

Handbook
for
Generic Photonic IC Design

Editors: Meint Smit and Xaveer Leijtens

4-4-2026

© ⓘ ⓘ ⓘ ⓘ *Handbook for generic photonic IC design*, by the *Photonic Integration group*, Technische Universiteit Eindhoven, is licensed under a Creative Commons “Attribution-NonCommercial-NoDerivatives 4.0 International” license.

We traced the ownership of all figures used as far as we could. However, if you are a copyright owner and believe we used your work without permission, please contact us at coordinator@jeppix.eu.

Appendix E

VPIdeviceDesigner Detailed Descriptions

HANNES LÜDER

E.1 Introduction

This appendix provides detailed descriptions of how to model the simulation examples shown in Chapter 5 with VPIdeviceDesigner. It is fully self-contained and enables the readers to reproduce all the shown results. The code is compatible with VPIdeviceDesigner 2.8.2 and above.

We would like to point out that many of the results are arranged in rather complex figures. This is done for the specific needs of this books. In the daily work with VPIdeviceDesigner, plotting results is usually much easier (typically just a one-liner).

It is important to note that this book and VPIdeviceDesigner use different coordinate system conventions. In this book, the coordinate system for waveguides is defined as follows:

- z is the propagation direction.
- x is the axis perpendicular to the layer interfaces (vertical).
- y is the remaining axis, parallel to the layer interfaces and orthogonal to z .

In VPIdeviceDesigner, however, a different convention is used:

- z is also the propagation direction.
- x is parallel to the layer interfaces and orthogonal to z .
- y is the axis perpendicular to the layer interfaces (vertical).

The simulations shown in this appendix will be done in the VPIdeviceDesigner coordinate system. For the resulting plots, however, we will relabel the axes to match the convention used in this book.

E.2 Modeling Slab Waveguides

In this section, guided modes in *slab* waveguides and their properties are modeled using VPIdeviceDesigner.

E.2.1 Optical Materials

The optical materials used in these simulations are defined as dispersionless and lossless dielectric materials:

```
from vipda.material import Dielectric

Air = Dielectric(n=1.0, name='Air', color='white')
InP = Dielectric(n=3.17, name='InP', color='#C8C8FF')
InGaAsP_Q125 = Dielectric(n=3.36, name='InGaAsP_Q125', color='#00FF00')
```

InGaAsP_Q125 defines the quaternary compound material InGaAsP. Its composition is chosen such that its band gap corresponds to a wavelength of 1.25 μm , and the lattice constant is matched to the InP substrate. However, as mentioned above, we will treat it as a dispersionless dielectric material for the simulations.

E.2.2 Slab Waveguide Definition

The slab waveguide cross-section is defined as a function of the core layer thickness and the cover material:

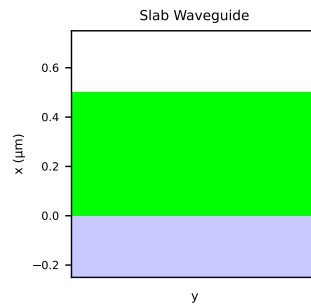
```
import matplotlib.pyplot as plt
from vipda.layout import HalfPlane, Layer, Layout2D, length2default
from vipda.mesh import Mesh2D

def create_slab(thickness='500 nm', cover=Air):
    # Convert thickness to default length units
    # to enable mathematical operations on it:
    thickness = length2default(thickness)
    substrate = HalfPlane(InP)
    core = Layer(InGaAsP_Q125, height=thickness, bottom=substrate.center)
    wg = Layout2D(cover, substrate, core)
    mesh = Mesh2D(wg)
    mesh.box = wg.box.expand('10.0 um')
    mesh.update(dy='10 nm')
    return wg, mesh
```

Please note that in addition to the slab waveguide cross-section `wg`, the `slab(...)` function returns a non-uniform discretization mesh, adapted for more accurate modeling of the slab waveguide with the given thickness. The waveguide cross-section can be verified as follows:

```
wg, mesh = create_slab(thickness=0.5, cover=Air)

fig, ax = plt.subplots()
wg.plot(fig=ax)
ax.set_xlabel('y')
ax.set_xticks([])
ax.set_ylabel('x (um)')
ax.set_title('Slab Waveguide');
```



E.2.3 Fundamental Mode Profiles

To calculate the fundamental mode(s), first a mode solver for the modeled slab waveguide must be created:

```
from vpipda.mode import SolverModeFDM
solver = SolverModeFDM(create_slab)
```

To calculate the first two fundamental modes that are supported by the slab waveguide with a core thickness of $0.5 \mu\text{m}$ at a wavelength of $1.55 \mu\text{m}$, the `sim.calc(...)` method is used. The calculation is conducted for two scenarios – one for the **symmetric** slab waveguide covered by InP material and one for **asymmetric** slab waveguide covered by Air to highlight the impact of symmetry on optical properties of the waveguide.

```
wl = '1.55 um'
modes_InP = solver.calc(wl, nmodes=2, thickness=0.5, cover=InP)
modes_Air = solver.calc(wl, nmodes=2, thickness=0.5, cover=Air)
```

The objects `modes_InP` and `modes_Air` contain the calculated modes, which allow us to easily analyze the mode profiles, effective mode indices, and other mode properties of common interest:

```
for m in modes_InP:
    # Determine the mode polarization:
    if m.E.x.max_abs() > m.E.y.max_abs():
        mode_polarization = 'TE'
    else:
        mode_polarization = 'TM'

    print(f"neff[{m.name}] = {m.neff()} ({mode_polarization} mode)")
```

```
neff[0] = 3.2677205228039328 (TE mode)
neff[1] = 3.261577238395286 (TM mode)
```

Using the third-party `matplotlib` library allows us to depict the spatial mode field profile (1D) of each waveguide for each polarization state and to fine-control the visualization:

```

import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2)

ax = axes[0]
modes_InP[0].E.x.slice(x=0).plot(fig=ax, color='royalblue',
                                  label='TE, $E_y$')
modes_InP[1].H.x.slice(x=0).plot(fig=ax, color='darkorange',
                                  linestyle='--',
                                  label='TM, $H_y$')
modes_Air[0].E.x.slice(x=0).plot(fig=ax, color='royalblue',)
modes_Air[1].H.x.slice(x=0).plot(fig=ax, color='darkorange',
                                  linestyle='--',
                                  edges=True, edges_linestyle=':',
                                  edges_linewidth=1)

ax.annotate('sym.', xy=(0.6, 0.5), xytext=(1.0, 0.55),
            arrowprops={'arrowstyle': '-|>'})

ax.annotate('asym.', xy=(0.55, 0.2), xytext=(1.0, 0.3),
            arrowprops={'arrowstyle': '-|>'})

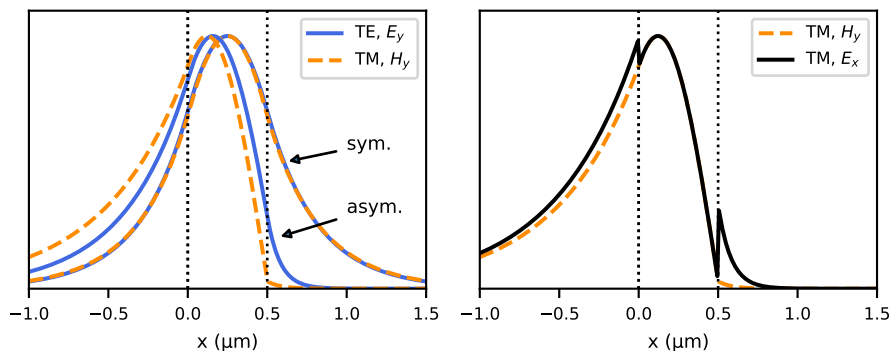
ax.set_xlim(-1.0, 1.5)
ax.set_ylim(0, 1.1)
ax.set_yticks([])
ax.set_ylabel('')
ax.set_xlabel('x (um)');
ax.set_title('')

ax = axes[1]
modes_Air[1].H.x.slice(x=0).plot(fig=ax, color='darkorange',
                                  linestyle='--',
                                  label='TM, $H_y$')
abs(modes_Air[1].E.y).slice(x=0).plot(
    fig=ax, color='black', label='TM, $E_x$',
    edges=True, edges_linestyle=':', edges_linewidth=1
)

ax.set_xlim(-1.0, 1.5)
ax.set_ylim(0, 1.1)
ax.set_yticks([])
ax.set_ylabel('')
ax.set_xlabel('x (um)')
ax.set_title('')

fig.tight_layout()

```



E.2.4 Effective Mode Index vs Slab Thickness

Thanks to VPIdeviceDesigner's parameter sweep tool, the effective indices for the first 4 modes, supported by both symmetric and asymmetric waveguide slabs, can be calculated as a function of the slab thickness:

```
import numpy as np
from vipda import units as u

# Make a sweep over 50 thickness values:
thicknesses = np.linspace(1.1, 0.01, 50) * u.um

# Set a better display name for 'thickness':
solver.set_sweep_parameter('thickness',
                           name='Waveguide Film Thickness',
                           units='um')

modes_InP = solver.calc(wl, nmodes=4, thickness=thicknesses, cover=InP)
modes_Air = solver.calc(wl, nmodes=4, thickness=thicknesses, cover=Air)

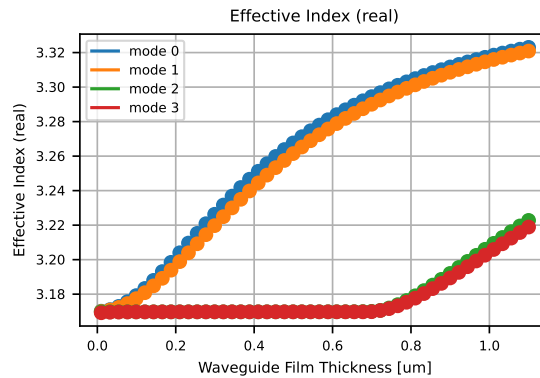
Modes calculation: 100%|=====| 50/50 [00:03<00:00, 16.25point/s]
Modes calculation: 100%|=====| 50/50 [00:03<00:00, 16.60point/s]
```

This custom parameter sweep is possible because the `slab(...)` function, which was used to initialize the `sim` object, supports an argument with the exact name `thickness`. Hence, the simulation sweep can pass the thicknesses to the `slab(...)` function and generate a specific waveguide cross-section for each sweep step.

The calculated effective mode indices can be easily visualized for the analysis. For instance, for the symmetric waveguide:

```
fig = plt.figure()

for m in modes_InP:
    m.neff.plot(fig=fig, label='mode ' + str(m.name))
```



Using the third-party `matplotlib` library, the effective indices of all calculated modes corresponding to both symmetric and asymmetric scenarios considering different polarization states are visualized using the following script:

```
fig, ax = plt.subplots()

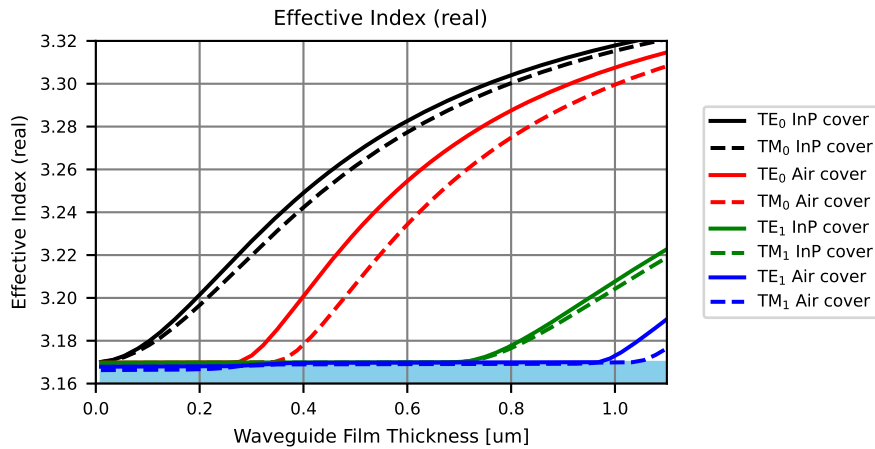
mode_mapping = {'TE', 0): 0, ('TM', 0): 1, ('TE', 1): 2, ('TM', 1): 3}
cover_mapping = {'InP': modes_InP, 'Air': modes_Air}
ls_mapping = {'TE': 'solid', 'TM': 'dashed'}
color_mapping = {'InP', 0): 'black', ('Air', 0): 'red',
                 ('InP', 1): 'green', ('Air', 1): 'blue'}

for index in [0, 1]:
    for cover in ['InP', 'Air']:
        for pol in ['TE', 'TM']:
            m = cover_mapping[cover][mode_mapping[pol, index]]
            m.neff.plot(
                fig=ax,
                label=rf'$\mathrm{{{pol}}}-{{{index}}}$ {cover} cover',
                color=color_mapping[cover, index],
                linestyle=ls_mapping[pol],
                points=False,
            )

ax.fill_between(thicknesses.m, InP.index(wl), 3.16,
               color='skyblue', alpha=1.0)
ax.set_xlim(0.0, 1.1)
ax.set_ylim(3.16, 3.32)
ax.grid(which='major', color='gray', linestyle='-')

# Place legend outside the plot (right side)
ax.legend(bbox_to_anchor=(1.05, 0.5), loc='center left');

fig.tight_layout()
```



It is also possible to save the results after the simulation, for instance, as a CSV file, using the third-party library pandas:

```
import pandas as pd

# Dictionary of lists of effective mode indices:
data = {'Thickness': thicknesses.m,
        'TE0_InP': modes_InP[0].neff(),
        'TM0_InP': modes_InP[1].neff(),
        'TE1_InP': modes_InP[2].neff(),
        'TM1_InP': modes_InP[3].neff(),
        'TE0_Air': modes_Air[0].neff(),
        'TM0_Air': modes_Air[1].neff(),
        'TE1_Air': modes_Air[2].neff(),
        'TM1_Air': modes_Air[3].neff(),
        }

df = pd.DataFrame(data)
df.to_csv('slab_neff_vs_thickness.csv', index=False)
```

E.2.5 Mode Profiles and Propagation Loss in Lossy Waveguides

To evaluate the mode profiles and propagation loss in lossy slab waveguides with the added layers of InGaAs and metal, both materials must be defined with negative imaginary parts in the refractive index (as VPIdeviceDesigner follows the $\exp(+j\omega t)$ sign convention), which are responsible for the material loss:

```
InGaAs = Dielectric(n=3.7434-0.287j, name='InGaAs', color='#FF0000')
Ti = Dielectric(n=3.7-4.5j, name='Ti', color='#FFFF00')
```

At this stage, we will define a new function `slab_with_metal` where we include the additional layers. This means 1.5 μm of InP as buffer, 300 nm of a lossy InGaAs layer as precontact and 200 nm of metallic Ti are added on top as shown below:

```
def create_slab_with_metal(thickness='1.5 um'):
    thickness = length2default(thickness)
```

```

substrate = HalfPlane(InP)
core = Layer(InGaAsP_Q125, height=0.5, bottom=substrate.center)
buffer = Layer(InP, height=thickness, bottom=core.top)
precontact = Layer(InGaAs, height=0.3, bottom=buffer.top)
contact = Layer(Ti, height=0.2, bottom=precontact.top)
wg = Layout2D(Air, substrate, core, buffer, precontact, contact)

mesh = Mesh2D(wg)
mesh.box = wg.box.expand(0, ('5 um', 0))
mesh.update(dy='5 nm')
mesh.set_boundary('top', 'PEC')
mesh.set_boundary('bottom', 'PEC')

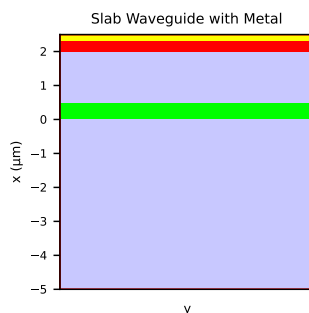
return wg, mesh

```

```

wg, mesh = create_slab_with_metal()
fig, ax = plt.subplots()
mesh.plot(fig=ax, grid=False)
ax.set_xlabel('y')
ax.set_xticks([])
ax.set_ylabel('x (um)')
ax.set_title('Slab Waveguide with Metal');

```



Now that we have defined a new waveguide layout function, a new mode solver must be defined to calculate the first five modes supported by this lossy waveguide:

```

solver = SolverModeFDM(create_slab_with_metal)

modes = solver.calc(wl, nmodes=4, thickness=1.5, target=3.6-1.0j)
extra_modes = solver.calc(wl, nmodes=1, thickness=1.5, target=3.0-0.2j)

```

In this example, a special target value was used in order to find a particular TM mode, whose effective index is well below the substrate effective index.

The effective mode indices of all modes are complex-valued, with the imaginary part responsible for mode attenuation (expressed in units of dB/cm):

```

for m in modes:
    print(f'neff[{m.name}] = {m.neff()}')
    print(f'    --> attenuation = {m.attenuation().to("dB_per_cm")}')

m = extra_modes[0]
print(f'neff[{m.name}] = {m.neff()}')
print(f'    --> attenuation = {m.attenuation().to("dB_per_cm")}')

```

```

neff[0] = (3.527652385430866-0.9824401141800838j)
--> attenuation = 345914.3382846412 dB_per_cm
neff[1] = (3.34514626538487-0.27988687320505123j)
--> attenuation = 98547.36298108425 dB_per_cm
neff[2] = (3.267725340560425-2.1910422839055134e-06j)
--> attenuation = 0.7714596857879495 dB_per_cm
neff[3] = (3.2615625552132514-4.09046578237306e-06j)
--> attenuation = 1.440241236043605 dB_per_cm
neff[0] = (2.989843820753025-0.18182901099276885j)
--> attenuation = 64021.47175251148 dB_per_cm

```

The mode profiles can be visualized as follows:

```

fig, ax = plt.subplots()

for m in modes:
    if m.E.x.max_abs() > m.E.y.max_abs():
        field = abs(m.E.x) / m.E.x.max_abs()
        label = f'mode {m.name} (TE)'
    else:
        field = abs(m.E.y) / m.E.y.max_abs()
        label = f'mode {m.name} (TM)'

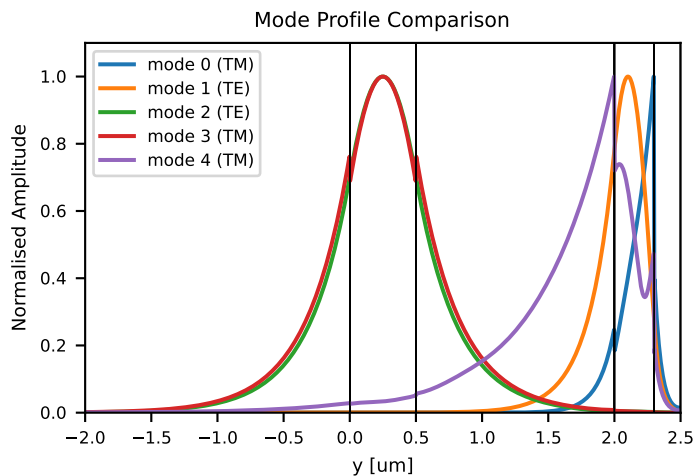
    field.slice(x=0).plot(fig=ax, edges=True,
                          box=field.box.modify(ymin=-2),
                          edges_linewidth=0.5,
                          label=label,
                          )

field = extra_modes[0].E.y
field = abs(field) / field.max_abs()
label = f'mode 4 (TM)'

field.slice(x=0).plot(fig=ax, edges=True,
                      box=field.box.modify(ymin=-2),
                      edges_linewidth=0.5,
                      label=label,
                      )

ax.set_ylim(ymin=0)
ax.set_title('Mode Profile Comparison')
ax.set_ylabel('Normalised Amplitude');

```



The first mode (mode 0) is a highly lossy plasmonic mode. The second mode (mode 1) corresponds to the fundamental TE_0 mode supported by the lossy InGaAs layer. The next two modes correspond (modes 2 and 3) to the fundamental TE_0 and TM_0 modes supported by the InGaAsP/Q_{1.25} layer, and they have a quite moderate loss around 1 dB/cm. The extra mode (mode 4) corresponds to the fundamental TM_0 mode in the lossy InGaAs layer. It shows additional plasmonic behavior in the metal layer.

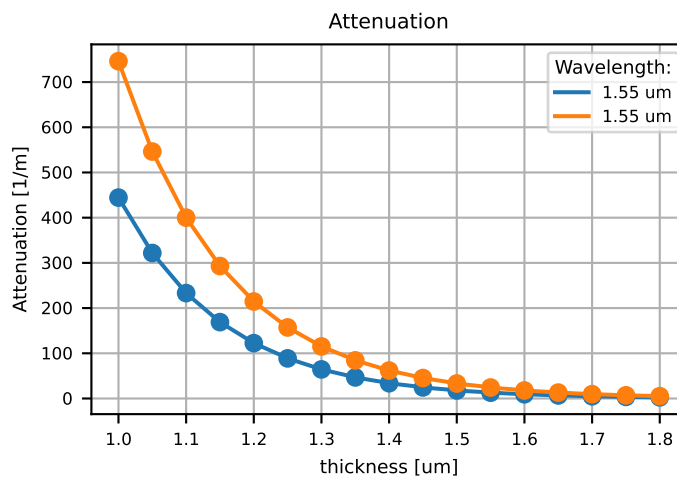
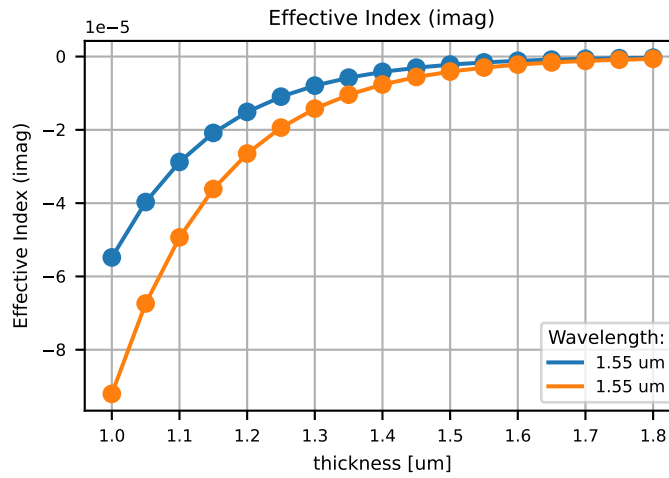
The evanescent tails of the mode 2 and 3 reach the lossy InGaAs and metal layers. Hence, the modes' absorption will depend on the thickness of the InP buffer layer. We can investigate that with a parameter sweep:

```
buffer_thicknesses = np.linspace(1.0, 1.8, 17)*u.um
modes_vs_thickness = solver.calc(wl, nmodes=2,
                                thickness=buffer_thicknesses,
                                target=3.26)
# Note: with `target=3.26`, and `nmodes=2`, we select specifically
# the TE0 and TM0 mode in the Q1.25 layer.
```

Modes calculation: 100%|=====| 17/17 [00:01<00:00, 16.12point/s]

```
fig, ax = plt.subplots()
for m in modes_vs_thickness:
    m.neff.imag.plot(fig=ax)

fig, ax = plt.subplots()
for m in modes_vs_thickness:
    m.attenuation.plot(fig=ax)
```



Finally, all results can be arranged in a publication-ready figure:

```

from vpipda.layout import Box1D

fig, axes = plt.subplots(2, 1, height_ratios=[1, 0.7])

ax = axes[0]
for m in modes:
    if m.E.x.max_abs() > m.E.y.max_abs():
        field = abs(m.E.x) / m.E.x.max_abs()
        label = 'TE, '
    else:
        field = abs(m.E.y) / m.E.y.max_abs()
        label = 'TM, '
    label += rf'$n_{\mathrm{{eff}}} = $' + \
        rf'${m.neff().real:.3f}{m.neff().imag:+.3f}i'

    # Shift the field values to plot modes separately:
    shifted_field = field - 1.5 * m.name

```

```

shifted_field.slice(x=0).plot(fig=ax,
                             label=label,
                             rescale=False)

color = fig.axes[0].get_legend_handles_labels()[0][-1].get_color()
ax.text(2.55, -1.5 * m.name, label, color=color,
        ha='left', va='center')

if m.name in [2, 3]:
    scaled_field = 100*field - 1.5*m.name
    scaled_field.slice(x=0).plot(fig=ax,
                                color=color,
                                linestyle='dotted',
                                linewidth=1,
                                rescale=False,
                                box=Box1D(1.8, 2.5))

    ax.text(1.75, scaled_field(0, 1.8).m, r'$\times 100$',
            ha='right', va='top', color=color,)

# ... and the extra mode:
m = extra_modes[0]
field = abs(m.E.y) / m.E.y.max_abs()
label = 'TM, ' + rf'$n\mathrm{{eff}} = $' + \
        rf'{m.neff().real:.3f}{m.neff().imag:+.3f}i'
shifted_field = field - 1.5*4
shifted_field.slice(x=0).plot(
    fig=ax,
    title='Mode Profiles for Lossy Slab Waveguide',
    label=label,
    rescale=False,
    edges=True,
    edges_linestyle(':',
    edges_linewidth=1,
)
color = fig.axes[0].get_legend_handles_labels()[0][-1].get_color()
ax.text(2.55, -1.5 * 4, label, color=color,
        ha='left', va='center')

ax.axvspan(-0.5, 0.0, facecolor=InP.color, alpha=0.5)
ax.axvspan(0.0, 0.5, facecolor=InGaAsP_Q125.color, alpha=0.5)
ax.axvspan(0.5, 2.0, facecolor=InP.color, alpha=0.5)
ax.axvspan(2.0, 2.3, facecolor=InGaAs.color, alpha=0.3)
ax.axvspan(2.3, 2.5, facecolor=Ti.color, alpha=0.3)

ax.set_xlim(-0.5, 2.5)
ax.set_ylim(-6.4, 1.3)
ax.set_xticks(ticks=[], labels=[])
ax.set_yticks(ticks=[], labels=[])
ax.set_xlabel('$x$')
ax.set_ylabel('Normalised mode profiles')

ax.get_legend().remove()

```

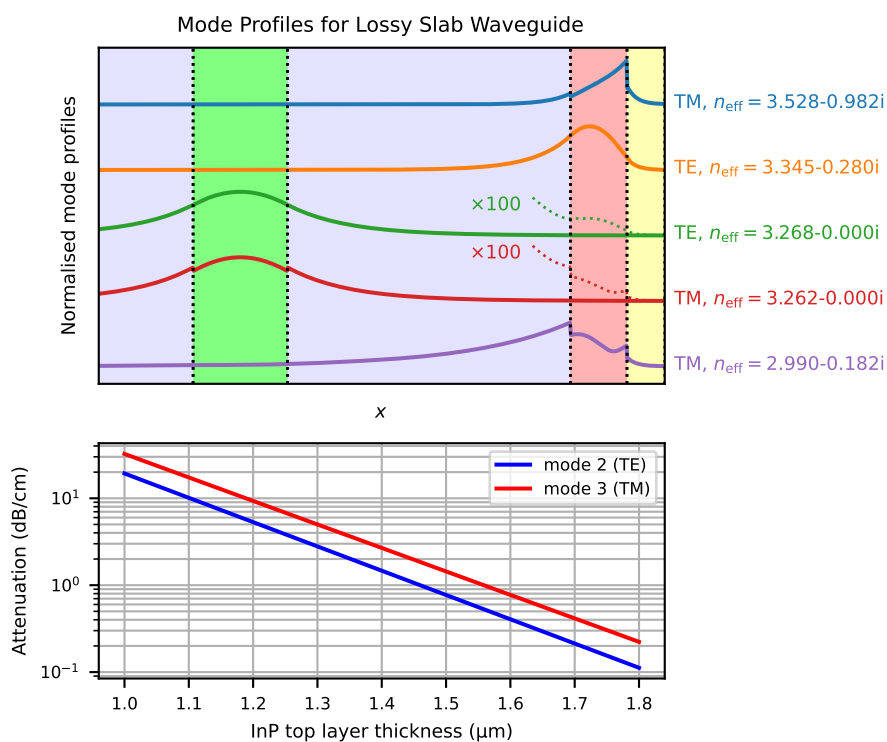
```

a_TE = modes_vs_thickness[0].attenuation().to('dB_per_cm')
a_TM = modes_vs_thickness[1].attenuation().to('dB_per_cm')

ax = axes[1]
ax.plot(buffer_thicknesses.m, a_TE.m, label='mode 2 (TE)', color='blue')
ax.plot(buffer_thicknesses.m, a_TM.m, label='mode 3 (TM)', color='red')
ax.set_yscale('log')
ax.set_xlabel('InP top layer thickness (um)')
ax.set_ylabel('Attenuation (dB/cm)')
ax.grid(which='both')
ax.legend()

fig.tight_layout()

```



E.2.6 Mode Profiles in Asynchronous Coupled Waveguides

In this scenario, the objective is to have a vertical slab waveguide. For this, first we defined a function that generates the cross-section for a slab waveguide uniform along the vertical axis later, we use it to create our desired layout:

```

def create_vertical_slab(width='500 nm'):
    core = Layer(InGaAsP_Q125, height=width, rotation='90 deg')
    return Layout2D(InP, core)

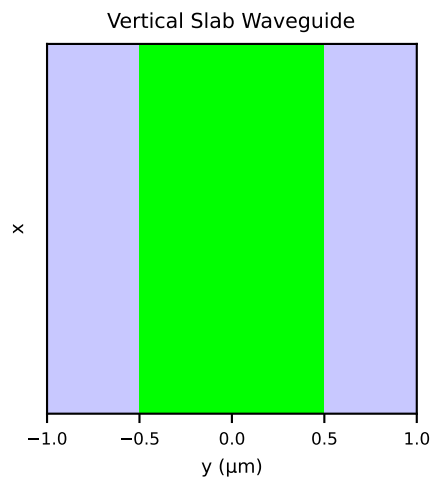
```

```

# Verify the cross-section layout:
wg = create_vertical_slab(width='1 um')
fig, ax = plt.subplots()

```

```
wg.plot(fig=ax)
ax.set_xlabel('y (um)')
ax.set_ylabel('x')
ax.set_yticks([])
ax.set_title('Vertical Slab Waveguide');
```



Based on this cross-section generating function, another function can be defined to create a directional coupler made of two straight parallel waveguides with the widths `width1` and `width2` and a gap between them:

```
def create_coupled_waveguides(width_1, width_2, gap):
    width_1 = length2default(width_1)
    width_2 = length2default(width_2)
    gap = length2default(gap)

    wg_1 = create_vertical_slab(width_1).shift(-(width_1+gap)/2)
    wg_2 = create_vertical_slab(width_2).shift(+(width_2+gap)/2)

    wg = wg_1.merge(wg_2)

    mesh = Mesh2D(wg)
    mesh.box = wg.box.expand('3 um')
    mesh.update(dx='5 nm')

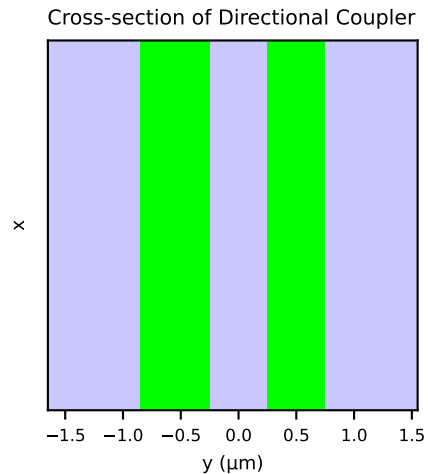
    return wg, mesh
```

Now an asynchronous (with different values for `width_1` and `width_2`) directional coupler can be created and verified:

```
# Device sizes in default units (um):
width_1 = 0.6
width_2 = 0.5
gap = 0.5

# Create the directional coupler:
wg, mesh = create_coupled_waveguides(width_1=width_1, width_2=width_2, gap=gap)
```

```
fig, ax = plt.subplots()
wg.plot(fig=ax)
ax.set_xlabel('y (um)')
ax.set_ylabel('x')
ax.set_yticks([])
plt.title('Cross-section of Directional Coupler');
```



The first four modes of the corresponding cross-section can be calculated as follows:

```
# Create and configure mode solver:
solver = SolverModeFDM(wg, mesh)
```

```
# Calculate the first 4 modes:
modes = solver.calc(wl, nmodes=4)
```

```
# Verify effective mode indices and mode types:
for m in modes:
    mode_type = 'TE' if m.E.x.max_abs() > m.E.y.max_abs() else 'TM'
    print(f'neff[{m.name}] = {m.neff()} ({mode_type} mode)')
```

```
neff[0] = 3.2863550826602177 (TM mode)
neff[1] = 3.2813544405816373 (TE mode)
neff[2] = 3.262496926505805 (TM mode)
neff[3] = 3.2560065860718534 (TE mode)
```

Here, the first two modes correspond to the even TM and TE super-modes supported by the coupled waveguides, while the next two modes correspond to the odd TM and TE super-modes.

Let's plot the E_x fields of the even and odd TM modes (mode 0 and mode 2, respectively):

```
fig, ax = plt.subplots()
wg.plot(fig=ax, aspect='auto')
```

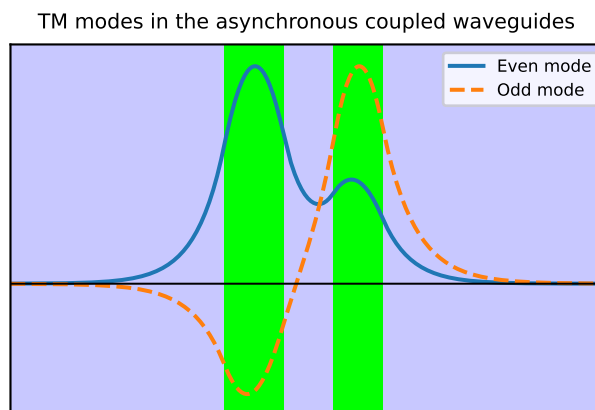
```

(-modes[0].E.y.slice(y=0)).plot(fig=ax, label='Even mode')
(-modes[2].E.y.slice(y=0)).plot(fig=ax,
                                title='TM modes in the asynchronous '
                                      'coupled waveguides',
                                label='Odd mode',
                                linestyle='--')

ax.axhline(0, linewidth=0.75, color='black')

ax.set_xlim(-3.0, 3.0)
ax.set_ylim(-0.6, 1.1)
ax.set_xticks(ticks=[], labels=[])
ax.set_yticks(ticks=[], labels=[])
ax.set_xlabel('')
ax.set_ylabel('');

```



E.3 Modeling Channel Waveguides

E.3.1 Library Setup and Material Definitions

Some of the libraries were imported previously. In principle, we could skip re-importing them. However, for the sake of demonstration, all the necessary modules for constructing the shallow- and deep-etched waveguides are imported at once, as follows:

```

import numpy as np
import matplotlib.pyplot as plt

from vpipda.spectral import wavelength
from vpipda.material import Dielectric, Air
from vpipda.layout import HalfPlane, Layer, Rectangle, \
    Layout2D, Box2D, length2default
from vpipda.mesh import Mesh2D
from vpipda.mode import SolverModeFDM

from vpipda import units as u

```

The material of the waveguide and the substrate are similar to the previous scenario. Here, we add a linear correction for the wavelength dependency of the refractive indices around 1.55 μm and define also all the doped materials according to the data given in table B.1 and B.2:

```
n_InP = 3.17 - 0.12*(wavelength/(1*u.um) - 1.55)
InP = Dielectric(n=n_InP, color='#c8c8ff')

InP_n3e17 = Dielectric(n=n_InP-0.002, color='#7d7dff')
InP_n5e17 = Dielectric(n=n_InP-0.003, color='#7d7dff')
InP_p3e17 = Dielectric(n=n_InP-0.002, color='#7d7dff')
InP_p5e17 = Dielectric(n=n_InP-0.003, color='#7d7dff')

InP_n1e18 = Dielectric(n=n_InP-0.007, color='#0000ff')
InP_p1e18 = Dielectric(n=n_InP-0.007, color='#0000ff')

n_Q125 = 3.36 - 0.23*(wavelength/(1*u.um) - 1.55)
Q125 = Dielectric(n=n_Q125, color='#00ff00')
Q125_n6e16 = Dielectric(n=n_Q125 - 0.0, color='#00ff00')
Q125_p1e17 = Dielectric(n=n_Q125 - 0.0006, color='#00ff00')
```

wavelength is VPIdeviceDesigner-specific object that allows to define simple wavelength-dependent analytical expressions.

After defining the materials, it is time to define the geometry and the meshing. To make it compact and usable later on for a parameter sweep, the structure is defined as a function/method that would allow to construct both shallow and deep-etched waveguides based on the structure specification. Using a boolean, if set to True, it will construct a deep-etched waveguide and if set to False, it will construct a shallow one.

E.3.2 Waveguide Cross-Section Setup

We define a waveguide-generating function that is able to create deep- or shallow-etched waveguides with different widths. If no width is given, then the respective default standard values will be used.

```
def create_channel(width=None, deep=True):
    if width is None:
        width = 1.5 if deep else 2.0
    width = length2default(width)

    sub_0 = HalfPlane(InP)
    sub_1 = Layer(InP, height=0.4, bottom=sub_0.center)
    if deep:
        sub_2 = Rectangle(InP, w=width, h=0.1, bottom=sub_1.top)
        Q_0 = Rectangle(Q125, w=width, h=0.2, bottom=sub_2.top)
        Q_1 = Rectangle(Q125, w=width, h=0.2, bottom=Q_0.top)
    else:
        sub_2 = Layer(InP, h=0.1, bottom=sub_1.top)
        Q_0 = Layer(Q125, h=0.2, bottom=sub_2.top)
        Q_1 = Layer(Q125, h=0.2, bottom=Q_0.top)
```

```

Q_2 = Rectangle(Q125, w=width, h=0.1, bottom=Q_1.top)

rib_0 = Rectangle(InP, w=width, h=0.2, bottom=Q_2.top)
rib_1 = Rectangle(InP, w=width, h=0.3, bottom=rib_0.top)
rib_2 = Rectangle(InP, w=width, h=1.0, bottom=rib_1.top)

wg = Layout2D([Air,
               sub_0, sub_1, sub_2,
               Q_0, Q_1, Q_2,
               rib_0, rib_1, rib_2])

mesh = Mesh2D(wg)

if deep:
    mesh.box = wg.box.expand(1.5, (2.0, 0.5))
else:
    mesh.box = wg.box.expand(4.0, (2.0, 0.5))

mesh.update(dx=0.02, dy=0.02)

return wg, mesh

```

To get a feeling about the structures, we can use matplotlib to plot both structures together:

```

wg_deep, mesh = create_channel(width=1.5, deep=True)
wg_shallow, mesh = create_channel(width=2, deep=False)

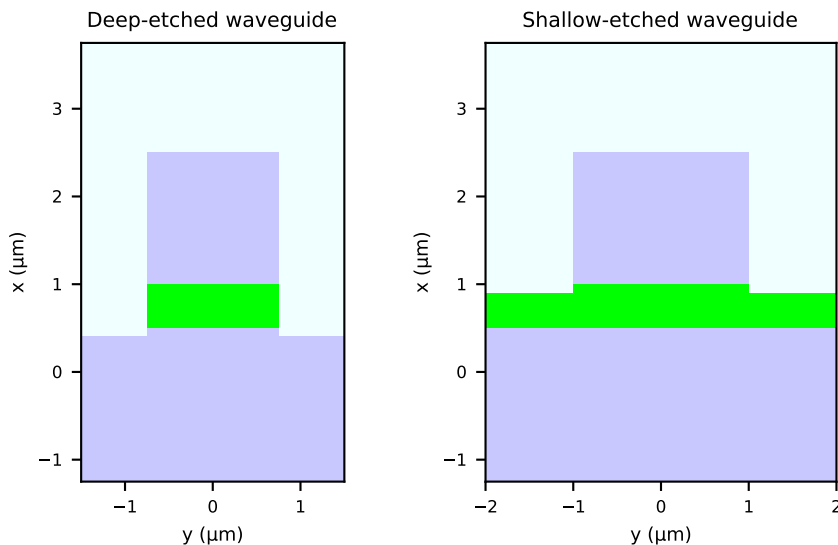
# Plot both structures together using matplotlib:
fig, axes = plt.subplots(1, 2)
wg_deep.plot(fig=axes[0])
axes[0].set_title('Deep-etched waveguide')

wg_shallow.plot(fig=axes[1])
axes[1].set_title('Shallow-etched waveguide')

for ax in axes:
    ax.set_xlabel('y (um)')
    ax.set_ylabel('x (um)')

fig.tight_layout()

```



It is also possible to define more specific methods that would provide us with e.g., a deep-etched and a shallow-etched waveguide.

E.3.3 Deep-Etched Waveguide Modeling & Analysis

Previously, we checked how the structure layout looks like. In addition to that, it is also reasonable to investigate how fine we have defined our mesh. Box2D allows us to define the area in which we would like to investigate our mesh.

```
solver = SolverModeFDM(create_channel)
```

At this point, we can proceed with calculating the cross-sections of the guided modes and their corresponding effective indices:

```
# Operating wavelength
wl = 1.55 * u.um

# Calculate the guided modes
modes_inset_deep = solver.calc(freq=wl, nmodes=2, deep=True)

# Plot the guided modes and their corresponding effective indices
fig, axes = plt.subplots(1, 2)

plot_box = Box2D((-1.5, -0.2), (1.5, 2.8))
modes_inset_deep[1].E.real.x.plot(
    fig=axes[0], box=plot_box, contour=False, cbar=False,
    title=r'$\mathrm{TE}_{\{00\}}$ mode, $E_y$'
)
modes_inset_deep[0].E.real.y.plot(
    fig=axes[1], box=plot_box, contour=False, cbar=False,
    title=r'$\mathrm{TM}_{\{00\}}$ mode, $E_x$'
)
```

```

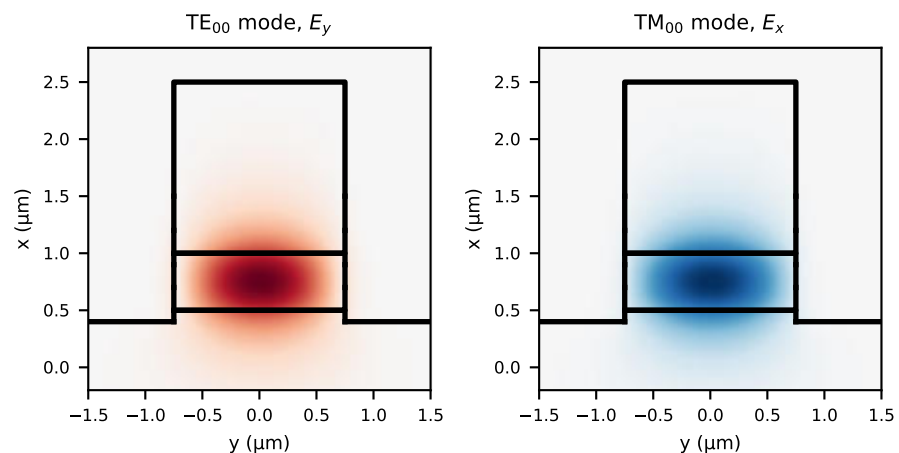
for ax in axes:
    ax.set_xlabel('y (um)')
    ax.set_ylabel('x (um)')
fig.tight_layout()

print(f'n_eff_TE00 = {modes_inset_deep[1].neff(wl)}')
print(f'n_eff_TM00 = {modes_inset_deep[0].neff(wl)}')

```

n_eff_TE00 = 3.2282979439173807

n_eff_TM00 = 3.2284255388502094



To investigate the impact of waveguide width on the effective indices of the TE and TM modes, we can conduct a parameter sweep:

```

# Define the range of the waveguide widths
widths_deep = np.linspace(1.0, 4.0, 31)

# Configure the parameter sweep
solver.set_sweep_parameter('width', 'waveguide width', 'um')

# Calculate the guided modes for each configuration
modes_deep = solver.calc(freq=wl, nmodes=2, width=widths_deep,
                        deep=True, sorter='mode_overlap')

```

Modes calculation: 100%|=====| 31/31 [02:25<00:00, 4.68s/point]

To check the evolution of effective index, let us plot the sweep results using matplotlib features:

```

# Apply fitting to the calculated points
modes_deep[0].neff.set_fitting('width', 'cubic')
modes_deep[1].neff.set_fitting('width', 'cubic')

# Generate the graph
fig, ax = plt.subplots()

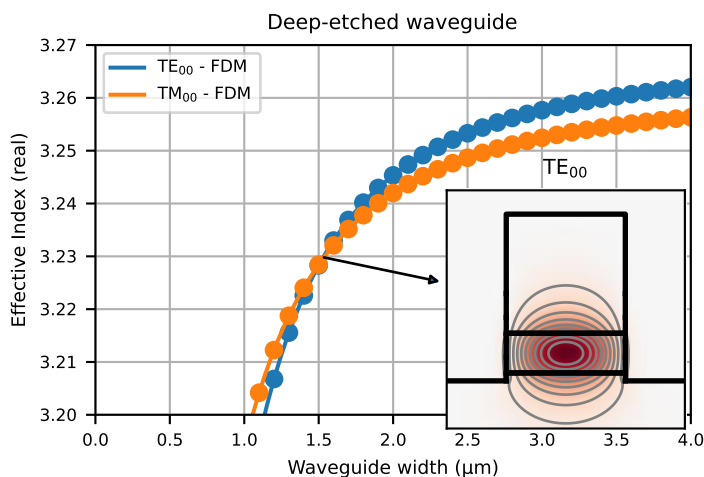
modes_deep[1].neff.plot(fig=fig, axis='width',

```

```

        label=r'$\mathrm{TE}_{00}$ - FDM')
modes_deep[0].neff.plot(fig=fig, axis='width',
        label=r'$\mathrm{TM}_{00}$ - FDM')
ax.set_xlim(0, 4)
ax.set_ylim(3.20, 3.27)
ax.legend(loc='upper left')
plt.title("Deep-etched waveguide")
ax.set_xlabel('Waveguide width (um)')
# adding the cross-section to the plot for reference
inset_ax = ax.inset_axes([0.59, -0.29, 0.4, 1.15])
modes_inset_deep[1].E.real.x.plot(
    fig=inset_ax, contour=True,
    levels=np.linspace(-1, 1, 21),
    levels_to_label=None,
    levels_linewidths=1,
    box=plot_box, edges_color='black',
    edges_linewidth=2,
    edges_linestyle='solid', cbar=False,
    title=r'$\mathrm{TE}_{00}$'
)
inset_ax.set_xlabel('')
inset_ax.set_xticks([])
inset_ax.set_ylabel('')
inset_ax.set_yticks([])
plt.annotate('', xy=(2.35, 3.225), xytext=(1.5, 3.23),
    arrowprops={'arrowstyle': '-|>'});

```



E.3.4 Shallow-Etched Waveguide Modeling & Analysis

Now we will repeat the same process for the shallow-etched waveguide:

```

# Calculate the guided modes
modes_inset_shallow = solver.calc(freq=wL, nmodes=2, deep=False)

# Plot the guided modes and their corresponding effective indices
fig, axes = plt.subplots(1, 2)

```

```

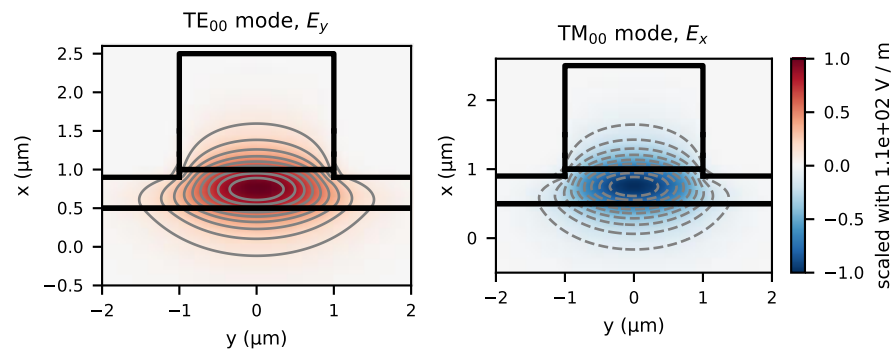
modes_inset_shallow[0].E.real.x.plot(
    fig=axes[0],
    contour=True, levels=np.linspace(-1, 1, 21), levels_linewidths=1,
    levels_to_label=None, box=Box2D((-2, -0.5), (2, 2.6)), cbar=False,
    title=r'$\mathrm{TE}_{00}$ mode, $E_y$',
)
modes_inset_shallow[1].E.real.y.plot(
    fig=axes[1],
    contour=True, levels=np.linspace(-1, 1, 21), levels_linewidths=1,
    levels_to_label=None, box=Box2D((-2, -0.5), (2, 2.6)),
    title=r'$\mathrm{TM}_{00}$ mode, $E_x$',
)
for ax in axes:
    ax.set_xlabel('y (um)')
    ax.set_ylabel('x (um)')
fig.tight_layout()

print(f'n_eff_TE00 = {modes_inset_shallow[0].neff(wl)}')
print(f'n_eff_TM00 = {modes_inset_shallow[1].neff(wl)}')

```

n_eff_TE00 = 3.252061582850388

n_eff_TM00 = 3.2461244008745025



Let us check the the behaviour of the effective indices with respect to the waveguide width variation:

```

widths_shallow = np.linspace(0.25, 4.0, 16)
modes_shallow = solver.calc(freq=wl, nmodes=2,
                             width=widths_shallow[:-1], deep=False)

```

Modes calculation: 100%|=====| 16/16 [01:43<00:00, 6.49s/point]

```

modes_shallow[0].neff.set_fitting('width', 'cubic')
modes_shallow[1].neff.set_fitting('width', 'cubic')

fig, ax = plt.subplots()
modes_shallow[0].neff.plot(fig=fig, axis='width', label='TE - FDM')
modes_shallow[1].neff.plot(fig=fig, axis='width', label='TM - FDM')
ax.set_xlim(0.0, 4.0)

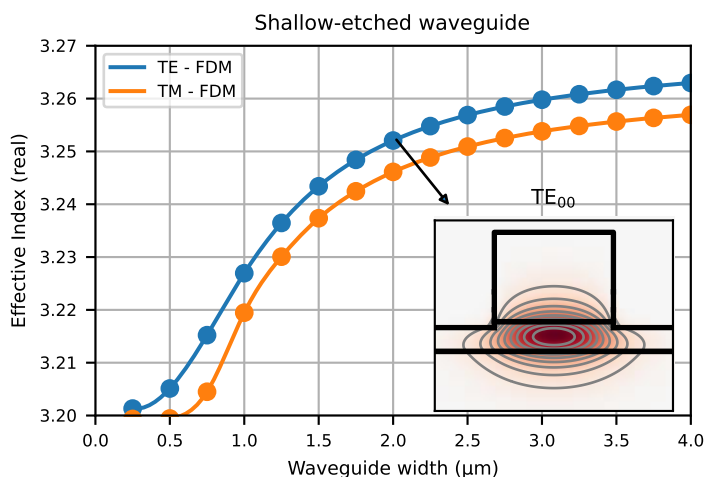
```

```

ax.set_ylim(3.20, 3.27)
ax.legend(loc='upper left')
ax.set_xlabel("Waveguide width (um)")
plt.title('Shallow-etched waveguide')

inset_ax = ax.inset_axes([0.57, -0.23, 0.4, 1])
modes_inset_shallow[0].E.real.x.plot(
    fig=inset_ax, contour=True,
    levels=np.linspace(-1, 1, 21), levels_linewidths=1,
    levels_to_label=None,
    density=True, box=Box2D((-2, -0.5), (2, 2.7)),
    edges_color='black', edges_linewidth=2,
    edges_linestyle='solid', cbar=False,
    title=r'\mathrm{TE}_{00}'
)
inset_ax.set_xlabel("")
inset_ax.set_xticks([])
inset_ax.set_ylabel("")
inset_ax.set_yticks([])
plt.annotate('', xy=(2.4, 3.239), xytext=(2, 3.253),
    arrowprops={'arrowstyle': '->'});

```



Finally, we plot the results for the deep- and shallow-etched waveguides side-by-side for a better comparison:

```

fig, ax = plt.subplots(1, 2)

# Shallow-etched waveguide
modes_shallow[0].neff.set_fitting('width', 'cubic')
modes_shallow[1].neff.set_fitting('width', 'cubic')

modes_shallow[0].neff.plot(fig=ax[0], axis='width',
    label=r'\mathrm{TE}_{00}$ - FDM',
    points=False)
modes_shallow[1].neff.plot(fig=ax[0], axis='width',
    label=r'\mathrm{TM}_{00}$ - FDM',
    points=False)

```

```

ax[0].set_xlim(0.0, 4.0)
ax[0].set_ylim(3.20, 3.27)
ax[0].legend(loc='upper left', fontsize='small')
ax[0].set_xlabel("Waveguide width (um)")
ax[0].set_title('Shallow-etched waveguide')

# Add the cross-section to the plot for reference
inset_ax_shallow = ax[0].inset_axes([0.57, -0.23, 0.4, 1])
modes_inset_shallow[0].E.real.x.plot(
    fig=inset_ax_shallow, contour=True,
    levels=np.linspace(-1, 1, 21), levels_linewidths=1,
    levels_to_label=None,
    density=True, box=Box2D((-2, -0.5), (2, 2.7)),
    edges_color='black', edges_linewidth=2,
    edges_linestyle='solid', cbar=False,
    title=r'$\mathrm{TE}_{00}$'
)
inset_ax_shallow.set_xlabel("")
inset_ax_shallow.set_xticks([])
inset_ax_shallow.set_ylabel("")
inset_ax_shallow.set_yticks([])
ax[0].annotate('', xy=(2.5, 3.238), xytext=(2, 3.253),
    arrowprops={'arrowstyle': '-|>'})

# Deep-etched waveguide
# Apply fitting to the calculated points
modes_deep[0].neff.set_fitting('width', 'cubic')
modes_deep[1].neff.set_fitting('width', 'cubic')

modes_deep[1].neff.plot(fig=ax[1], axis='width',
    label=r'$\mathrm{TE}_{00}$ - FDM',
    points=False)
modes_deep[0].neff.plot(fig=ax[1], axis='width',
    label=r'$\mathrm{TM}_{00}$ - FDM',
    points=False)
ax[1].set_xlim(0.0, 4.0)
ax[1].set_ylim(3.20, 3.27)
ax[1].legend(loc='upper left', fontsize='small')
ax[1].set_title("Deep-etched waveguide")
ax[1].set_xlabel('Waveguide width (um)')
ax[1].set_ylabel('')

# Add the cross-section to the plot for reference
inset_ax_deep = ax[1].inset_axes([0.57, -0.26, 0.4, 0.83])
modes_inset_deep[1].E.real.x.plot(
    fig=inset_ax_deep, contour=True,
    levels=np.linspace(-1, 1, 21), levels_linewidths=1,
    levels_to_label=None,
    box=plot_box, edges_color='black',
    edges_linewidth=2,
    edges_linestyle='solid', cbar=False,
    title=r'$\mathrm{TE}_{00}$'
)
inset_ax_deep.set_xlabel("")
inset_ax_deep.set_xticks([])
inset_ax_deep.set_ylabel("")

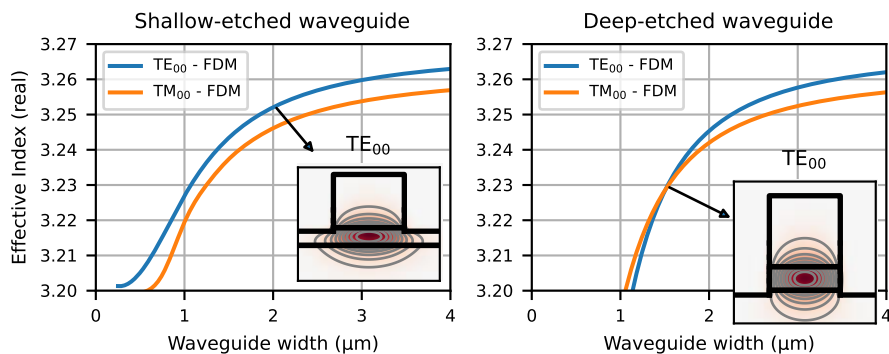
```

```

inset_ax_deep.set_yticks([])
ax[1].annotate(' ', xy=(2.3, 3.22), xytext=(1.5, 3.23),
               arrowprops={'arrowstyle': '-|>'})

plt.tight_layout()

```



E.3.5 Bending Loss Analysis

By convention, bent waveguides are bent to the left. Therefore, we expect outgoing radiation to the right and bottom (i.e. predominantly into the substrate).

To allow for the outgoing radiation, we need to set the right and bottom boundary conditions to “PML” (“perfectly matched layers”, a non-reflecting, absorbing layer that emulates free space) for the left bend and increase the simulation domain in order to make some room for the outgoing radiation:

```

def create_channel_pml(width=None, deep=True):
    wg, mesh = create_channel(width=width, deep=deep)

    mesh.box = mesh.box.expand((0, 4), (4, 0))
    mesh.set_boundary('bottom', 'PML', depth=10, damping=1e-8)
    mesh.set_boundary('right', 'PML', depth=10, damping=1e-8)
    mesh.update(dx=0.05, dy=0.05)

    return wg, mesh

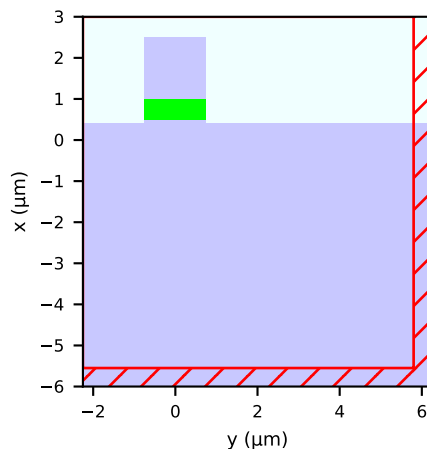
```

Let us have a look at the waveguide cross-section including the PML layers at bottom and right areas:

```

wg, mesh_PML = create_channel_pml(deep=True)
fig, ax = plt.subplots()
mesh_PML.plot(fig=ax, grid=False)
ax.set_xlabel('y (um)')
ax.set_ylabel('x (um)');

```



E.3.5.1 Bending Loss in Deep-Etched Waveguides

The idea is to calculate the bending loss for various bend radii. Therefore, we will first prepare a valid range for the bending radius:

```
bend_radii_deep = np.linspace(80, 10, 15) * u.um
print(bend_radii_deep)
```

```
[80.0 75.0 70.0 65.0 60.0 55.0 50.0 45.0 40.0 35.0 30.0 25.0 20.0 15.0
10.0] micrometer
```

Now it is time to proceed with creating a solver which will be used later on via the `sim.calc(...)` method to calculate the guided modes.

```
solver_pml = SolverModeFDM(create_channel_pml)
```

```
modes_bent_deep = solver_pml.calc(freq=wl, nmodes=2, deep=True,
                                bend_radius=bend_radii_deep,
                                target=3.23, sorter='mode_overlap')
```

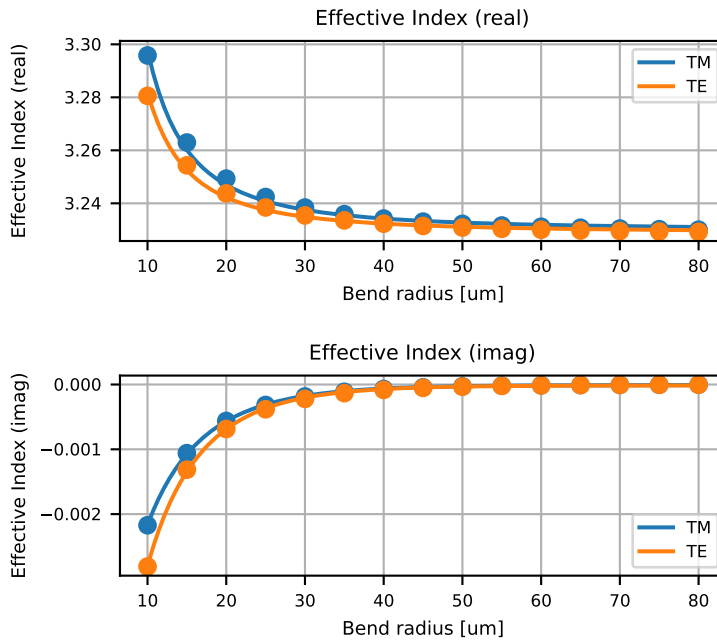
```
Modes calculation: 100%|=====| 15/15 [00:51<00:00, 3.45s/point]
```

Here we show how the effective indices of the guided modes depend on the bending radius:

```
fig, axes = plt.subplots(2, 1)
```

```
for m in modes_bent_deep:
    label = 'TM' if m.name == 0 else 'TE'
    m.neff.real.plot(fig=axes[0], label=label, legend_loc='upper right')
    m.neff.imag.plot(fig=axes[1], label=label, legend_loc='lower right')
```

```
fig.align_ylabels()
plt.tight_layout(h_pad=2.0)
```



It is also possible to calculate the loss (in decibel) over a 90° for each bend radius and print them in the form of a list that could be used later on for post processing:

$$a = -10 \log_{10}(e) \frac{4\pi}{\lambda_0} \operatorname{Im}(n_{\text{eff}}) \frac{\pi}{2} R$$

```
a_TE_deep = -10 / np.log(10) * (4 * np.pi / wl) * \
    modes_bent_deep[1].neff.imag() * (np.pi / 2) * bend_radii_deep

a_TM_deep = -10 / np.log(10) * (4 * np.pi / wl) * \
    modes_bent_deep[0].neff.imag() * (np.pi / 2) * bend_radii_deep
```

We can easily represent these data in a table using the third-party library pandas:

```
import pandas as pd

data_deep = {
    'Bending radius (um)': bend_radii_deep.m,
    'TE 90° bending loss (dB)': a_TE_deep.m,
    'TM 90° bending loss (dB)': a_TM_deep.m,
}

df = pd.DataFrame(data_deep)
print(df.to_string(index=False))
```

Bending radius (um)	TE 90° bending loss (dB)	TM 90° bending loss (dB)
80.0	0.012592	0.010267
75.0	0.017468	0.014475
70.0	0.024472	0.020068
65.0	0.034032	0.027662
60.0	0.046989	0.039061

55.0	0.065959	0.055038
50.0	0.092702	0.076248
45.0	0.129080	0.108785
40.0	0.183643	0.152065
35.0	0.257717	0.217605
30.0	0.368435	0.306198
25.0	0.528419	0.436656
20.0	0.759676	0.622180
15.0	1.090335	0.878760
10.0	1.553760	NaN

E.3.5.2 Bending Loss in Shallow-Etched Waveguides

We repeat the same calculations for the shallow-etched waveguide:

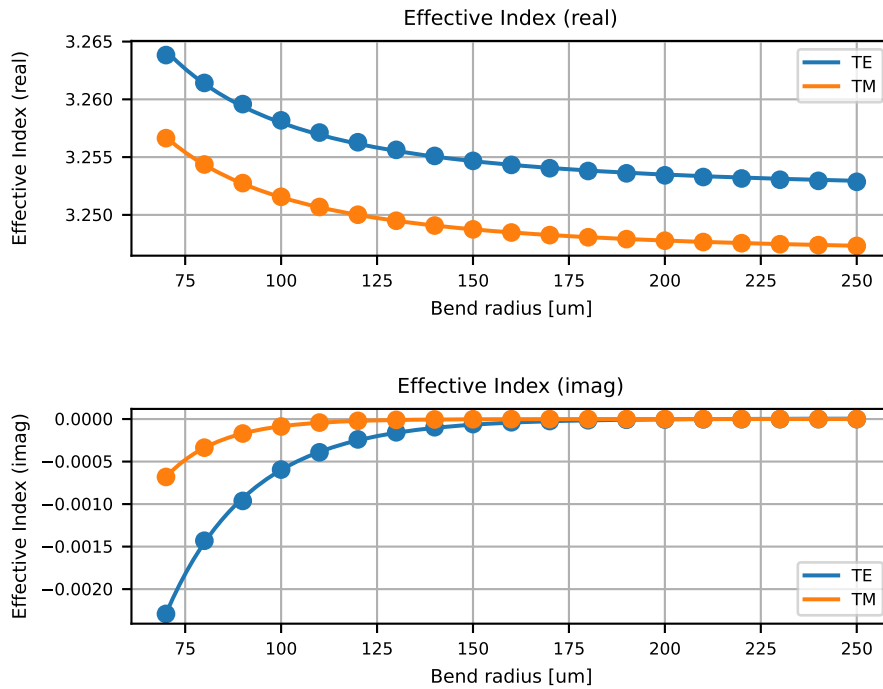
```
bend_radii_shallow = np.linspace(250, 70, 19) * u.um
print(bend_radii_shallow)
```

```
[250.0 240.0 230.0 220.0 210.0 200.0 190.0 180.0 170.0 160.0 150.0 140.0
130.0 120.0 110.0 100.0 90.0 80.0 70.0] micrometer
```

```
modes_bent_shallow = solver_pml.calc(wl, nmodes=2, deep=False,
                                     bend_radius=bend_radii_shallow,
                                     target=3.25)
```

```
Modes calculation: 100%|=====| 19/19 [02:37<00:00, 8.28s/point]
```

```
fig, axes = plt.subplots(2, 1)
for m in modes_bent_shallow:
    label = 'TE' if m.name == 0 else 'TM'
    m.neff.real.plot(fig=axes[0], label=label, legend_loc='upper right')
    m.neff.imag.plot(fig=axes[1], label=label, legend_loc='lower right')
fig.align_ylabels()
plt.tight_layout(h_pad=3.0)
```



```
a_TE_shallow = -10 / np.log(10) * (4 * np.pi / wl) * \
    modes_bent_shallow[0].neff.imag() * (np.pi / 2) * bend_radii_shallow

a_TM_shallow = -10 / np.log(10) * (4 * np.pi / wl) * \
    modes_bent_shallow[1].neff.imag() * (np.pi / 2) * bend_radii_shallow
```

```
import pandas as pd
```

```
data_shallow = {
    'Bending radius (um)': bend_radii_shallow.m,
    'TE 90° bending loss (dB)': a_TE_shallow.m,
    'TM 90° bending loss (dB)': a_TM_shallow.m,
}
```

```
df = pd.DataFrame(data_shallow)
print(df.to_string(index=False))
```

Bending radius (um)	TE 90° bending loss (dB)	TM 90° bending loss (dB)
250.0	0.006242	0.000018
240.0	0.009674	0.000034
230.0	0.015200	0.000071
220.0	0.024274	0.000146
210.0	0.038623	0.000288
200.0	0.059781	0.000573
190.0	0.090665	0.001198
180.0	0.139562	0.002501
170.0	0.219087	0.004862
160.0	0.333842	0.009506
150.0	0.491077	0.019569

140.0	0.745977	0.037301
130.0	1.121226	0.071181
120.0	1.589623	0.140231
110.0	2.380047	0.252279
100.0	3.286792	0.484887
90.0	4.791896	0.847583
80.0	6.331502	1.489771
70.0	8.871344	2.636111

Finally, we plot all the bending losses over the 90° arc in one plot:

```
fig, ax = plt.subplots()

ax.plot(bend_radii_shallow.m, a_TE_shallow.m,
        label=r'$\mathrm{TE}_{00}$, shallow',
        color='#1f78b4', linestyle='-', linewidth=2)
ax.plot(bend_radii_shallow.m, a_TM_shallow.m,
        label=r'$\mathrm{TM}_{00}$, shallow',
        color='#e31a1c', ls='--', linewidth=2)

ax.plot(bend_radii_deep.m, a_TE_deep.m,
        label=r'$\mathrm{TE}_{00}$, deep',
        color='green', ls='-', linewidth=2)
ax.plot(bend_radii_deep.m, a_TM_deep.m,
        label=r'$\mathrm{TM}_{00}$, deep',
        color='#ff7f00', ls='--', linewidth=2)

# Set labels and title
ax.set_xlabel('Bending Radius (um)')
ax.set_ylabel('Bending Loss (dB/90°)')

ax.minorticks_on()
ax.set_yscale('log')
ax.grid(which='major')
ax.grid(which='minor', linestyle=':', linewidth=1)

ax.set_xlim(0, 250)
ax.set_ylim(0.01, 2)

ax.legend()
fig.tight_layout()
```

