          y_euler_center_1(i) = eta * yf(s_steps_euler_1(i) /
               eta);
76          x_circle_center_1(i) = eta *
               (R_p*sin((s_steps_circle_1(i) / eta - s_p) / R_p +
               p * alpha / 2) + deltaX);
77          y_circle_center_1(i) = eta * (R_p*(1 -
               cos((s_steps_circle_1(i) / eta - s_p) / R_p + p *
               alpha / 2)) + deltaY);
78      end
79
80      % line orthogonal to the mid point
81      m = - 1 / tan((s_steps_circle_1(end) / eta - s_p) / R_p +
           p * alpha / 2);
82      q = y_circle_center_1(end) - m * x_circle_center_1(end);
83
84      % first half inner and outer circular parts
85      mc = - (1 ./ tan((s_steps_circle_1 ./ eta - s_p) / R_p +
           p * alpha / 2))';
```

```
86      qc = y_circle_center_1 - mc .* x_circle_center_1;
87      a = (1 + mc.^2);
88      b = 2*(mc.*(qc - y_circle_center_1) - x_circle_center_1);
89      c = x_circle_center_1.^2 + (qc - y_circle_center_1).^2 -
           d^2;
90
91      x1sol = [(-b-sqrt(b.^2 - 4.*a.*c))./(2*a), (-b+sqrt(b.^2
           - 4.*a.*c))./(2*a)];
92      x_circle_in_1 = x1sol(x1sol < x_circle_center_1);
93      y_circle_in_1 = mc.*x_circle_in_1 + qc;
94      x_circle_out_1 = x1sol(x1sol > x_circle_center_1);
95      y_circle_out_1 = mc.*x_circle_out_1 + qc;
96
97      % second half inner and outer circular parts
98      x_circle_in_2 = (- 2*m*q + x_circle_in_1 -
           m^2*x_circle_in_1 + 2*m*y_circle_in_1) / (m^2 + 1);
99      y_circle_in_2 = (2*q + 2*m*x_circle_in_1 - y_circle_in_1
           + m^2*y_circle_in_1) / (m^2 + 1);
100     x_circle_out_2 = (- 2*m*q + x_circle_out_1 -
           m^2*x_circle_out_1 + 2*m*y_circle_out_1) / (m^2 + 1);
101     y_circle_out_2 = (2*q + 2*m*x_circle_out_1 -
           y_circle_out_1 + m^2*y_circle_out_1) / (m^2 + 1);
102
103     % first half inner and outer Euler parts
104     me = (- 1 ./ tan(s_steps_euler_1(2:n_steps).^2 / (2 *
           eta^2 * R_0^2)))';
105     qe = y_euler_center_1(2:n_steps) - me .*
           x_euler_center_1(2:n_steps);
106     a = (1 + me.^2);
107     b = 2*(me.*(qe - y_euler_center_1(2:n_steps)) -
           x_euler_center_1(2:n_steps));
108     c = x_euler_center_1(2:n_steps).^2 + (qe -
           y_euler_center_1(2:n_steps)).^2 - d^2;
109
110     x1sol = [(-b-sqrt(b.^2 - 4.*a.*c))./(2*a), (-b+sqrt(b.^2
           - 4.*a.*c))./(2*a)];
111     x_euler_in_1 = x1sol(x1sol < x_euler_center_1(2:n_steps));
112     y_euler_in_1 = me.*x_euler_in_1 + qe;
113     x_euler_out_1 = x1sol(x1sol >
           x_euler_center_1(2:n_steps));
```

```
114     y_euler_out_1 = me.*x_euler_out_1 + qe;
115
116     x_euler_in_1 = [x_euler_center_1(1); x_euler_in_1]; % we
           can't use tan for theta = 0
117     y_euler_in_1 = [y_euler_center_1(1) + d; y_euler_in_1];
118     x_euler_out_1 = [x_euler_center_1(1); x_euler_out_1];
119     y_euler_out_1 = [y_euler_center_1(1) - d; y_euler_out_1];
120
121     % second half inner and outer Euler parts
122     x_euler_in_2 = (- 2*m*q + x_euler_in_1 - m^2*x_euler_in_1
           + 2*m*y_euler_in_1) / (m^2 + 1);
123     y_euler_in_2 = (2*q + 2*m*x_euler_in_1 - y_euler_in_1 +
           m^2*y_euler_in_1) / (m^2 + 1);
124     x_euler_out_2 = (- 2*m*q + x_euler_out_1 -
           m^2*x_euler_out_1 + 2*m*y_euler_out_1) / (m^2 + 1);
125     y_euler_out_2 = (2*q + 2*m*x_euler_out_1 - y_euler_out_1
           + m^2*y_euler_out_1) / (m^2 + 1);
126
127     % glue everything together, in a proper way for Lumerical
128     euler_in_1(1:n_steps, 1:2) = [x_euler_in_1, y_euler_in_1];
129     circle_in_1(1:n_steps, 1:2) = [x_circle_in_1,
           y_circle_in_1];
130     euler_out_1(1:n_steps, 1:2) = [x_euler_out_1,
           y_euler_out_1];
131     circle_out_1(1:n_steps, 1:2) = [x_circle_out_1,
           y_circle_out_1];
132
133     euler_in_2(1:n_steps, 1:2) = [x_euler_in_2, y_euler_in_2];
134     circle_in_2(1:n_steps, 1:2) = [x_circle_in_2,
           y_circle_in_2];
135     euler_out_2(1:n_steps, 1:2) = [x_euler_out_2,
           y_euler_out_2];
136     circle_out_2(1:n_steps, 1:2) = [x_circle_out_2,
           y_circle_out_2];
137
138     bend_in = [euler_in_1; circle_in_1; flip(circle_in_2);
           flip(euler_in_2)];
139     bend_out = [euler_out_1; circle_out_1;
           flip(circle_out_2); flip(euler_out_2)];
140
```

```
141     bend = [bend_in; flip(bend_out)];
142     bend = double(bend);
143 
144 end
145 
146 end
```

B.2 S-Bend

```
1  (function [bend, bend_in, bend_out] = SBend(p, alpha, R_eff,
       d, n_steps, l, reverse)
2  % This function creates an S bend using two Partial Euler
       Bends using
3  % the order of coordinates needed to create the correct
       structure in
4  % Lumerical
5  % p: indicates the percentage of the bend that is an Euler
       Bend instead of
6  % a circular bend (0 <= p <= 1)
7  % alpha: is the angle of the bend
8  % R_eff: is radius of the bend in the center
9  % d: width of the bend
10 % n_steps: number of points of each part, the total number of
       steps is
11 % 4*n_steps
12 % l: length of the straight part in the middle (between the
       two bends)
13 % reverse: (boolean), if false is an input bend (left part
       higher than
14 % right part), if true is an output bend (right part higher
       than left one)
15 
16 R_0 = 1 / sqrt(2);
17 
18 d = d / 2;
19 
20 % circular bend case (for speed-up reasons it is better to
       separete it from
21 % the partial euler bend case)
22 if p == 0
```

```
23
24      % first half midline parametric equations
25      ivet = [0:(4*n_steps-1)];
26      x_circle_center_1 =
           (R_eff*sin((ivet-1)*alpha/(4*n_steps-1)))';
27      y_circle_center_1 = (R_eff*(1 -
           cos((ivet-1)*alpha/(4*n_steps-1))))';
28
29      % perpendicular lines characteristics
30      ivet = [1:4*n_steps] / (4*n_steps);
31      mc = (- 1 ./ tan(ivet * alpha))';
32      qc = y_circle_center_1 - mc .* x_circle_center_1;
33      a = (1 + mc.^2);
34      b = 2*(mc.*(qc - y_circle_center_1) - x_circle_center_1);
35      c = x_circle_center_1.^2 + (qc - y_circle_center_1).^2 -
           d^2;
36
37      % solutions for the outer and inner edges of the bend
38      x1sol = [(-b-sqrt(b.^2 - 4.*a.*c))./(2*a), (-b+sqrt(b.^2
           - 4.*a.*c))./(2*a)];
39      x_circle_in_1 = x1sol(x1sol < x_circle_center_1);
40      y_circle_in_1 = mc.*x_circle_in_1 + qc;
41      x_circle_out_1 = x1sol(x1sol > x_circle_center_1);
42      y_circle_out_1 = mc.*x_circle_out_1 + qc;
43
44      % second half
45      x_in_0 = x_circle_in_1(end);
46      y_in_0 = y_circle_in_1(end);
47
48      x_out_0 = x_circle_out_1(end);
49      y_out_0 = y_circle_out_1(end);
50
51      m = (y_out_0 - y_in_0) / (x_out_0 - x_in_0);
52
53      x_circle_in_2 = - flip(x_circle_out_1) + x_in_0 + x_out_0
           - l*m/sqrt(1+m^2);
54      y_circle_in_2 = - flip(y_circle_out_1) + y_in_0 + y_out_0
           + l*1/sqrt(1+m^2);
55
56      x_circle_out_2 = - flip(x_circle_in_1) + x_out_0 + x_in_0
```

```
        - l*m/sqrt(1+m^2);
57      y_circle_out_2 = - flip(y_circle_in_1) + y_out_0 + y_in_0
        + l*1/sqrt(1+m^2);
58
59      % glue everything together, in a proper way for Lumerical
60      circle_in_1 = [x_circle_in_1, y_circle_in_1];
61      circle_out_1 = [x_circle_out_1, y_circle_out_1];
62
63      circle_in_2 = [x_circle_in_2, y_circle_in_2];
64      circle_out_2 = [x_circle_out_2, y_circle_out_2];
65
66      bend_in = [circle_in_1; circle_in_2];
67      bend_out = [circle_out_1; circle_out_2];
68
69      bend = [bend_in; flip(bend_out)];
70      bend = double(bend);
71
72  else
73      % switching point from Euler spiral to circular arc
74      s_p = sf(p * alpha /2);
75      R_p = 1 / kf(p * alpha / 2);
76      x_p = xf(s_p);
77      y_p = yf(s_p);
78
79      deltaX = x_p - R_p*sin(p * alpha / 2);
80      deltaY = y_p - R_p*(1 - cos(p * alpha / 2));
81
82      s_0 = 2*s_p + R_p * alpha * (1 - p); % total bend length
83
84      % mid point coordinates
85      x_bend_half = R_p*sin((s_0 / 2 - s_p) / R_p + p * alpha /
        2) + deltaX;
86      y_bend_half = R_p*(1 - cos((s_0 / 2 - s_p) / R_p + p *
        alpha / 2)) + deltaY;
87
88      eta = R_eff / (y_bend_half + x_bend_half/ tan(alpha /
        2)); %rescaling factor
89
90      % first half of the midline
91      s_steps_euler_1 = linspace(0, eta * s_p, n_steps);
```

```
92      s_steps_circle_1 = linspace(eta * s_p, eta * s_0 / 2 ,
            n_steps);
93
94      x_euler_center_1 = zeros(n_steps, 1);
95      y_euler_center_1 = zeros(n_steps, 1);
96
97      x_circle_center_1 = zeros(n_steps, 1);
98      y_circle_center_1 = zeros(n_steps, 1);
99
100     for i=1:n_steps
101         x_euler_center_1(i) = eta * xf(s_steps_euler_1(i) /
                eta);
102         y_euler_center_1(i) = eta * yf(s_steps_euler_1(i) /
                eta);
103         x_circle_center_1(i) = eta *
                (R_p*sin((s_steps_circle_1(i) / eta - s_p) / R_p +
                p * alpha / 2) + deltaX);
104         y_circle_center_1(i) = eta * (R_p*(1 -
                cos((s_steps_circle_1(i) / eta - s_p) / R_p + p *
                alpha / 2)) + deltaY);
105     end
106
107     % line orthogonal to the mid point
108     m = - 1 / tan((s_steps_circle_1(end) / eta - s_p) / R_p +
            p * alpha / 2);
109     q = y_circle_center_1(end) - m * x_circle_center_1(end);
110
111     % first half inner and outer circular parts
112     mc = - (1 ./ tan((s_steps_circle_1 ./ eta - s_p) / R_p +
            p * alpha / 2))';
113     qc = y_circle_center_1 - mc .* x_circle_center_1;
114     a = (1 + mc.^2);
115     b = 2*(mc.*(qc - y_circle_center_1) - x_circle_center_1);
116     c = x_circle_center_1.^2 + (qc - y_circle_center_1).^2 -
            d^2;
117
118     x1sol = [(-b-sqrt(b.^2 - 4.*a.*c))./(2*a), (-b+sqrt(b.^2
            - 4.*a.*c))./(2*a)];
119     x_circle_in_1 = x1sol(x1sol < x_circle_center_1);
120     y_circle_in_1 = mc.*x_circle_in_1 + qc;
```

```
121     x_circle_out_1 = x1sol(x1sol > x_circle_center_1);
122     y_circle_out_1 = mc.*x_circle_out_1 + qc;
123
124     % second half circle part
125     x_in_0 = x_circle_in_1(end);
126     y_in_0 = y_circle_in_1(end);
127     x_out_0 = x_circle_out_1(end);
128     y_out_0 = y_circle_out_1(end);
129
130     m = (y_out_0 - y_in_0) / (x_out_0 - x_in_0);
131
132     % second half inner and outer circular parts
133     x_circle_in_2 = - flip(x_circle_out_1) + x_in_0 + x_out_0
           - l*m/sqrt(1+m^2);
134     y_circle_in_2 = - flip(y_circle_out_1) + y_in_0 + y_out_0
           + l*1/sqrt(1+m^2);
135     x_circle_out_2 = - flip(x_circle_in_1) + x_out_0 + x_in_0
           - l*m/sqrt(1+m^2);
136     y_circle_out_2 = - flip(y_circle_in_1) + y_out_0 + y_in_0
           + l*1/sqrt(1+m^2);
137
138     % first half inner and outer Euler parts
139     me = (- 1 ./ tan(s_steps_euler_1(2:n_steps).^2 / (2 *
           eta^2 * R_0^2)))';
140     qe = y_euler_center_1(2:n_steps) - me .*
           x_euler_center_1(2:n_steps);
141     a = (1 + me.^2);
142     b = 2*(me.*(qe - y_euler_center_1(2:n_steps)) -
           x_euler_center_1(2:n_steps));
143     c = x_euler_center_1(2:n_steps).^2 + (qe -
           y_euler_center_1(2:n_steps)).^2 - d^2;
144
145     x1sol = [(-b-sqrt(b.^2 - 4.*a.*c))./(2*a), (-b+sqrt(b.^2
           - 4.*a.*c))./(2*a)];
146     x_euler_in_1 = x1sol(x1sol < x_euler_center_1(2:n_steps));
147     y_euler_in_1 = me.*x_euler_in_1 + qe;
148     x_euler_out_1 = x1sol(x1sol >
           x_euler_center_1(2:n_steps));
149     y_euler_out_1 = me.*x_euler_out_1 + qe;
150
```

```
151     x_euler_in_1 = [x_euler_center_1(1); x_euler_in_1]; % we
            can't use tan for theta = 0
152     y_euler_in_1 = [y_euler_center_1(1) + d; y_euler_in_1];
153     x_euler_out_1 = [x_euler_center_1(1); x_euler_out_1];
154     y_euler_out_1 = [y_euler_center_1(1) - d; y_euler_out_1];
155
156     % second half inner and outer Euler parts
157     x_euler_in_2 = - flip(x_euler_out_1) + x_in_0 + x_out_0 -
            l*m/sqrt(1+m^2);
158     y_euler_in_2 = - flip(y_euler_out_1) + y_in_0 + y_out_0 +
            l*1/sqrt(1+m^2);
159     x_euler_out_2 = - flip(x_euler_in_1) + x_out_0 + x_in_0 -
            l*m/sqrt(1+m^2);
160     y_euler_out_2 = - flip(y_euler_in_1) + y_out_0 + y_in_0 +
            l*1/sqrt(1+m^2);
161
162     % glue everything together, in a proper way for Lumerical
163     euler_in_1(1:n_steps, 1:2) = [x_euler_in_1, y_euler_in_1];
164     circle_in_1(1:n_steps, 1:2) = [x_circle_in_1,
            y_circle_in_1];
165     euler_out_1(1:n_steps, 1:2) = [x_euler_out_1,
            y_euler_out_1];
166     circle_out_1(1:n_steps, 1:2) = [x_circle_out_1,
            y_circle_out_1];
167
168     euler_in_2(1:n_steps, 1:2) = [x_euler_in_2, y_euler_in_2];
169     circle_in_2(1:n_steps, 1:2) = [x_circle_in_2,
            y_circle_in_2];
170     euler_out_2(1:n_steps, 1:2) = [x_euler_out_2,
            y_euler_out_2];
171     circle_out_2(1:n_steps, 1:2) = [x_circle_out_2,
            y_circle_out_2];
172
173     bend_in = [euler_in_1; circle_in_1; circle_in_2;
            euler_in_2];
174     bend_out = [euler_out_1; circle_out_1; circle_out_2;
            euler_out_2];
175
176     bend = [bend_in; flip(bend_out)];
177     bend = double(bend);
```

```
178
179 end
180 if reverse
181     bend = [-bend(:,1), bend(:,2)];
182 end
183 end
```

B.3 Lumerical Adiabatic Directional Coupler Setup Script

```
1  select("::model");
2  set("setup script", '
3  path_string =
       "/scratch/SemesterProjects/AluminaPhotonicComponents/MATLAB_functions";
4  matlabput(path_string);
5  matlab("addpath(path_string)");
6
7  matlabput(alpha_rad, p, R_eff, n_steps, w, w1, w2, deltaw,
       w3, w4, L, Lt, sep, s);
8
9  matlab("s_bend_up_out = SBendUpOut(p, alpha_rad, R_eff, w3,
       n_steps, L, s);");
10 matlab("s_bend_up_in = SBendUpIn(p, alpha_rad, R_eff, w1,
       n_steps, s);");
11 #matlab("taper_up_in = TaperUpIn(w,w1,Lt,n_steps,
       s_bend_up_in);");
12 #matlab("taper_up_center = TaperUpCenter(w3,w1,L,n_steps);");
13 #matlab("taper_up_out = TaperUpOut(w,w3,Lt,n_steps,
       s_bend_up_out);");
14
15 matlab("s_bend_down_out = SBendDownOut(p, alpha_rad, R_eff,
       w4, n_steps, sep, L, s);");
16 matlab("s_bend_down_in = SBendDownIn(p, alpha_rad, R_eff, w2,
       n_steps, sep, s);");
17
18 matlabget(s_bend_up_in, s_bend_up_out, s_bend_down_in,
       s_bend_down_out);
19
20 x_min = min(s_bend_up_in(:,1));
21 x_max = max(s_bend_up_out(:,1));
22 y_min = min(s_bend_down_in(:,2));
```

```
23 y_max = max(s_bend_up_in(:,2));
24
25 margin = (x_max - x_min) / 5;
26
27 x_source = x_min - lin_wg_len * 1/2 + 10e-9;
28 y_source = y_max - w1/2;
29
30 outer_XY_dims = {
31     "x min" : x_min - lin_wg_len - margin,
32     "x max" : x_max + lin_wg_len + margin,
33     "y min" : y_min - margin,
34     "y max" : y_max + margin
35 };
36
37 select("Oxide");
38 set("x min", x_min - lin_wg_len - margin);
39 set("x max", x_max + lin_wg_len + margin);
40 set("y min", y_min - 4*margin);
41 set("y max", y_max + 4*margin);
42 set("render type", 1);
43 set("detail", 0.3);
44 set("alpha", 0.3);
45 set("override mesh order from material database", 1);
46 set("mesh order", 10);
47
48
49 #select("TaperUpIn");
50 #set("material", mat_wg);
51 #set("vertices", taper_up_in);
52
53 select("SBendUpIn");
54 set("material", mat_wg);
55 set("vertices", s_bend_up_in);
56
57 select("SBendUpOut");
58 set("material", mat_wg);
59 set("vertices", s_bend_up_out);
60
61 taper_up_center = [[0, w1/2]; [L, w3/2]; [L, -w3/2]; [0,
      -w1/2]];
```

```
62
63 select("TaperUpCenter");
64 set("material", mat_wg);
65 set("vertices", taper_up_center);
66
67 #select("TaperUpOut");
68 #set("material", mat_wg);
69 #set("vertices", taper_up_out);
70
71 taper_down_center = [[0, -sep+w2/2]; [0, -sep-w2/2];  [L,
      -sep-w4/2]; [L, -sep+w4/2]];
72
73 select("TaperDownCenter");
74 set("material", mat_wg);
75 set("vertices", taper_down_center);
76
77 #select("TaperDownIn");
78 #set("material", mat_wg);
79 #set("vertices", taper_down_in);
80
81 #select("TaperDownOut");
82 #set("material", mat_wg);
83 #set("vertices", taper_down_out);
84
85 select("SBendDownOut");
86 set("material", mat_wg);
87 set("vertices", s_bend_down_out);
88
89 select("SBendDownIn");
90 set("material", mat_wg);
91 set("vertices", s_bend_down_in);
92
93 select("LinearWgUpIn");
94 #set("material", mat_wg);
95 set("x min", -lin_wg_len + min(s_bend_up_in(:,1)));
96 set("x max", min(s_bend_up_in(:,1)));
97 set("y", max(s_bend_up_in(:,2)) - w1/2);
98 set("y span", w1);
99 #set("z span", z_thick_wg);
100
```

```
101 select("LinearWgUpOut");
102 #set("material", mat_wg);
103 set("x min", max(s_bend_up_out(:,1)));
104 set("x max", max(s_bend_up_out(:,1)) + lin_wg_len);
105 set("y min", max(s_bend_up_out(:,2)) - w3);
106 set("y max", max(s_bend_up_out(:,2)));
107 #set("z span", z_thick_wg);
108
109 select("LinearWgDownIn");
110 #set("material", mat_wg);
111 set("x min", -lin_wg_len + min(s_bend_down_in(:,1)));
112 set("x max", min(s_bend_down_in(:,1)));
113 set("y", min(s_bend_down_in(:,2)) + w2/2);
114 set("y span", w2);
115 #set("z span", z_thick_wg);
116
117 select("LinearWgDownOut");
118 #set("material", mat_wg);
119 set("x min", max(s_bend_down_out(:,1)));
120 set("x max", max(s_bend_down_out(:,1)) + lin_wg_len);
121 set("y min", min(s_bend_down_out(:,2)));
122 set("y max", min(s_bend_down_out(:,2)) + w4);
123 #set("z span", z_thick_wg);
124
125 select("varFDTD");
126 #addvarfdtd;
127 set("simulation time", sim_time);
128 set("x min", x_min - lin_wg_len/2);
129 set("x max", x_max);
130 set("y min", min(s_bend_down_in(:,2)) - 2*margin);
131 set("y max", y_max + 2*margin);
132 test_matrix = [[-(x_max - x_min)/2, y_max + 2*w1]; [-(x_max -
       x_min)/2, y_min - 5*w2]; [(x_max - x_min)/2, y_min -
       5*w2]; [(x_max - x_min)/2, y_max + 2*w1]];
133 set("test points", test_matrix);
134 set("x0", -(x_max-(x_min - lin_wg_len/2))/2 +lin_wg_len/4);
135 set("y0", (y_max + margin - (min(s_bend_down_in(:,2)) -
       margin))/2 - margin - w1/2);
136 set("mesh accuracy", mesh_acc);
137
```

```
select("post::source");
#addmodesource;
##set("name", "source");
#set("injection axis", "x");
#set("direction", 2);
set("x", x_source);
set("y", y_source);
set("y span", 10*w);
set("mode selection", "fundamental mode");
#set("use global source settings", 1);
#set("mode selection", 1);
#set("selected mode number", input_mode_num);
##addtogroup("post");

select("movie_monitor");
set("x min", x_min-lin_wg_len/2);
set("x max", x_max);
set("y min", min(s_bend_down_in(:,2))-margin);
set("y max", y_max+margin);
set("enabled", enable_movie);

lams = [lambda_um, lambda_um2, lambda_um3]*1e-6;

for (k_lam=1:3){
    select("in_plane_monitor_lam"+num2str(k_lam));
    set("monitor type", 7); # 2D, y-normal
    set("override global monitor settings", 0);
    #set("use source limits", 0);
    #set("frequency points", 1);
    #set("wavelength center", lams(k_lam));
    #set("wavelength span", 0);
    set("x min", x_min-lin_wg_len/2);
    set("x max", x_max);
    set("y min", min(s_bend_down_in(:,2))-margin);
    set("y max", y_max+margin);
    set("z", 0);
    #set("z", z_thick_wg/2);
    set("enabled", enable_monitor);
}

```

```
178 for (k_lam=1:3){
179     for (k_wg=1:1){
180         select("field_wg"+num2str(k_wg)+"_lam"+num2str(k_lam));
181         set("monitor type", 5); # 2D, x-normal
182         #this needs to be modified
183         set("x",  x_max);
184         set("y",  max(s_bend_up_out(:,2)) - w3/2);
185         set("y span", abs((max(s_bend_up_out(:,2)) - w3/2) -
               (min(s_bend_down_out(:,2)) + w4/2)));
186         #set("use source limits", 0);
187         set("use relative coordinates", 0);
188 
189         select("field_wg_b"+num2str(k_wg)+"_lam"+num2str(k_lam));
190         set("monitor type", 5); # 2D, x-normal
191         #this needs to be modified
192         set("x",  x_max);
193         set("y", min(s_bend_down_out(:,2)) + w4/2);
194         set("y span", abs((max(s_bend_up_out(:,2)) - w3/2) -
               (min(s_bend_down_out(:,2)) + w4/2)));
195         #set("use source limits", 0);
196         set("use relative coordinates", 0);
197         }
198     }
199 }
200 
201 
202 for (i=1:length(output_analysis_modes)){
203     for (k_lam=1:3){
204 for (k_wg=1:1){
205             select("::model::post::mode_expansion_wg"+
206                 num2str(k_wg)+"_lam"+num2str(k_lam));
207             set("x",  x_max);
208             set("y", max(s_bend_up_out(:,2)) - w3/2);
209             set("y span", abs((max(s_bend_up_out(:,2)) -
                   w3/2) - (min(s_bend_down_out(:,2)) + w4/2)));
210             set("mode selection", "fundamental mode");
211             #set("z min", -z_thick_wg/2);
212             #set("z max", z_thick_wg/2);
213             setexpansion("input","::model::field_wg"+
214                 num2str(k_wg)+"_lam"+num2str(k_lam));
```

```
215             set("use relative coordinates", 0);
216
217             select("::model::post::mode_expansion_wg_b"+
218                 num2str(k_wg)+"_lam"+num2str(k_lam));
219             # this needs to be modified
220             set("x",  x_max);
221             set("y", min(s_bend_down_out(:,2)) + w4/2);
222             set("y span", abs((max(s_bend_up_out(:,2)) -
                   w3/2) - (min(s_bend_down_out(:,2)) + w4/2)));
223             set("mode selection", "fundamental mode");
224             #set("z min", -z_thick_wg/2);
225             #set("z max", z_thick_wg/2);
226             setexpansion("input","::model::field_wg_b"+
227                 num2str(k_wg)+"_lam"+num2str(k_lam));
228             set("use relative coordinates", 0);
229         }
230     }
231 }
232
233 ');
```

Bibliography

[1] Karan K. Mehta, Chi Zhang, Maciej Malinowski, Thanh-Long Nguyen, Martin Stadler, and Jonathan P. Home. Integrated optical multi-ion quantum logic. *Nature*, 586(7830):533–537, October 2020. doi:10.1038/s41586-020-2823-6.

[2] Karan K. Mehta, Colin D. Bruzewicz, Robert McConnell, Rajeev J. Ram, Jeremy M. Sage, and John Chiaverini. Integrated optical addressing of an ion qubit. *Nature Nanotechnology*, 11(12):1066–1070, December 2016. doi:10.1038/nnano.2016.139.

[3] Florian Vogelbacher, Stefan Nevlacsil, Martin Sagmeister, Jochen Kraft, Karl Unterrainer, and Rainer Hainberger. Analysis of silicon nitride partial Euler waveguide bends. *Optics Express*, 27(22):31394–31406, October 2019. doi:10.1364/OE.27.031394.

[4] Andrea Melloni, Paolo Monguzzi, Raffaella Costa, and Mario Martinelli. Design of curved waveguides: the matched bend. *J. Opt. Soc. Am. A*, 20(1):130–137, Jan 2003. doi:10.1364/JOSAA.20.000130.

[5] Takeshi Fujisawa, Shuntaro Makino, Takanori Sato, and Kunimasa Saitoh. Low-loss, compact, and fabrication-tolerant Si-wire 90Â° waveguide bend using clothoid and normal curves for large scale photonic integrated circuits. *Optics Express*, 25(8):9150, April 2017. doi:10.1364/OE.25.009150.

[6] Daoxin Dai and Shipeng Wang. Asymmetric directional couplers based on silicon nanophotonic waveguides and applications. *Frontiers of Optoelectronics*, 9(3):450–465, September 2016. doi:10.1007/s12200-016-0557-8.

[7] E. A. J. Marcatili. Dielectric rectangular waveguide and directional coupler for integrated optics. *The Bell System Technical Journal*, 48(7), September 1969. doi:10.1002/j.1538-7305.1969.tb01166.x.

[8] Deng Mao, Yun Wang, Eslam El-Fiky, Luhua Xu, Amar Kumar, Maxime Jaques, Alireza Samani, Olivier Carpentier, Santiago Bernal, Md Samiul Alam, Jinsong Zhang, Mingyue Zhu, Ping-Chiek Koh, and David V. Plant. Adiabatic coupler with design-intended splitting ratio. *Journal of Lightwave Technology*, 37(24):6147–6155, 2019. doi:10.1109/JLT.2019.2946948.

[9] The FDTD Solver Region - Setup Tips - Mesh Setup Tips, . URL `https://optics.ansys.com/hc/en-us/articles/360044942074-The-FDTD-Solver-Region-Setup-Tips-Mesh-Setup-Tips`.

[10] Dominic F. G. Gallagher and Thomas P. Felici. Eigenmode expansion methods for simulation of optical propagation in photonics: pros and cons. In *Integrated Optics: Devices, Materials, and Technologies VII*, volume 4987, pages 69 – 82. International Society for Optics and Photonics, SPIE, 2003. doi:10.1117/12.473173.

[11] Katsunari Okamoto. Chapter 7 - Beam propagation method. In Katsunari Okamoto, editor, *Fundamentals of Optical Waveguides (Second Edition)*. Academic Press, January 2006. doi:10.1016/B978-012525096-2/50008-8.

[12] Steven G. Johnson. Notes on Perfectly Matched Layers (PMLs), August 2021. arXiv:2108.05348 [physics].

[13] Understanding Mesh Refinement and Conformal Mesh in FDTD, . URL `https://optics.ansys.com/hc/en-us/articles/360034382594-Understanding-Mesh-Refinement-and-Conformal-Mesh-in-FDTD`.

[14] Wenhua Yu and R. Mittra. A conformal finite difference time domain technique for modeling curved dielectric surfaces. *IEEE Microwave and Wireless Components Letters*, 11(1):25–27, January 2001. doi:10.1109/7260.905957.