User Guide v-1

Process Simulator User Guide

SemiVi LLC

Zollikon, Switzerland.

September 1, 2026

Contents

Chapter 1

Introduction

The process solver provided with the SemiVi package simulates various steps in the semiconductor fabrication process flow. The process solver uses the structure generator and mesher to create an initial semiconductor structure (e.g. bare Silicon wafer) together with the mesh. Fabrication process steps in the process flow such as implantation, annealing, etching, and deposition are simulated to create the final device structure. The initial device structure as well as the process steps in the process flow are given as input to the simulator in a config file. Alternately, a python interface is provided to create the process flow in a python file. The structure can be saved and visualized after any of the steps by providing appropriate instructions.

The process simulation generates various files which store spatial distribution of as-implanted ions or active dopant atoms at any step specified by the user. The quantities stored in these files can be visualized in paraview.

1.1 Features

The process solver has the following features.

- The process solver can simulate etching, deposition, implantation, and annealing steps on the 2D/3D structures.

- Gas flows can be specified in the annealing step to create oxidizing environment. In such conditions, the process solver also simulates the Silicon oxidation step.

- Implant tables are provided to extract analytic implant parameters at the given implant energy and tilt. The use of implant tables allows the users to simulate implantation at various energy values without explicitly providing the required parameters.

- Enhancement of dopant diffusivity due to the defects and the electric field can be included in the dopant diffusion by using a three-stream model. Simpler and faster models for the dopant diffusion are also provided.

1.2 Installation

SemiVi currently supports software installation on various Linux distributions. The software installer is available in Debian package (*.deb file) and in RPM format (*.rpm file).

Note, that *if you have downloaded mkl version of the ProcessSolver*, the following package needs to be installed manually by you before installing the circuit solver from the installer package.

- Intel math kernel libraries (released in 2020 or later), which include distributions of open-mp, pardiso, etc. specific for Intel processors.

The ProcessSolver sources mkl functions from the above installation. These functions can offer speed-up in the calculations on Intel processors. The mkl package can be downloaded from Intel website.

If the ProcessSolver without mkl-acceleration is downloaded, then installation of the above package is not necessary.

Once `Intel math kernel libraries` are installed, download the installer on the local machine. The installer file named `prosolver_amd64.deb` will appear in the `Downloads` directory. Go to the directory using `cd` command. Use the following command to install the `prosolver` from the installer.

```
>> sudo apt install ./prosolver_amd64.deb
```

Alternately, one may use `dpkg` to install the software and use `apt` to install missing dependencies as follows.

```
>> sudo dpkg -i ./prosolver_amd64.deb
>> sudo apt install -f
```

You need to have `root` access to install the software on your machine.

1.3 Licensing

Two types of licenses can be purchased for SemiVi process solver.

Node-locked licenses enable unlimited number of *simultaneous* executions of the process solver on the client machine. The node-locked license limits the usage of the process solver only to the machine on which the license is activated.

With *server licenses*, the process solver can be run on any of the machines in the client organization on which the server license is activated. However, only the specified number of *simultaneous* executions are possibie at a time.

1.3.1 Purchasing the licenses

The clients can place order for any of the above licenses on SemiVi website (https://www.semivi.ch/sales) or by contacting our sales-person.

We will process the request and send the license files by email. The license files need to be activated on the desired machines using the license key which is emailed separately using the following command.

1.3.2 Installation of SemiVi-activator

The license file must be activated on the desired computer before use. For that purpose, download the installer `semivila_amd64.deb` file on the local machine and install it as follows.

```
>> sudo apt install ./semivila_amd64.deb
```

1.3.3 License activation

To activate the license file, please run the following command.

```
>> semivila -a File.lic <Server|NodeLocked>License.lic\\
```

Replace `File.lic` with the your license file, and use appropriate name for the activated file. You will be prompted to input the 16 digit license key. A successful activate of the license file will generate the activated license file. Copy the activated license file to the `/opt/semivi/licenses/` folder and rename it to `ServerLicense.lic` or `NodeLockedLicense.lic` for server and node-locked licenses respectively. If you have more than one license files, please delete the older expired license files. If you wish to keep more than one active license files, you can also name the license files as `<i>NodeLockedLicense.lic` where `<i>` could be from 0 to 49. For ex. `49NodeLockedLicense.lic` or `49ServerLicense.lic`. The program will read the license files and lock the first available license. All the target users must have read rights on the license file.

User-guides of all the software provided by SemiVi are stored at the location `/opt/semivi/userguides/`.

Tutorials of all the software provided by SemiVi are stored at the location `/opt/semivi/tutorials/prosolver`.

Chapter 2

Process Simulator Config File

The process simulator reads inputs on the sequence of process steps, materials, meshing from the configuration file and performs process simulations to create a device structure. The program can be executed using the following command –

```
prosolver --pro example_pro.cfg
```

In the above command, the word after `prosolver` is the name of the program to be executed (in this case "–pro"). The program name is followed by the configuration file name.

2.1 Example config file

A sample configuration file of the process simulator is provided below.

```
Device: {
  Name = "Diode";
  MeshType = "FEM";
  Simulation = "PRO";
```

```
}

RefWin*RefAll: {
  Position: ([-0.3, -0.2, 0.], [0.3, 1., 0.]);
  Shape = "Rectangle";
}

RefWin*RefSi1: {
  Position: ([-.3, -0.2, 0.], [0.3, 0.2, 0.]);
  Shape = "Rectangle";
}

Region*RegBack: {
  RefWin = ["RefAll"];
  Material = "Gas";
}

Region*RegSi1: {
  RefWin = ["RefSi1"];
  Material = "Silicon";
}

MeshDef*MDefAll: {
  RefWin = "RefAll";
  X: {cen = 0., minspacing = 0.02, maxspacing = 0.02, incr = 1. }
  Y: {cen = 0., minspacing = 0.02, maxspacing = 0.02, incr = 1. }
}

MeshDef*MDefSi1: {
  RefWin = "RefSi1";
  X: {cen = 0., minspacing = 0.02, maxspacing = 0.02, incr = 1. }
  Y: {cen = 0., minspacing = 0.005, maxspacing = 0.005, incr = 1. }
}

RefWin*polydep1: {
  Position = ([-0.2, -0.1, 0.], [-0.16, -0.3, 0.],
              [0.16, -0.3, 0.], [0.2, -0.10, 0.]);
  Shape = "Polygon";
```

```
}

Save*sav0: {
  Quantities = ["BActive"];
}

// PROCESS STEPS BEGIN
Deposit*dep1: {
  Material = "Oxide";
  OnMaterial = "Silicon";
  Type = "Anisotropic";
  Thickness = 0.4;
  InSituDopantConc = 1E10; // 1/cm3
  InSituDopantSpecies = "Boron";
}

Save*sav1: {
  Quantities = ["BActive"];
}

RefWin*polyetMaskWin: {
  Position = ([-0.2001, 0., 0.], [0.2001, 0., 0.]);
  Shape = "Segment";
}

Mask*polyetMask: {
  RefWinList = ["polyetMaskWin"];
}

Etch*et1: {
  Material = "Oxide";
  Type = "Masked";
  Mask = "polyetMask";
  Depth = 0.5;
}

Save*sav2: {
  Quantities = ["BActive"];
```

```
}

Implant*impl1: {
  Species = "Boron";
  Energy = 90;
  Table = "Taurus";
  Dose = 1E12;
}

Save*sav3: {
  AsImplanted  = ["Boron"];
}

Etch*et2: {
  Material = "Oxide";
  Type = "Strip";
}

SetParam*s0: {
  Command = "Silicon BActive IntNeutral
             Dstar 0 Arrhenius 0.0121 3.341";
}

SetAnnealParam*s1: {
  initstep = 1E-4; minstep = 1E-7; maxstep = 10.0;
  Pressure = 1.; TolRes = 1.0; TimeStepping = "BE";
  MaxIterations = 10; MaxInnerIterations = 10;
  UndampedIterations = 0;
}

Anneal*ann1: {
  Temperature = [800., 1000., 1000., 800.];
  TimeIntervals = [400., 30., 400.];
  N2 = [20., 20., 20.];
}

Save*sav4: {
  Quantities = ["BActive"];
```

```
}
```

The above config file consists of various sections including those used in the structure generator command file. In each of the section names, string before '*' gives the object type, whereas the string after '*' specifies the section name. The section types `RefWin`, `Region`, and `MeshDef` have been described in the structure generator manual. They will not be described here. These steps create an initial structure on which the process simulations are performed. The semiconductor structure can increase in size during the process simulations. To enable creation of new regions, meshing must be performed

The rest of the section types are unique to the process simulator. These process steps are simulated in exactly the same order as written in the file. They are described below.

2.2 Device section

There can be only one `Device` section in the file. `Name` sets the device name. `MeshType` sets that internal 'FEM' meshing engine to be used. `Simulation` provides the type of simulation to be performed using the device. For process simulations, it is set to `PRO`.

2.3 Mask object

Various masked etching processes are typically used in semiconductor fabrication. These masks are instantiated by the `Mask` object in the config file. In 2D process simulations, a mask is composed of multiple horizontal segments. These segments are defined as `RefWin` objects. A mask is composed of a group of `RefWin` objects whose names are listed as a provided separated list to the argument `RefWinList`.

2.4 Deposit object

The `Deposit` object simulates deposition of a specific material in the process simulations. The material to be deposited is set by `Material`. For a selective deposition process, the material on which the deposition

is performed is specified by `OnMaterial`. The argument `Type` specifies that an anisotropic deposition is to be performed. Thickness of the deposited material layer (in μm) is set by `Thickness`.

2.5 Etch object

The `Etch` object simulates etching with the specified settings. When `Material` is set, selective etching of the specified material is performed. The `Type` sets the etch type, whether isotropic, anisotropic, etc. When the argument `Mask` is set, a masked etch is performed. Etch depth (in μm) is set by the argument `Depth`.

2.6 Implant object

The `Implant` object simulates analytic implantation with the specified settings. Implant species is set by `Species`, energy of the implants is set by `Energy` (in keV), and dose of the implanted ions is set by `Dose` (in cm^{-2}). Various parameters necessary for analytic implantation process are obtained from the implantation table type specified by `Table`.

2.7 Anneal object

The `Anneal` object simulates annealing step in the process simulation. During annealing, implanted dopant atoms diffuse which modifies net dopant concentration. This is modeled in the process simulator by evolving the doping concentrations on the finite element grid using time-stepping. The simulator takes temperature profile during annealing as input with the arguments `Temperature` and `TimeIntervals`. Notice, that the number of entries in `Temperature` list is larger than `TimeIntervals` by a single entry. Temperature (in $\deg C$) at the beginning of `Anneal` process is set as the first entry. Temperature values at the end of each subsequent time intervals (in seconds) are listed in `TimeIntervals`.

In the presence of oxidizing gases (O_2, H_2O, or N_2O), Silicon surface is oxidized creating a layer of Silicon oxide on the top. The

oxidation rate is determined by the partial pressure of oxidizing gases (and H_2O). The gas flows are specified by setting the gas name as an argument followed by the gas flow rates during each of the time intervals.

2.8 SetAnnealParam object

The numeric parameters required for solving diffusion equations in annealing process can be updated in `SetAnnealParam` object. The parameters modified by this object are stored by the simulator and are used for all subsequent annealing steps, unless the parameters are modified again. The parameter `initstep` sets time-step (in sec) at the beginning of annealing process. If the time-step is less than `minstep`, then the diffusion process is terminated and an error is returned. Maximum step for the time-stepping during annealing process is set by `maxstep`.

2.9 SetParam object

Various diffusion, oxidation, or other parameters can be set by specifying `Command` in `SetParam` object. For example, the command

```
Command = "Silicon BActive IntNeutral Dstar 0
                  Arrhenius 0.0121 3.341";
```

sets `Dstar` parameter of `Silicon` material, corresponding to zero charged Boron-interstitial complex (`BActive` and `IntNeutral`) to `Arrhenius 0.0121 3.341`. Here, the text `Arrhenius <A> <Ea>` specifies Arrhenius function with a prefactor of `A` and an activate energy of `Ea`. Thus, `Arrhenius 0.0121 3.341` sets the diffusion parameter `Dstar` to $0.0121 \times \exp\left(\frac{-3.341}{kT/q}\right)$.

2.10 Save object

The `Save` object saves spatial distributions of the specified quantities to an hdf file. An xdmf file is also stored to view the quantities in paraview.

Names of the as-implanted implant species to be saved are specified with the argument `AsImplanted`. Names of the physical quantities to be saved such as dopant concentrations, interstitial concentrations, etc. are provided with the argument `Quantities`.

The hdf file is named '<step>_<name>_pro.h5', where <step> is set by `Name` argument in `save` function and <name> is the name of the process object. Also, an xdmf script file named '<step>_<name>_pro.xdmf' is stored for visualizing the data with paraview.

2.11 Running process simulations

The process steps in the above config file can be simulated in the simulator using the following command.

```
prosolver --pro example_pro.cfg
```

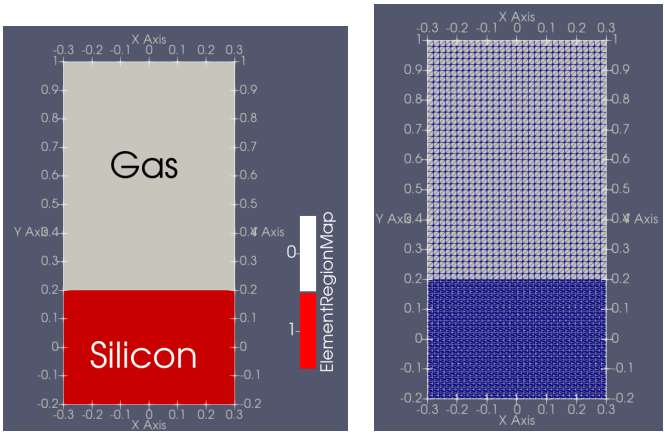
There are in total five `Save` objects in the above config file, which are named `sav0`, `sav1`, ... , and `sav4`. Each of the object saves the structure file after a process step. Output structures at these steps are explained below.

Initial structure

Structure at the beginning of the process (in this case, bare wafer) is stored by the step named `sav0`. It can be viewed in paraview as follows.

```
>> paraview sav0_Diode_pro.xdmf
```

Opening the above file and looking at `ElementRegionMap` reveals the initial structure as shown in Fig. 2.1(a). Also, selecting `Surface With Edges` instead of `Surface` shows generated mesh as in Fig. 2.1(b). Since implantation and doping in Silicon region is of importance, it shows refined mesh in y- direction. This mesh remains the same throughout the process simulations.



(a) Initial structure (b) Mesh

Figure 2.1: Structure and mesh at the beginning of the process flow. Region-0 corresponds to the vacuum (or Gas) and Region-1 refers to Silicon wafer.

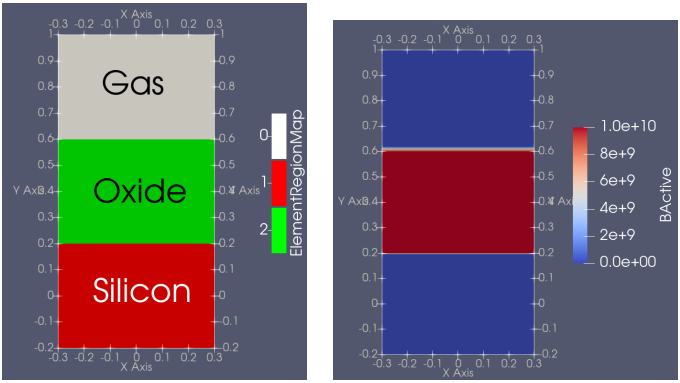
Deposition of oxide

Structure generated by deposition of oxide on the bare wafer is stored by the step named `sav1`. It can be viewed in paraview as follows.

```
>> paraview sav1_Diode_pro.xdmf
```

Fig. 2.2(a) shows the generated structure after the deposition step. Here, Region-2 corresponds to the new region of oxide. `InSituDopantSpecies` and `InSituDopantConc` is specified in the deposition step. This setting defines uniform dopant concentration in the deposited material as shown in Fig. 2.2(b).

In-situ doping during deposition can be used to mimic epitaxial growth step. Thermal diffusion of the existing dopants during epitaxy may be simulated separately by adding `Anneal` step after the deposition.



(a) Structure after oxide deposition

(b) Doping concentration

Figure 2.2: (a) Structure generated after deposition of oxide on the wafer. (b) *In-situ* doping of the oxide results in B dopants in the oxide.

Masked etch of oxide

Masked etching of oxide is performed after the deposition step. The resulting structure can be viewed using the following command.

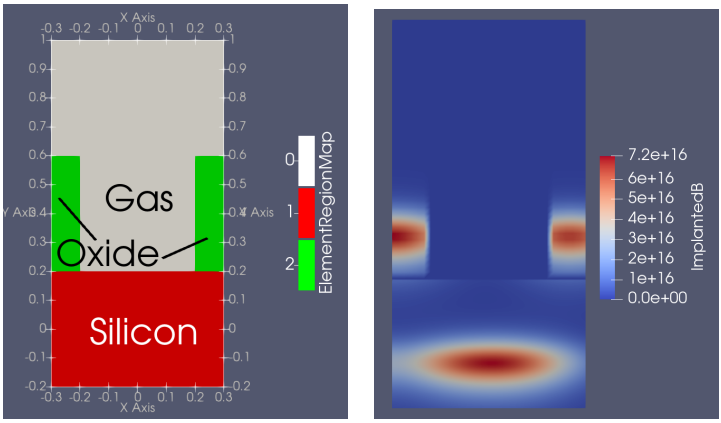
```
>> paraview sav2_Diode_pro.xdmf
```

Fig. 2.3(a) shows the structure generated by masked etch of oxide. This structure is used for masked implantation of `Boron` into Silicon wafer as described below.

Boron implantation

Masked implantation of Boron can be carried out by using an oxide hard-mask. Distribution of Boron atoms after implantation is shown in Fig. 2.3(b). It can be viewed in paraview as follows.

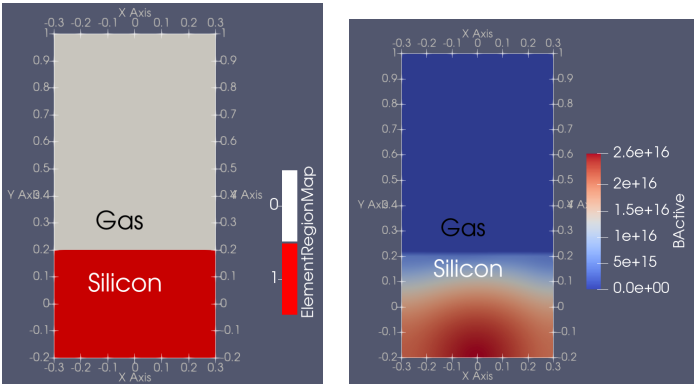
```
>> paraview sav3_Diode_pro.xdmf
```



(a) Masked etch of oxide

(b) Oxide-masked Boron implantation

Figure 2.3: (a) Structure generated after masked etch of oxide. (b) As-implanted Boron distribution obtained by masked implantation of Boron.



(a) Strip etch of oxide (b) Strip etch of oxide

Figure 2.4: (a) Structure generated after strip etch of oxide. (b) Active Boron concentration after the annealing step in the process flow.

Strip etch of oxide

Oxide mask used for implantation is stripped away before performing subsequent process steps. The structure obtained by strip etch is shown in Fig. 2.4(a).

Annealing Silicon

Inert annealing is performed to diffuse dopant atoms to the desired location, such as active regions. Distribution of active B atoms at the end of `Anneal` step is viewed as follows.

```
>> paraview sav4_Diode_pro.xdmf
```

Annealing of Silicon wafer causes diffusion active B atoms (dataset name - `BActive`). Concentration of active B atoms after annealing step is shown in Fig. 2.4(b). Inert annealing of 30 minutes at 1000 degC together with the ramp up and ramp down processes redistributes implanted B atoms from Fig. 2.3(b) to Fig. 2.4(b).

Chapter 3

Process Simulator Python Interface

Parametrization of simulation scripts enables reuse of the script and saves development time. The process simulator offers python interface to the necessary steps to enable parametrization of the simulation script. Python interface of the process simulator is explained in this chapter. If the process simulator python interface is installed in the computer, a python file containing such an interface can be executed just like any other python file, using the command below.

```
>> python trial_pro.py
```

Note, that the process simulator python interface requires python 3 (version 3.7 or above) installed.

3.1 Python file

A python script file equivalent to the config script in Chapter 2 is provided below.

```
import process as pr
import numpy as np
```

```
# define process object
dioPro = pr.Process (Name="Diode", ProcessStepsFile="",
                     MeshType="FEM")

xmin = -0.24
xmax = 0.24
ymin = -0.2
ySiTop = 0.2
ymax = 1.0
zmax = 0
zmin = 0
posRef = [pr.position(xmin, ymin, zmin),
          pr.position(xmax, ymax, zmax)]
meshPar = {"MaxAreaBulk" : 0.01, "MaxAreaInterface" : 0.005}
# define RefWins
dioPro.setRefWin (Name="RefAll", Shape="Rectangle", Position=posRef,
                  NumericParams=meshPar, NumericParamsList={})

posRef = [pr.position(xmin, ymin, zmin),
          pr.position(xmax, ySiTop, zmax)]
dioPro.setRefWin (Name="RefSi", Shape="Rectangle", Position=posRef,
                  NumericParams={}, NumericParamsList={})

# define Regions
dioPro.setRegion (Name="RegBack", RefWin="RefAll", Material="Gas",
                  NumericParams=meshPar)

dioPro.setRegion (Name="RegSi", RefWin="RefSi", Material="Silicon",
                  NumericParams=meshPar)

# define Meshing
dioPro.setMeshDef (Name="MDefAll", RefWin="RefAll",
                   Xarg=[0., 0.01, 0.01, 1.],
                   Yarg=[0., 0.01, 0.01, 1.],
                   Zarg=[0., 0.01, 0.01, 1.])

dioPro.setMeshDef (Name="MDefSi", RefWin="RefSi",
                   Xarg=[0., 0.01, 0.01, 1.],
```

```
                    Yarg=[-0.075, 0.002, 0.01, 1.3],
                    Zarg=[0., 0.01, 0.01, 1.])

# create Mesh
dioPro.createDeviceMesh ()

dioPro.saveMeshData ()

# DO NOT START PROCESS BEFORE CREATING DEVICE MESH
dioPro.save (Name="Sav0", PhysicalQuantities=["BActive"])

# start process steps
dioPro.deposit (Name="DepOx", Material="Oxide",
                Type="Anisotropic", Thickness=0.4)

dioPro.save (Name="Sav1", PhysicalQuantities=["BActive"])

# define RefWin for Mask
arrTmp=np.array ([[xmin*0.6, 0., 0.], [xmax*0.6, 0., 0.]])
dioPro.setRefWin (Name="OxEtMaskWin", Shape="Segment",
                  Position=arrTmp)

# define Mask for etching oxide
dioPro.mask (Name="OxEtMask", RefWinList=["OxEtMaskWin"])

# etch oxide
dioPro.etch (Name="EtOx", Material="Oxide", Type="Masked",
             Depth=0.5, EtchMask="OxEtMask")

dioPro.save (Name="Sav2", PhysicalQuantities=["BActive"])

# implant Boron
dioPro.implant (Name="Impl1", ImplantSpecies="Boron",
                Energy=90, Dose=1E12)

dioPro.save (Name="Sav3", AsImplantedSpecies=["Boron"])

# strip oxide
```

```
dioPro.etch (Name="StripOx", Material="Oxide", Type="Strip")

# set global parameters
dioPro.setParam (Name="s0", Material = "Silicon",
     Dopant = "BActive", Defect = "IntNeutral",
     ParamName = "Dstar", Charge = "0",
     ParamValue = "Arrhenius 0.0121 3.341")

# set numeric parameters for annealing step.
# create two python maps, one with string parameters and
#        one with numeric parameters
numPar = { "initstep" : 1E-4, "minstep" : 1E-7, "maxstep" : 10.0,
      "Pressure" : 1., "RelativeError" : 1.E0, "TolRes" : 1.0E-1,
      "TolStep" : 1.0E-1, "MaxIterations" : 15,
       "MaxInnerIterations" : 5, "UndampedIterations" : 0}
strPar = { "TimeStepping" : "BE"}
dioPro.setAnnealPar (Name="s1", NumericParMap=numPar,
                     StringParMap=strPar)

# perform annealing
TempProfile = [800., 950., 950., 800.]
TimeIntervals = [20., 5., 20.]
Gases = {"O2" : [0., 20., 0.],"N2" : [20., 20., 20.]}
dioPro.anneal (Name="ann1", Temperature=TempProfile,
               Time=TimeIntervals, GasFlows=Gases)

dioPro.save (Name="Sav4", PhysicalQuantities=["BActive"])

# The following command begins simulation
dioPro.doProcessing ()

dioPro.saveDevice ()
```

First line of the python file `import process as pr` imports process library which has the object called `Process`. The 'Process' object contains all the procedures (python equivalent of functions) that are used to create and mesh a Semiconductor structure, and perform process simulations on the structure. Once a 'Process object' is

instantiated, these procedures can be invoked on the instance as shown on subsequent lines of the file.

All the python functions required to define the semiconductor structure and mesh are described in Structure generator and mesher user guide. `createDeviceMesh` function uses the defined regions, refwins, and mesh definitions to mesh the region of interest. Process simulation can be performed only after creating the device mesh. Following functions add various process simulation steps to the setup.

3.2 Initialization

`dioPro = pr.Process(...)` line instantiates a process object 'dioPro'. This is similar to `Device` section in config file and is described in Sec. 2.2. It takes the structure name as an input. A process config file containing initial device structure and/or process steps can be optionally provided with the argument `ProcessStepsFile`. If such a file is provided, then it is parsed to read the objects. `MeshType` specifies the mesh engine used throughout the simulation.

The functions `setRefWin`, `setParam`, `setMeshDef`, `createDeviceMesh`, and `saveMeshData` are described in Structure Generator and Mesher user guide.

3.3 Implantation step

`dioPro.implant (...)` procedure is invoked on 'dioPro' object and adds an implantation step to the process flow. The procedure takes inputs similar to the `Implant` object in config file as described in Sec. 2.6. The step name is set by arguments `Name`, implantation species are set by `ImplantSpecies`, implantation energy (in keV) is set by `Energy`, and implantation dose (in cm^{-2}) is set by the argument `Dose`.

3.4 Anneal step

`dioPro.anneal (...)` procedure is invoked on 'dioPro' object and adds an annealing step to the process flow. This procedure takes

similar inputs to the `Anneal` object in config file as described in Sec. 2.7. Temperature profile at the end of each time interval is set as a python list of numbers with the argument `Temperature`. Temperature at the beginning of the anneal process is set at the beginning of the list `Temperature`. Time intervals are set as list of numbers with the argument `Time`. Gas flows are set as a map of gas name to the list of values of gas flow (see variable `Gases`). The map is specified with the argument `GasFlows`.

3.5 Mask definition

`dioPro.mask (...)` procedure is invoked on 'dioPro' object and adds a mask to the process flow. This mask can be used in other steps, e.g. etching. This procedure is similar to the `Mask` object in config file as described in Sec. 2.3. Name of the mask is set by the argument `Name`. List of RefWins which compose the mask is set by the argument `RefWinList`.

3.6 Etching step

`dioPro.etch (...)` procedure is invoked on 'dioPro' object and adds an etch step to the process flow. This procedure takes similar inputs to the `Etch` object in config file as described in Sec. 2.5. In this procedure, material to be etched is set by the argument `Material`, etch type is set by `Type`, etch depth (in μm) is set by `Depth`. If the etch is masked etch, then etch mask is set with the argument `EtchMask`.

3.7 Deposition step

`dioPro.deposit (...)` procedure is invoked on 'dioPro' object and adds a deposit step the process flow. This procedure takes similar inputs as the `Deposit` object in config file as described in Sec. 2.4. Material to be deposited is set with the argument `Material`. If the argument `OnMaterial` is specified, the selective deposition on this material takes place. Deposition step type is set with the argument `Type`, and thickness of the deposited layer is set with `Thickness`.

3.8 Setting parameters

`dioPro.setParam (...)` procedure is invoked on 'dioPro' object and updates various process parameters such as diffusivity. Name of the parameter to be updated is set by `ParamName` and its value is specified by `ParamValue`. In this case, the diffusivity parameter D^* of zero charged Boron-interstitial complex in Silicon is modified. Material of the parameter is set by `Material`, the dopant species by `Dopant`, the defect species by `Defect`. The text `Arrhenius <A> <Ea>` specifies Arrhenius function with a prefactor of `A` and an activate energy of `Ea`. Thus, `Arrhenius 0.0121 3.341` sets the diffusion parameter `Dstar` to $0.0121 \times \exp\left(\frac{-3.341}{kT/q}\right)$.

3.9 Setting annealing parameters

Various numerical and convergence related parameters of diffusion step in anneal process are set by the keyword `dioPro.setAnnealPar (...)`. Numeric parameters are specified as a map of parameter name and its numeric value. Here, variable `numPar` defines such a map. Text parameters are set as a map of parameter name and its text value. Here, variable `strPar` defines such a map. In the argument list of the function, numeric parameters are specified with the argument `NumericParMap` and the text parameters are specified with the argument `StringParMap`.

3.10 Saving data

`dioPro.save (...)` procedure is invoked on 'dioPro' object. This procedure takes arguments similar to the `Save` object described in Sec. 2.10. This procedure creates an hdf5 file and stores spatial distributions of the physical quantities list specified by the argument `Quantities` and that of as-implanted dopant list specified by `AsImplanted`. The hdf5 file is named '<step>_<name>_pro.h5', where <step> is set by `Name` argument in `save` function and <name> is the name of the process object. Also, an xdmf script file named '<step>_<name>_pro.xdmf' is stored for visualizing the data with paraview.

3.11 Additional procedures

3.11.1 doProcessing procedure

All the above functions add process steps to the process flow or add visualization/set-parameter steps. Process simulations *do not* begin until `dioPro.doProcessing (...)` procedure is called. This procedure begins process simulations by going through each of the steps in the process flow one-by-one.

3.11.2 saveMeshData procedure

`dioPro.saveMeshData (...)` procedure saves mesh data on the process grid.

3.11.3 saveDevice procedure

`dioPro.saveDevice (...)` procedure saves device mesh as well as doping information. This procedure must be invoked *after* `doProcessing` procedure.

3.12 Running simulations

Similar to the config file in Chapter 2, the above python script file is run to simulate the process flow as follows.

```
>> python example_pro.py
```

The script creates the same output as the config file given in Chapter 2. Output files and structures are described in Section 2.11.

Chapter 4

Etching and deposition

Semiconductor fabrication involves different types of etching steps in which a specific material is etched away from the wafer. Similarly, different types of deposition steps are carried out during fabrication in which a layer of a specific material is deposited on the wafer. These processes can be modeled in the process simulator by providing appropriate arguments to `etch` or `deposit` procedures (in python) or objects (in config file). Various settings available in the simulator for modeling the above processes are described below.

4.1 Deposit step

`Deposit` object in config file is configured with different arguments for different deposition types. Also, `deposit (...)` procedure in python interface can also be configured with the same arguments as explained below. The following arguments are necessary in all the deposition types.

- `Material`: Material to be deposited on the wafer is specified with this argument.

- `Thickness`: Thickness of the layer in μm is specified with this argument.

- `NumericParMap`: Various numeric parameters can be provided as python map of parameter name to its value. In this case, mesh ('TriMesh' or 'TetMesh') parameters, which are used in meshing the newly created region are supplied as numeric parameters with `NumericParMap`.

In addition to the above arguments, various additional arguments may be required depending on the specific deposition type specified by the user. Available deposition types and the additional arguments for those deposition types are listed below.

4.1.1 Types

Deposition type is specified with the argument `Type`. Available deposition types are listed below.

Isotropic

If `Type` is set to `Isotropic`, isotropic deposition is performed. The `Material` layer of specific `Thickness` is deposited on all the horizontal, vertical, and slanted surfaces.

Anisotropic

If `Type` is set to `Anisotropic`, anisotropic deposition is performed. The `Material` layer of specific `Thickness` is deposited on all the horizontal. No deposition is performed on the vertical surfaces. On slanted surface, appropriate thick layer is deposited such that vertical thickness of the layer is equal to `Thickness`. If `OnMaterial` is specified, then the `Material` is deposited only on the surfaces of `OnMaterial`.

Polygonal

If `Type` is set to `Polygonal`, `Material` layer is set in the area confined in the RefWin specified by `DepWin`. All the other arguments are ignored.

4.1.2 Selective deposition

If `OnMaterial` is specified, then the `Material` is deposited only on the surfaces of `OnMaterial` on the top of the wafer. Selective deposition can be combined with `Isotropic` and `Anisotropic` types.

4.1.3 *In-situ* doping

If `InSituDopantSpecies` and `InSituDopantConc` are specified, then the deposited material layer is *in-situ* uniformly doped with the dopant species specified by `InSituDopantSpecies`. *In-situ* dopant concentration (in cm^{-3}) is set by `InSituDopantConc`. These two arguments can be combined with any type of deposition above.

4.2 Etch step

`Etch` object in config file is configured with specific arguments for the specified etch type. Following arguments are mandatory in all the deposition type.

- `Material`: Material to be selectively etched is specified.
- `Depth`: Etch depth is specified in μm.
- `NumericParMap`: Various numeric parameters can be provided as python map of parameter name to its value. In this case, mesh ('TriMesh' or 'TetMesh') parameters, which are used in meshing the newly created region are supplied as numeric parameters with `NumericParMap`. Note, that a new region is 'typically' not created in etching process. But, sometimes topology requires creation of new region.

Various additional arguments may be required for specific etch types. Available etch types and the additional arguments for those etch types are specified below.

4.2.1 Types

Etch type is specified with the argument `Type`. Available etch types are listed below.

Isotropic

If `Type` is set to `Isotropic`, isotropic etching is performed. In this type, specified `Material` is etched by `Depth` on the horizontal, vertical, and also slanted material surfaces.

Anisotropic

If `Type` is set to `Anisotropic`, anisotropic etching is performed. In this type, specified `Material` is etched by `Depth` on the horizontal and slanted material surfaces. Material exposed on the slanted surface is etched such that the vertical etch depth is equal to `Depth`.

Polygonal

If `Type` is set to `Polygonal`, the specified material in the area confined in the RefWin specified by `EtchWin` is etched away i.e. replaced by Gas.

Masked etching

If etch `Type` is set to `Masked`, masked anisotropic etching is performed using the mask set by `Mask`.

Strip etching

If etch `Type` is set to `Strip`, all the regions of the specified material which are exposed to 'Gas' are completely etched away i.e. replaced by Gas. This etch is typically used for etching of the mask material e.g. photoresist.

Chapter 5

Ion Implantation

Ion implantation is often used in fabrication processes to introduce dopant atoms in semiconductors in a pattern to create the device structures. The implanted ions undergo multiple random collisions with Silicon atoms in the crystal lattice. This results in a peculiar shaped distribution profile of the ions perpendicular to the wafer surface. This is called as-implanted distribution of that particular dopant. The dopant distribution can be characterized by its first moment (implant depth), second moment (standard deviation, and so on. These distribution moments depend on the selected implantation energy, wafer tilt, and wafer rotation.

Ion implantation step can be included in the process simulator by specifying various input parameters such as implant energy, wafer tilt, etc. The process simulator uses this information together with the device structure to calculate as-implanted distribution of the dopant atoms in the structure. This is called analytic ion implantation technique. This chapter describes analytic ion implantation technique in the process simulator.

5.1 Analytic ion implantation

In this technique, a predefined set of implant depth, standard deviation of the implanted ion distribution profile is used to calculated as-implanted distribution of dopant atoms in the 2D structure.

5.1.1 Defining implantation parameters

An implantation step is added to the process flow in a config file by setting an implant object as follows.

```
Implant*impl1: {
  Species = "Boron";
  Energy = 90;
  Table = "Taurus";
  Dose = 1E12;
  Tilt = 0;
  Rotation = 0;
  Mode = "Quad"; // or "Dual" or "Single"
}
```

Following input parameters can be specified in the `Implant` object.

- `Species`: One of the following implant species can be specified. 1. `Boron` 2. `BF2` 3. `Phosphorus` 4. `Arsenic` 5. `Antimony`. This parameter is mandatory.
- `Energy`: Implant energy in `keV` is set. This parameter is mandatory.
- `Dose`: Implant dose in the units of cm^{-2} is set. This parameter is mandatory.
- `Tilt`: Implant tilt in `degree` is set. Default value is 0 deg.
- `Rotation`: Implant rotation in `degree` is set. Default value is 0 deg.
- `Mode`: Implantation mode is set. Following modes are available -
  - `Dual` : Dual mode - implantation is performed at `Rotation` angle and `Rotation` + 180 deg angle.

  - `Quad` : Quad mode - implantation is performed at all of the `Rotation` + x deg angles, where $x \in 0\,\text{deg}, 90\,\text{deg}, 180\,\text{deg}, 270\,\text{deg}$.
  - `Single` : Single mode - implantation is performed at `Rotation` angle. (Default mode)

- `Type` : Following implantation methods are supported - `Analytic` (default), `MonteCarlo`. Analytic implantation method is described in this chapter, while Monte-Carlo method is described in Chapter 6.

- `Table`: If the implanted ion distribution parameters are not specified, then the distribution moments corresponding to the specified implantation energy, tilt, etc. are taken from the predefined implant tables. The predefined table name can be specified. At the moment, only `Taurus` tables are supported.

Alternately, the implanted ion distribution parameters can be specified by setting the following arguments in the `Implant` section.

- `mu`: Depth of implanted ion distribution.
- `sigma`: Standard deviation of implanted ion distribution.
- `gamma`: Skewness of the implanted ion distribution.
- `beta`: Kurtosis of the implanted ion distribution.
- `mu2`: Depth of implanted ion distribution of the tail of the profile.
- `sigma2`: Standard deviation of implanted ion distribution of the tail of the profile.
- `gamma2`: Skewness of the implanted ion distribution of the tail of the profile.
- `beta2`: Kurtosis of the implanted ion distribution of the tail of the profile.
- `ratio`: Ratio of head to tail distributions in `DualPearson` profile is set.
- `lat.sigma`: Standard deviation of lateral straggle.

- `lat.sigma2`: Standard deviation of lateral straggle of the tail distribution.
- `lat.dSigma`: Depth dependent standard deviation parameter.
- `lat.lv`: Depth dependent standard deviation parameter.

Descriptions of the parameters and their usage is described in the next sections. The above arguments overwrite the parameter values obtained from the implantation tables.

Similarly, an implantation process can be added in a process flow defined in a python file by using `dioPro.implant (...)` procedure on the defined process object `dioPro`. The implant process parameters (energy, dose, etc.) can be set in the implant procedure together with the name of the implant step as follows.

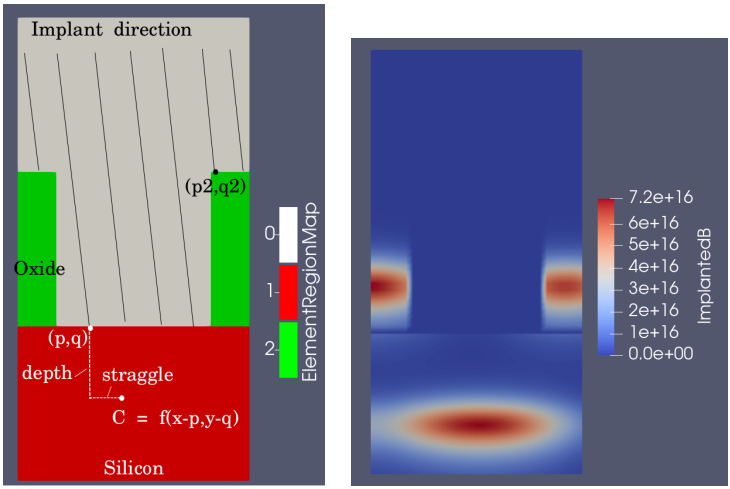
```
# implant Boron
BimplPar = {"mu" : 0.381, "sigma" : 0.0986,
            "gamma" : -0.458, "beta" : 5.2}
dioPro.implant (Name="Impl1", ImplantSpecies="Boron",
                Energy=90, Dose=1E12,
                NumericParMap=BimplPar)
```

Notice, how the parameters of the implanted ion distribution profile are provided by setting the argument `NumericParMap` to the python map variable `BimplPar` defined before.

5.1.2 Analytic implantation method

A point response distribution is the as-implanted ion distribution in the structure generated by a single ion beam incident on a point at the surface of the structure. In analytic implantation, a point-response distribution is empirically generated at each regularly placed point on the surface of the structure using the moments (or equivalent parameters) obtained from the table or specified by the user. At each of the vertex in the structure, contributions of all such point-response distributions are summed up to calculate the total as-implanted ion density at that vertex.

Mathematically, let $F(x, y, p, q)$ represent a point-response distribution at an interior vertex (x, y) due to an ion beam incident at



(a) Analytic distribution (b) Distribution profile

Figure 5.1: (a) Definition of the points (p, q) and (x, y) for defining analytic dopant distribution. (b) Analytic implant profile obtained by implantation of B atoms into the structure in (a).

the points (p, q) on the surface $(p, q) \in \Omega_{\text{gas}}$. Density of as-implanted ions at that interior vertex (x, y) is calculated by superposing the distribution functions generated at all the points on the surface.

$$C(x, y) = N_d \int_{\Omega_{\text{gas}}} F(x, y, p, q) dpdq \tag{5.1}$$

where N_d is the total dose per exposed area and $C(x, y)$ is the doping profile.

The point-response distribution function $F(x, y, p, q)$ is composed of a product of a primary distribution function $f_p(y)$ along the beam direction and a lateral distribution function $f_l(x)$ perpendicular to the beam.

$$F(x, y, p, q) = f_p(y - q) \cdot f_l(x - p) \tag{5.2}$$

5.1.3 Analytic profiles

The primary and the lateral distribution functions are specific for the specific profile shape. Each of the distribution profile is characterized by its moments. The first moment R_p is defined as,

$$m_1 = R_p = \int_{-\infty}^{\infty} y \cdot f(y) \cdot dy \tag{5.3}$$

The higher moments are given by,

$$m_i = \int_{-\infty}^{\infty} (y - R_p)^i \cdot f(y) \cdot dy \tag{5.4}$$

Using these moments the parameters such as standard deviation σ, skewness (γ), and kurtosis (β) can be determined as follows.

$$\sigma = \sqrt{m_2} \tag{5.5a}$$
$$\gamma = \frac{m_3}{\sigma^3} \tag{5.5b}$$
$$\beta = \frac{m_4}{\sigma^4} \tag{5.5c}$$

The following primary distribution profile shapes are supported by the process simulator.

Gaussian profile

Primary function of the Gaussian profile is given below.

$$f_p(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y - R_p)^2}{2\sigma^2}\right) \tag{5.6}$$

Here, R_p and σ is set by the user-provided arguments `mu` and `sigma` respectively. Alternately, they are read from the implant table. For the Gaussian profile, the rest of the parameters are set as follows.

$$\gamma = 0$$
$$\beta = 3$$

Pearson profile

The Pearson distributions are the solutions of the following differential equation.

$$\frac{d}{dy}f(y) = \frac{y-a}{b_0 + b_1 y + b_2 y^2} \cdot f(y) \tag{5.7a}$$

$$A = 10\beta - 12\gamma^2 - 18 \tag{5.7b}$$

$$a = b_1 = -\frac{\gamma\sigma(\beta+3)}{A} \tag{5.7c}$$

$$b_0 = -\frac{\sigma^2(4\beta - 3\gamma^2)}{A} \tag{5.7d}$$

$$b_2 = -\frac{2\beta - 3\gamma^2 - 6}{A} \tag{5.7e}$$

The Pearson distribution of type-IV is given by the following equation.

$$\begin{aligned} f_p(y) =& K|b_2 y^2 + b_1 y + b_0|^{\frac{1}{2b_2}} \times \\ & \exp\left(-\frac{\frac{b_1}{b_2} + 2a}{\sqrt{2b_0 b_2 - b_1^2}} \operatorname{atan}\left(\frac{2b_2 y + b_1}{\sqrt{2b_0 b_2 - b_1^2}}\right)\right) \end{aligned} \tag{5.8}$$

Pearson profile is modeled by using the Pearson-IV distribution. The profile parameters R_p, σ, γ, and β used in Eq. 5.7 and Eq. 5.8 are set by the user-provided arguments `mu`, `sigma`, `gamma`. and `beta`, respectively. Alternately, they are read from the implant table.

Dual-Pearson profile

As-implanted ion distributions often exhibit the main distribution at an implant depth and a tail of the distribution going deep in the substrate. This behavior is reproduced analytically in Dual-Pearson profile by using two Pearson distribution, each for head and tail part.

$$f_p(y) = \alpha \cdot f_{\text{head}}(y) + (1-\alpha) \cdot f_{\text{tail}}(y) \tag{5.9}$$

Parameters for the head of the distribution $f_{\text{head}}(y)$ are specified by the keywords as explained in Sec. 5.1.3. Parameters for Pearson profile of the tail are set by the user-provided arguments `mu2`, `sigma2`, `gamma2`. and `beta2`. Alternately, they are read from the implant table.

5.1.4 Lateral straggle

The lateral distribution function $f_l(x, y)$ determines lateral spread of the point-response function. It is also sometimes called the lateral straggle. It is modeled as a Gaussian distribution centered at the points along the beam as follows.

$$f_l(x, y) = \frac{1}{\sqrt{2\pi}\sigma_l(y)} \exp\left(-\frac{x^2}{2\sigma_l^2(y)}\right) \tag{5.10}$$

Here, $\sigma_l(y)$ is the standard deviation of the lateral straggle.

5.1.5 Depth dependence

Standard deviation of the lateral straggle can be constant or depth-dependent. Depth dependence of lateral straggle is activated by specifying `lat.dSigma` parameter. If activated, the lateral standard deviation as a function of depth (y) is defined as follows.

$$\sigma_1(y) = \sigma_0 \cdot \left(1 + \Delta\sigma\left(\frac{x}{R_p} - 1\right)\right) \tag{5.11}$$

Note: The above expressions are derived for 2D analytic implantation. They are also valid for 3D case.

5.1.6 Implantation in multi-layer structures

Analytic implantation using point-response functions is valid for a single-material layer only and cannot be directly applied to the structures with multiple material layers. For a given ion, each material has a different stopping capability which is inversely proportional to the projected range of that ion in the material. When performing analytic implantation in multi-material structures, point-response functions for a given ion must be corrected to account for blocking capabilities of different materials.

The point-response function in the m^{th} layer is corrected by shifting it along the beam direction by δ_m. This shift δ_m effectively accounts

for different stopping power of all the material layers above the given layer. It is given by,

$$\delta_m = \sum_{k=1}^{k<m} d_k \left(1 - \frac{R_{p,m}}{R_{p,k}}\right) \tag{5.12}$$

where, d_k represents the thickness of the k^{th} layer. Using this correction, the primary distribution function $f'_p(x)$ is calculated as follows.

$$f'_p(y) = \begin{cases} f_{p,1}(y) & \text{for } 0 < y < d_1 \\ \alpha_k f_{p,k}(y - \delta_k) & \text{for } d_{k-1} < y < d_k \end{cases} \tag{5.13}$$

Here, α_k is a rescaling factor of $f_p(y)$ of the k^{th} layer which satisfies the normalization condition. For the point-response function in the first layer, the shift δ_k is equal to zero.

5.1.7 Domain Boundary conditions

Two types of boundary conditions can be set on the domain – `Reflect` and `Extend`. They are set with the argument `LeftBnd` for left-side domain boundary and `RightBnd` for right-side domain boundary.

When `Reflect` Boundary Condition (BC) is set, the implant ion beam incident on the boundary is reflected. When `Extend` BC is set, the implant ion beam is terminated at the boundary. By default `Extend` BC is set on both left-side and right-side domain boundaries.

5.2 Implantation table format

For a given material, parameters of the analytic profiles such as R_p, σ depend on the implant energy, wafer tilt angle. For a given energy and/or tilt, the parameter values can be obtained experimentally or can be determined from more sophisticated Monte-Carlo implantation technique. Implantation tables store the parameter values of the analytic profiles for a range of energies and/or tilt. The parameters can then be calculated for a specific set of energy and/or tilt by interpolating the parameter values from the implant tables.

5.2.1 Taurus table format

Currently, implant tables in `Taurus` format are supported. It consists of a file header and a table of numeric data. The file header consists of two lines. The first line lists three names as follows.

`<type> <species> <mater>`

`<type>` specifies whether the table lists `Implant` parameters or `Damage` parameters, `<species>` specifies the species name, and `<mater>` specifies the material name for which the parameters are listed. The second line of the file header specifies the list of names of the implantation conditions, in lowercase. The following names are currently recognized.

`energy tilt rotation dose`

The names can be listed in any arbitrary sequence and can also be omitted. These names are in the same order as the first columns of the table in which implantation conditions are listed. The implantation conditions must be listed in the following units only.

- `energy` in `keV`.
- `tilt` in `degrees`.
- `rotation` in `degrees`.
- `dose` in $\texttt{cm}^{-2}$.

The numeric data consists of a number of lines. Each line specifies the implantation condition followed by all the implantation parameters for that specific implantation condition. The numeric values must be separated by whitespace characters.

In each table, the number of entries in each line must be fixed for the entire table. If c is the number of implantation conditions, then there should be at least $c + 4$ values per line for a Gaussian profile, $c + 6$ values per line for a Pearson profile, and $c + 13$ values per line for a dual-Pearson profile. The implantation conditions must be listed in the same sequence as specified on the second header line. The implantation moments must be listed for each distribution type as follows.

- Gaussian: `mu sigma lat.sigma lat.dSigma`
- Pearson: `mu sigma lat.sigma lat.dSigma gamma beta`

- Dual Pearson: `mu sigma lat.sigma lat.dSigma gamma beta` followed by
  `mu2 sigma2 lat.sigma2 lat.dSigma2 gamma2 beta2 ratio`.

Any line beginning with * is considered a comment and is not parsed. An example of Taurus implant table for `Phosphorus` implantation in `Silicon` with the implant condition `energy` is shown below.

```
Implant Phosphorus Silicon
energy
10 0.0139 0.0069 0.0080 0 0.641 3.68617127
20 0.0252 0.0119 0.0139 0 0.549 3.50333967
30 0.0368 0.0166 0.0195 0 0.459 3.35183727
```

5.3 Normalization of dopant profiles

Chapter 6

Monte-Carlo Ion Implantation Method

Simulation of ion implantation process by using analytic profiles is described in Chapter 5. In this chapter, ion implantation by Monte-Carlo (MC) method is described together with the underlying physics-based models.

6.1 Defining Monte-Carlo implantation

A Monte-Carlo implantation step is added to the process flow in a config file by setting an implant object as follows.

```
Implant*impl1: {
  Species = "Boron";
  Type = "MonteCarlo";
  Energy = 90;
  Dose = 1E12;
  Tilt = 0;
  Rotation = 0;
  Mode = "Quad"; // or "Dual" or "Single"
}
```

The above text declares an `Implant` step named `impl1`. Various parameters declared in the above section have the same meaning as described in Section 5.1.1. MC method is selected for simulating implantation by specifying `MonteCarlo` as `Type`. Various hyper-parameters in MC method can also be set in the `Implant` section. The following MC hyper-parameters can be specified.

- `NumberOfParticles` : Maximum number of ions implanted into the structure.
- `MaxReplicatedTrajectories` : Maximum number of replications of each trajectory. Default value is 0, which means trajectory replication is turned off.
- `SmoothingDist` : Dopant profile obtained MC implantation is smoothened to reduce noise. Smoothing distance is specified in "μm". It defaults to 0.002μm.

6.1.1 Python interface

Monte-Carlo implantation step can be initialized in python script by using the following python script.

```
MCImplNumPar = {"NumberOfParticles" : 10, "MaxReplicatedTrajectories"
                "SmoothingDist" : 0.002}
dioPro.implant (Name="Impl1", Type="MonteCarlo", ImplantSpecies="Boro
                Tilt=0, Dose=1E12, NumericParMap=MCImplNumPar)
```

Here, `dioPro` is a 'Process' object instantiated before. Various keywords and their arguments have the same meaning as described in Section 5.1.1. Various MC hyper-parameters are specified as a python 'map' of parameter name to its value. The map is named `MCImplNumPar` in the above code. This map is provided with `NumericParMap` as an argument.

6.2 Monte-Carlo Implantation Method

In MC implantation method, a fixed number of particles are implanted into the patterned substrate. Their trajectories inside the substrate

are obtained by simulating elastic/inelastic collisions. After every collision, kinetic energy of these particles decreases until they come to rest (energy < 0). Rest positions of all the particles are scaled to obtain the dopant concentration. The dopant concentration is diffused by the user-specified smoothing distance.

For each particle, the algorithm calculates the particle trajectory as follows. Particle incidence direction is determined from the implantation conditions. Incidence location of the particle on the surface of the substrate is obtained from uniform distribution on the plane. Incidence particle energy is set to implant energy. At each step, nuclear and electronic energy loss, scattering angle, and mean-free-path are calculated. At the end of the step, the particle energy is updated by subtracting the energy losses. It is scattered by scattering angle and its location is advanced by mean-free-path. This procedure is repeated till the particle comes to rest (particle energy ¡ 0). This algorithm is described in Fig. 1.

Data : For each particle,
incidence location ($\vec{r_0}$), direction ($\vec{d_0}$), and energy (E);
while $E_i > 0$ **do**
  At step i, init
  mean free path L_i,
  scattering angle θ_i,
  nuclear energy loss ΔE_i^n,
  electronic energy loss ΔE_i^e. **if** $\vec{r_i}$ *is in semiconductor* **then**
    Calculate for crystalline semiconductor -
    L_i, Θ_i, ΔE_i^n, ΔE_i^e
  else
    Calculate for amorphous semiconductor -
    L_i, Θ_i, ΔE_i^n, ΔE_i^e
  end
  Update
  particle position, $\vec{r}_{i+1} = \vec{r}_i + \Delta r_i(\theta_i)$
  particle energy, $E_{i+1} = E_i - \Delta E_i^n - \Delta E_i^e$
end

Algorithmus 1 : Monte-Carlo implantation algorithm

6.2.1 Binary collision approximation

When an implanted particle of mass M_i enters an amorphous or a crystalline material, it undergos a series of inelastic collisions with the atomic nuclei of mass M_s in the material. Each of these collision events deflect the particle by the scattering angle θ_i. It also decelerates these particles. Energy loss (ΔE_i^n) during i_{th} inelastic nuclear scattering event is given by -

$$\Delta E_i^n = E_i \cdot \frac{4 \cdot M_i \cdot M_s}{(M_i + M_s)^2} \cdot \cos^2(p\Theta) \tag{6.1}$$

where, p is the impact parameter, E_i is implant atom energy before scattering.

Nuclear scattering changes direction of the implanted ion. This change of direction θ_i is given by,

$$\theta_i = \cos^{-1}\left(\frac{1 - 0.5 \cdot (1 + \frac{M_s}{M_i}) \cdot \frac{\Delta E_i^n}{E_i}}{\sqrt{1 - \frac{\Delta E_i^n}{E_i}}}\right) \tag{6.2}$$

In amorphous materials, scattered atom has an equal probability of scattering in any direction along the conic surface formed by all the lines inclined by angle θ_i to the incident direction of the implant atom. To model it, azimuthal angle of scattered direction is set by drawing a number from uniformly distributed angles between $[0, 2\pi)$. The direction of the scattered implant atom is determined from the scattering angle θ_i and the randomly drawn azimuthal angle.

In crystalline material, due to the patterned nature of nuclear locations, the scattering events due to subsequent collisions with the patterned collision centers can cancel out the deflections of the implanted atoms. This patterned nature of collisions is considered in determining the scattering angle as explained in Subsection 6.2.1.

Parameter Θ in Eq. 6.1 is given by the following integral.

$$\Theta = \int_{r_{\min}}^{\infty} \frac{dr}{r^2 \sqrt{(1 - V(r)/E_r - p^2)}} \tag{6.3}$$

Here, r is the separation between incident implant atom and the substrate atom, $r_{\min}$ is the minimum separation between them, $V(r)$

is the interatomic potential between them, while $E_r = \frac{M_s \cdot E_i}{M_i + M_s}$ is a constant energy.

Substituting $s = \frac{1}{r}$ and changing integration limits,

$$\Theta = \int_0^{s_{\max}} \frac{ds}{\sqrt{(1 - V(s)/E_r - p^2 s^2)}} \tag{6.4}$$

Here, $s_{\max}$ is the inverse distance of nearest approach of the two interacting particles, given by the root of the equation,

$$1 - V(s)/E_r - p^2 s^2 = 0. \tag{6.5}$$

Coulomb potential

The interatomic potential $V(r)$ can be set to bare Coulomb potential between the two interacting nuclei. It is given by,

$$V(r) = \frac{Z_i Z_s}{4\pi\epsilon_0 r} \tag{6.6}$$

where, Z_i and Z_s are the nuclear charges of the implant particle and substrate atom. By transforming to variable s ($s = 1/r$), the Coulomb potential can be written as,

$$V(s) = \frac{Z_i Z_s}{4\pi\epsilon_0} \cdot s \tag{6.7}$$

This form of potential is applicable when electrons in the substrate *do not* screen the nuclear charges of the implant particle and the substrate atoms. Substituting Eq. 6.7 in Eq. 6.5 and solving it gives,

$$s_{\max} = \frac{\sqrt{1 + 4 \cdot e_r^2 \cdot p_n^2} - 1}{2 \cdot e_r \cdot p_n^2} \tag{6.8}$$

where e_r is given by Eq. 6.15. Using $s_{\max}$ in Eq. 6.4 and performing intergration gives $\cos^2(p\Theta)$ as follows,

$$\cos^2(p\Theta) = \frac{1}{1 + 4 \cdot e_r^2 \cdot p_n^2} \tag{6.9}$$

Thus, setting interatomic potential $V(r)$ to bare Coulomb potential gives an analytic expression for $\cos^2(p\Theta)$, which is used to evaluate energy loss Eq. 6.2.

Universal potential

In reality, the interatomic potential between the two nuclei is screened by electrons. Ziegler, Biersack, and Littmark performed the calculation of the interatomic potentials for numeruos atom-pairs. Based on these calculations they could derive the universal screening potential which is suitable for arbitrary atom-pairs. This potential is given by,

$$V(r) = \frac{Z_i Z_s}{4\pi\epsilon_0 r}\Phi(r) \tag{6.10}$$

where

$$\begin{aligned}\Phi(r) =& 0.1818 \cdot \exp\left(-3.2\frac{r}{a_u}\right) + 0.5099 \cdot \exp\left(-0.9423\frac{r}{a_u}\right) + \\ & 0.2802 \cdot \exp\left(-0.4029\frac{r}{a_u}\right) + 0.02817 \cdot \exp\left(-0.2016\frac{r}{a_u}\right) \end{aligned} \tag{6.11}$$

$$\tag{6.12}$$

$$a_u = 0.8854 \cdot \frac{a_0}{Z_i^{0.23} + Z_s^{0.23}}\mathring{A} \tag{6.13}$$

Integration in Eq. 6.4 is performed for each collision event between an implanted atom and a substrate atom. Due to lack of analytic form, this integration has to be performed numerically. To avoid large computational overhead, Eq. 6.4 is transformed to a dimensionless integral which depends on two parameters p_n ($p_n = p/a_u$) and s' ($s' = a_u \cdot s = a_u/r$) as follows.

$$\begin{aligned}\Theta =& \frac{1}{a_u}\int_0^{s'_{\max}} \frac{ds'}{\sqrt{(1 - \frac{Z_i Z_s}{4\pi\epsilon_0 a_u E_r} \cdot s'\Phi(s') - p_n^2 s'^2)}} \\ \Theta =& \frac{1}{a_u}\int_0^{s'_{\max}} \frac{ds'}{\sqrt{(1 - s'\Phi(s')/e_r - p_n^2 s'^2)}} \end{aligned} \tag{6.14}$$

Here,

$$e_r = \frac{4\pi\epsilon_0 a_u E_r}{Z_i Z_s} \tag{6.15}$$

The reduced Eq. 6.14 depends only on the two parameters namely, reduced impact parameter p_n and reduced energy e_r. Integral of Eq. 6.14 is stored as a lookup table in the MC simulator.

In this way, once the impact parameter of the collision is determined, p_n and e_r are calculated. These two values are used to obtain Θ using Eq. 6.14. Eq. 6.1 and Eq. 6.2 are evaluated using Θ to get nuclear energy loss ΔE_i and scattering angle θ_i of the i^{th} step.

Calculation of the impact parameter p for amorphous and crystalline materials is explained below.

Amorphous materials

In amorphous material of atom density N_{dens}, the implant atom travels by distance L between successive collisions with the substrate atoms.

$$L = \frac{1}{N_{dens}^{1/3}} \tag{6.16}$$

This distance is assumed fixed for each collision step. For amorphous materials, in which atoms are assumed to be uniformly distributed, the impact parameter for each collision step is given by.

$$p = \sqrt{r_{\text{rand}}}\pi N_{dens}^{2/3} \tag{6.17}$$

Here, r_{rand} is a random number drawn from a uniform random distribution between 0 and 1. For i^{th} nuclear scattering event, the impact parameter p is obtained from the random number. Once p is set, Eq. 6.14 is evaluated from the look-up table to obtain $\cos^2(p\Theta)$. This is used to determine energy loss ΔE_i^n and scattering angle θ_i during the nuclear scattering event. This information is used to obtain implant ion energy and direction after the scattering event. After the scattering event, the implanted ion travels by mean-free path L before the next nuclear scattering event.

Crystalline materials

In crystalline material, due to the patterned nature of nuclear locations, the scattering events due to subsequent collisions with the patterned collision centers can cancel out the deflections of the implanted atoms. This patterned nature of collision centers must be considered in calculating impact parameter p as well as direction of the scattered implant ion. Suppose, the implant ion is at the position $\vec{r}_i$ before i^{th}

scattering event traveling in the direction $\hat{d}_i$. The implant ion is paired with all the substrate atoms which lie in the imaginary cylinder of height D and radius R, whose axis begins at $\vec{r}_i$ and ends at $\vec{r}_i + D \cdot \hat{d}_i$. Net energy loss is calculated by adding energy losses arising from scattering due to each of the selected substrate atoms. Direction of the scattered ion is calculated by vectorially adding deflections from scattering due to each of the selected substrate atoms. After scattering, the atom travels a distance of $L = D$. Note, that unlike in amorphous material, the above procedure does not set azimuthal angle from a random distribution. The parameter $D = k \cdot a_0$ (where a_0 is the lattice constant) is an adjustable parameter. The scaling factor k is set to 1.2.

6.2.2 Electron stopping model

In addition to collisions with the nuclei, the implant ion is decelerated by electrons in the material. This is modeled by electronic stopping model which has both local and nonlocal components. In this model, energy loss (ΔE_i^e) of implanted ions due to local ($\Delta E^{\mathrm{e,l}}$) and nonlocal ($\Delta E^{\mathrm{e,nl}}$) components is given by,

$$\Delta E_i^e = x_{nl} \cdot \Delta E^{\mathrm{e,nl}} + (1 - x_{nl}) \cdot \Delta E^{\mathrm{e,l}} \tag{6.18}$$

where x_{nl} determines relative contribution of nonlocal component. It is given by,

$$x_{nl} = \min\left(A_{nl} e_r^{nle}, 1\right) \tag{6.19}$$

Here, e_n is the scaled reduced energy given by Eq. 6.15. A_{nl} and nle are adjustable parameters, which can be set using the following script in the config file.

```
SetParam*s0: {
  Command = "<mater> <dopant> <Par> <val>";
}
```

Here, `<mater>` is the material name, `<dopant>` could be any of the dopant species, `<Par>` is set to `NlocPre` or `NlocExp` to set A_{nl} or nle, respectively. `<val>` is the value of the respective parameter. In a python file, the following command can be used.

```
<obj>.setParamCmd (Name=<name>,
  Command="<mater> <dopant> <Par> <val>")
```

The nonlocal component is proportional to the distance (L) traveled by implant atom between two successive nuclear scattering events in the material density N_{dens}.

$$\Delta E^{\mathrm{e,nl}} = L \cdot N_{dens} \cdot S_e \tag{6.20}$$

Here, S_e is the electronic stopping power given by,

$$S_e = L_{\mathrm{pre}} \cdot \alpha_{\mathrm{es}} \cdot \frac{\sqrt{E_m}}{f_{es}} \tag{6.21}$$

Here, L_{pre} is an adjusting parameter which can be set as explained above when `<Par>` is set to `Lpre`. E_m is the ion energy at maximum stopping power, given by,

$$E_m = \frac{Z_s \cdot I_0}{\frac{4m_e}{M_i m_0}} \tag{6.22}$$

$$I_0 = \begin{cases} 12 + \frac{7}{Z_s} & \text{for } Z_s < 13 \\ 9.76 + \frac{58.5}{(Z_s)^{1.19}} & \text{otherwise} \end{cases} \tag{6.23}$$

Similarly, α_{es} in Eq. 6.21 is given by,

$$\alpha_{\mathrm{es}} = \frac{1.212 \cdot Z_i^{7/6} \cdot Z_s}{(Z_i^{2/3} + Z_s^{2/3})^{3/2}\sqrt{M_i}} \tag{6.24}$$

while, f_{es} in Eq. 6.21 is given by,

$$f_{es} = \left(\frac{\frac{E_i}{E_m}}{\ln(\frac{E_i}{E_m} + \frac{E_m}{E_0} + e - 2))^{\delta/2}} + (\frac{E_m}{E_i W})^{\delta/2} \right)^{1/\delta} \tag{6.25}$$

In the above equation, E_i is the implant ion energy, E_m is the energy at maximum stopping power (Eq. 6.22), δ is adjustable parameter. δ can be set as explained above when `<Par>` is set to `StopPExp`.

Local component of electron stopping model in Eq. 6.18 is given by,

$$\Delta E^{\mathrm{e,l}} = \frac{S_e}{2\pi a^2} \cdot \exp\left(-\frac{p}{a}\right) \tag{6.26}$$

$$a = S_{par} \frac{1.45}{Z_i^{2/5}} \frac{a_u}{0.3} \tag{6.27}$$

Here, S_{par} is adjustable screening length parameter, which can be set as explained above when `<Par>` is set to `Scrpre`.

Chapter 7

Annealing

Various semiconductor fabrication processes utilize high temperature treatment on the wafer for oxidation, dopant activation, defect-removal, or for diffusing dopants to the active area. At high temperature, the dopant diffusivity in Silicon is high enough to cause notable dopant diffusion. This changes the net dopant distribution and relocates p-n junctions thereby influencing the device characteristics. Therefore, it is necessary to model the dopant diffusion during the high temperature treatments.

Dopant diffusion due to the high temperature processes can be modeled in the process simulator by introducing annealing step in the process flow. This chapter describes available settings in the annealing step and their usage.

7.1 Defining annealing step

Annealing step is added to the process flow in a config file using the following syntax.

```
Anneal*ann1: {
  Temperature = [800., 1100., 1100., 800.];
  TimeIntervals = [600., 60., 600.];
  N2 = [20., 20., 20.];
}
```

The following arguments can be specified in the `Anneal` object.

- `TimeIntervals`: A list of time-intervals (in seconds) in a temperature ramp cycle.
- `Time`: A list of time values (in seconds) at the end of each time interval.
- `Temperature`: A list of temperature values starting with the initial chamber temperature followed by temperature values at the end of each time interval. This list is longer by one element than `TimeIntervals` list.
- Gas flows: Gas names can be listed followed by a list of gas flow values at each time interval. Gas flow list has the same number of elements as `TimeIntervals` list.

Currently following gas names are recognized by the process simulator.

- `H2` Hydrogen gas (H_2).
- `O2` Oxygen gas (O_2).
- `N2` Nitrogen gas (N_2).
- `H2O` Water vapour (H_2O).
- `N2O` Nitrous oxide (N_2O).
- `HCl` Hydrogen chloride (HCl).
- `Cl2` Chlorine gas (Cl_2).

These gases can be listed in the `Anneal` object followed by the list of gas flows. Gas flows are mainly used to determine partial pressure of the oxidizing gases, which determine oxidation rate at Silicon surface. Oxidation process is described in more detail in the next chapter.

In a python file, annealing step is added to the process flow using the `dioPro.anneal (...)` procedure on 'dioPro' object.

```
# perform annealing
TempProfile = [800., 1100., 1100., 800.]
TimeIntervals = [600., 60., 600.]
Gases = {"N2" : [20., 20., 20.]}
dioPro.anneal (Name="ann1", Temperature=TempProfile,
               Time=TimeIntervals, GasFlows=Gases,
               NumericParMap=TriangleMeshParRefined,
               TimesToSave=[100,400,800,1200],
               QuantitiesToSave=["BActive",
               "PhActive","IntNeutral","VacNeutral"])
```

Names of the arguments are the same as for the `Anneal` object in the config file. Gas flows are specified as a python map of gas name to the gas flow list as shown by `Gases` variable in the above code snippet. Various numeric parameters can be provided as python map of parameter name to its value. In this case, mesh ('TriMesh' or 'TetMesh') parameters, which are used in meshing the new oxide region created by oxidation, are supplied as numeric parameters with `NumericParMap`. Similarly, `TimesToSave` specifies times at which the distribution of various diffusing species is to be stored. `QuantitiesToSave` specifies the diffusing species whose distribution is stored. They are stored in hdf (*.h5) format with the following name - <DevName>_<Name>_dat_<id>.h5. A *.xdmf script is also stored which enables viewing the dopants with paraview.

7.2 Diffusion models

7.2.1 Defect dynamics

Dopant diffusion in Silicon is influenced not only by the gradient of the dopant distribution, but also by the concentration of defects such as interstitials and vacancies. Therefore it is necessary to model defect dynamics and defect diffusion.

In Silicon, defects can be neutral or can carry a charge. At any given temperature, the neutral and the charged defects are always assumed to be in a specific proportion. $C^*_{X^0}$ is the equilibrium concentration of

the neutral defects. It is related to the total equilibrium concentration of defect X as follows.

$$C^*_{X^0} = \frac{C^*_{X(intr)}}{\sum_c k_{X^c}} \tag{7.1}$$

Here, $C^*_{I(intr)}$ and $C^*_{V(intr)}$ are equilibrium concentrations of interstitials and vacancies, respectively. They can be set using the following command.

```
SetParam*s0: {
  Command = "<mater> <defect> Cstar <val>";
}
```

Here, `<mater>` is the material name, `<defect>` could `IntNeutral` or `VacNeutral`, `<val>` can be a constant value or the Arrhenius function (e.g. `Arrhenius 1.43E23 2.0`). When `<val>` is the Arrhenius function, its value is calculated at every temperature using the provided prefactor (in appropriate units) and activation energy (always in eV). In a python file, the following command can be used.

```
<obj>.setParamCmd (Name=<name>,
  Command="<mater> <defect> Cstar <val>")
```

Here, `<obj>` is the Process object variable, `<name>` is the step name.

The equilibrium charging constant, k_{X^c} are defined as follows.

$$C_{X^c} = k_{X^c} C_{X^0} (\frac{n}{n_i})^{-c} \tag{7.2}$$

Here, n is the net electron concentration, n_i is the intrinsic carrier concentration. C_{X^c} and C_{X^0} are the concentration of neutral and charged defect X ($\in \{I, V\}$)) in a doped semiconductor. k_{X^c} is the charging constant of the defect X.

$$k_{X^c} = k^0_{X^c} \exp\left(-\frac{k^E_{X^c}}{kT/q}\right) \tag{7.3}$$

k_{X^c} can be set using the following command.

```
SetParam*s0: {
  Command = "<mater> <defect> ChargeStates <c> <val>";
}
```

Here, `<c>` is the charge state.

The defect flux $\vec{J}_X$ is given by

$$\vec{J}_X = -\frac{\sum_c k_{X^c} D_{X^c} (\frac{n}{n_i})^{-c} C_X^*}{\sum_c k_{X^c} (\frac{n}{n_i})^{-c}} \nabla \frac{C_X}{C_X^*} \tag{7.4}$$

Here, C_X denotes total concentration of defects at all the charge states. The diffusivity of the defect X with charge state c is noted by D_{X^c}.

$$D_{X^c} = D^0_{X^c} \exp\left(-\frac{D^E_{X^c}}{kT/q}\right) \tag{7.5}$$

It can be set using,

```
SetParam*s0: {
  Command = "<mater> <defect> D <c> <val>";
}
```

Recombination of defects

Interstitial Silicon atoms can enter in a vacant position in Silicon crystal lattice. This process is called recombination of an interstitial and a vacancy. Recombination of interstitial defects and vacancy defects R_{IV} is given by,

$$R_{IV} = K_{IV}(C_I C_V - C_I^* C_V^*) \tag{7.6}$$

The recombination constant is given by,

$$K_{IV} = \frac{\sum_i \sum_j K_{I^i V^j} k_{I^i} k_{V^j} (\frac{n}{n_i})^{-(i+j)}}{\sum_c k_{I^c} (\frac{n}{n_i})^{-c} \sum_c k_{V^c} (\frac{n}{n_i})^{-c}} \tag{7.7}$$

Where $K_{I^i V^j}$ is given by `KbulkChargeStates`.

7.2.2 Charged-pair model

Dopant atoms (A) diffuse in Silicon by forming a charged dopant-defect complex (AX^c) with the defects. The dopant-defect complex undergoes diffusion and thereby transports the dopant atoms. In charged-pair model, this process of formation of AX^c from the dopants and the

defects is assumed to be fast-enough that the dopants and the defects are always in equilibrium with the dopant-defect complex. That is,

$$A^z + I^c \rightleftarrows AI^{(z+c)}$$
$$A^z + V^c \rightleftarrows AV^{(z+c)}$$

Here, A is any of the dopant atom with the charge z, and I,V are charged defects with charge c. The defects themselves are not assumed to be in equilibrium.

In this model, the following differential equations are solved in time evolution.

$$\frac{\partial C_A}{\partial t} = -\nabla \cdot \vec{J}_A \tag{7.8a}$$
$$\frac{\partial C_{X^{all}}}{\partial t} = -\nabla \cdot \vec{J}_A - \nabla \cdot \vec{J}_X - R_{IV} \tag{7.8b}$$

where, $C_{X^{all}} = C_X + C_{AX}$ is the total defect concentration including dopant-defect pairs, $\vec{J}_A$ is the sum of AI and AV pair fluxes, $\vec{J}_X$ is the defect flux and C_A is the total dopant concentration.

The dopant flux is given by

$$J_A = -\left(\sum_{c,X} D_{AX^c}\left(\frac{n}{n_i}\right)^{-c}\right) \times \left(\frac{C_{X^0}}{C^*_{X^0}}\nabla C^+_A + C^+_A \nabla(\frac{C_{X^0}}{C^*_{X^0}}) + C^+_A(\frac{C_{X^0}}{C^*_{X^0}})\nabla \ln\left(\frac{n}{n_i}\right)\right) \tag{7.9}$$

Here, c is the charge on the dopant-defect complex (AX^c). C^+_A is the active dopant concentration. The rest of the symbols have the same meaning as before. Notice, that the dopant flux due to the defect is proportional to the gradient of not only the dopant concentration, but that of the the defect concentration ∇C_{X^0} and the electron Fermi level $\nabla \ln\left(\frac{n}{n_i}\right)$. The above equation accounts for dopant movement due to the forces of defect diffusion and electric field.

Effective diffusivity D_{AX^c} of the dopant-defect pair with charge c is given by,

$$D_{AX^c} = D^0_{AX^c} \exp\left(-\frac{D^E_{AX^c}}{kT/q}\right) \tag{7.10}$$

It can be modified by the following command.

```
SetParam*s0: {
  Command = "<mater> <dopant> <defect> D <c> <val>";
}
```

The above command updates effective diffusivity of the `<dopant>`-`<defect>` pair with the charge `<c>` to the value (or Arrhenius function) set by `<val>`.

7.2.3 Charged-Fermi model

Charged Fermi model is similar to the charged pair model described in the previous section, except that the defect concentration is always assumed to be in equilibrium. The process of interstitial-vacancy recombination is assumed to be fast enough that the excess defect flux is instantaneously absorbed into the crystal lattice to reach equilibrium defect concentration.

$$I^i + V^j \rightleftarrows -q(i+j)$$

Thus, the diffusion equation for dopant-defect pairs (AX) is solved for the given dopant A when this model is switched on. The following equation is solved in time-stepping iterations.

$$\frac{\partial C_A}{\partial t} = -\nabla \cdot \vec{J}_A \tag{7.11}$$

Here the dopant flux $\vec{J}_A$ is given by the following expression.

$$\vec{J}_A = -\left(\sum_{c,X} D_{AX^c} (\frac{n}{n_i})^{-c}\right)\left(\nabla C_A^+ + C_A^+ \nabla \ln\left(\frac{n}{n_i}\right)\right) \tag{7.12}$$

Here, c is the charge on the AX pair, C_A^+ is the active dopant concentration. Notice, that the dopant flux depends on the gradient of diffusivity term of electrons and holes (given by $\ln\left(\frac{n}{n_i}\right)$) together with that of the dopant concentration. Dependence on the gradient of defect concentration is ignored.

Here, D_{AX^c} is the diffusivity of the dopant-defect pair of charge c. It is given by,

$$D_{AX^c} = D^0_{AX^c} \exp\left(-\frac{D^E_{AX^c}}{kT/q}\right) \tag{7.13}$$

The term D_{AX^c} can be modified using the following command.

```
SetParam*s0: {
  Command = "<mater> <dopant> <defect> Dstar <c> <val>";
}
```

7.2.4 Constant model

This model is the simplest model of dopant diffusivity. It ignores any interaction between the dopants and the defects. Also, it ignores the effect of electron and hole diffusion on the dopant diffusivity. This model can be activated in any material, although it is particularly useful for modeling dopant diffusion in the oxide.

In this model, the following equation is solved in time-stepping iterations.

$$\frac{\partial C_A}{\partial t} = -\nabla \cdot \vec{J}_A \tag{7.14}$$

The dopant flux is given by,

$$\vec{J}_A = -D^* \partial C_A^+ \tag{7.15}$$

where C_A^+ is active portion of C_A.

Constant diffusivity D^* is given by,

$$D^* = D_0^* \exp\left(-\frac{D^* E}{kT/q}\right) \tag{7.16}$$

It can be modified using the following command.

```
SetParam*s0: {
  Command = "<mater> <dopant> Dstar <val>";
}
```

Note, that the parameter is named `Dstar` same as in `ChargedFermi` model. But it is a `<dopant>` property, *not* a `<dopant>-<defect>` property.

7.2.5 Selecting diffusion model

Any of the following diffusion models can be selected depending on the process and the substrate conditions.

- `ChargedPair`: It computes time evolution of the dopant concentration as well as Interstitial and vacancy concentration by calculating the flux of the respective quantities. Hence, it is also called three-stream model. It assumes that the dopant-defect pair formation and dissociation process is fast enough that the pairs are always in equilibrium with the dopants and the defects. It does not assume equilibrium concentration for defects. This model is required when additional interstitials or vacancies are pumped into Silicon structure from the substrate or from the oxidizing surface during oxidation process.

- `ChargedFermi`: It computes time evolution of the dopant concentration, and assumes equilibrium the defect concentration at all times and the dopant-defect pairs are always in equilibrium with the dopants and the defects. This model accounts for electric field effects on the dopant diffusion. This model is useful for long annealing processes when the interstitials and the vacancies are in near equilibrium concentration at each temperature.

- `Constant`: This model assumes fixed value of diffusivity at each given temperature and no electric field effect. It is useful for modeling dopant diffusion in oxide.

Appropriate diffusion model can be activated for the dopant in a specific material by the following command.

```
SetParam*s0: {
  Command = "<mater> <dopant> DiffusionModel <val>";
}
```

Here, `<Dopant>` can be any recognized dopant atom. `<val>` can be set to any one of the three strings `ChargedPair`, `ChargedFermi`, or `Constant`.

In each material, time evolution of the defect concentration during diffusion process is solved, if `ChargedPair` model is active for any one of the active dopants. If `ChargedPair` model is not active for any of the dopants, then the defect concentration is set to its equilibrium value throughout the device. Also, when `ChargedPair` model is active for one of the dopants, while `ChargedFermi` or `Constant` model is

active for another one, then the latter dopant diffusion process will continue assuming equilibrium defect concentration.

It is advisable to keep the choice of diffusion models and the diffusion parameters in a specific material fixed throughout the process simulation for comparative analysis of the dopant distribution at the end of process.

7.3 Boundary conditions

BCs are active at the interface of each pair of materials in the structure. A distinction is made between domain boundaries, at which `homogeneous Neumann` BC is always active. Also, at the boundary between two regions of the same material, `Continuous` BC is always active.

For the dopants or the defects, various BC models can be set at the hetero-interface using the following command.

```
SetParam*s0: {
  Command = "<mater1>_<mater2> [<defect>|<dopant>] SurfaceRecVelModel
}
```

Here, `<val>` can be set to any one of the BCs.

- `HomNeumann`
- `Continuous`
- `Natural`: Valid only for defects
- `Segregation`: Valid only for dopants
- `Dirichlet`

7.3.1 Homogenous Neumann

In this model, no fluxes or transfers across the interface are assumed. It is active by default for the domain boundary on the left, right, bottom, and top. It is also active at the interface between Gas and any other material.

7.3.2 Continuous

In this BC, both concentration and fluxes of dopants or defects normal to the interface are set to be equal.

7.3.3 Natural

This is the default boundary condition at oxide-Silicon interface. It is activated by specifying the following command.

```
SetParam*s0: {
  Command = "<mater1>_<mater2> <defect> SurfaceRecVelModel Natural";
}
```

In this model, flux of defects normal to the interface is given by the following expression.

$$\vec{J}_X \cdot \hat{n} = k_s \left(1 + k_{Rat}(\frac{|V_{ox}|}{V_{Scale}})^{k_{pow}} P_0^{k_{ppow}}\right)(C_{X^0} - C^*_{X^0}) - G_{ox} \quad (7.17)$$

Here, $\vec{J}_X$ is the defect flux, $\hat{n}$ is the unit normal to the interface, k_s is the surface recombination rate, G_{ox} is the generation rate of the defect X due to the oxidation process, P_0 is the oxygen partial pressure, $|V_{ox}|$ is the local oxidation rate, V_{Scale} is the reference oxidation rate, k_{Rat}, k_{pow}, and k_{ppow} are model parameters. These parameters can be set by using the following command

```
SetParam*s0: {
  Command = "<mater1>_<mater2> <defect> <param> <val>";
}
```

Here, `<param>` is set to `Ksurf`, `Scale`, `Krat`, `Kpow`, and `Kppow` to modify k_s, V_{Scale}, k_{Rat}, k_{pow}, and k_{ppow}, respectively. `<val>` is the value of these models which could be a number or an Arrhenius function.

Interstitial generation

Generation rate of the interstitials during oxidation process G_{ox} is given by the following equation.

$$G_{ox} = |V_{ox}| L_{dens} \left(\frac{|V_{ox}|}{V_{Scale}}\right)^{G_{pow}} P_0^{G_{gpow}} G_{Scale} \quad (7.18)$$

Here, G_{pow} and G_{gpow} are the fitting parameters and can be modified by using the `<param>` name `Gpow` and `Ggpow`, respectively. L_{dens} is the lattice density of Silicon and can be set using `LatticeDensity`. This generation rate is used in Eq. 7.17 to calculate calculation of defect flux at the interface.

7.3.4 Segregation

This is the default BC at oxide-Silicon interface. Dopant flux normal to the interface is given by,

$$\vec{J}_X \cdot \hat{n} = k_{\text{transfer}} \left(C_A^a - \frac{C_A^b}{k_{\text{segregation}}} \right) \tag{7.19}$$

where, C_A^a is the concentration of the dopant A on one side of the interface and C_A^b is that on the other side of the interface. k_{transfer} is the transfer rate while $k_{\text{segregation}}$ is the segregation of the dopant at the material interface `a/b`. Additionally, Total dopant fluxes at the interface are conserved. The parameters k_{transfer} and $k_{\text{segregation}}$ are named `Transfer` and `Segregation` in the parameter set. They can be modified by using these names as `<param>` in the following command.

```
SetParam*s0: {
  Command = "<mater1>_<mater2> <dopant> <param> <val>";
}
```

7.3.5 Dirichlet

Dirichlet boundary condition allows to specify a fixed concentration of the dopant or the defect species at the interface. This technique can be used to mimic in-diffusion of dopants from the dopant source. It can also model interstitial injection from Silicon substrate. This BC can only be activated at the interface between any material and the gas. If active, Dirichlet BC sets the dopant or the defect concentration to its equilibrium value set by the parameter `Cstar` as follows.

```
SetParam*s0: {
  Command = "<mater1>_Gas [<dopant>|<defect>] Cstar <val>";
}
```

Chapter 8

Oxidation

Thermal oxidation of Silicon is often employed in the process flow to generate high quality oxide-Silicon interface. When Silicon wafer is annealed in the oxidative atmosphere, Silicon oxide (SiO_2) is formed on the exposed Silicon surface. Rate of oxidation and shape of the oxide depend on the anneal temperature as well as partial pressure of the reactive gases. Also, oxidation process generates interstitials at the interface which are injected into the wafer. This affects the dopant diffusion unevenly. For this purpose, it is useful to model the oxidation process.

8.1 Defining oxidative anneal

The process simulator can simulate the above described effects as soon as an oxidative atmosphere is specified in the for of the Gas flows. An oxidative anneal process can be added to the process flow defined in a python file as follows.

```
TempProfile = [800., 1100., 1100., 800.]
TimeIntervals = [60., 1200., 60.]
Gases = {"H2O" : [0., 20., 0.],
         "N2" : [20., 20., 20.]}
dioPro.anneal (Name="ann1",
               Temperature=TempProfile,
```

```
Time=TimeIntervals,
GasFlows=Gases)
```

In the Gas-name→Gas-flow-list python map defined in `Gases` variable, H_2O vapour flow is specified together with an inert N_2 gas. H_2O creates an oxidizing atmosphere in the middle time interval of 20 minutes (1200 seconds). When simulated, the above code will create a thick oxide (¿ 1μm) on Silicon surface. Additionally, if `ChargedPair` model is selected for the dopant diffusion, the generated interstitials at the top of Silicon wafer would alter the dopant diffusion rate during annealing. Notice, that no separate oxidizing anneal procedure is used. The regular `anneal (...)` procedure invoked on the 'process' object can simulate oxidative anneal, when an appropriate oxidizing atmosphere is specified by the gas flows.

8.2 Gas flows and oxidation

Various gases are used to create oxidizing atmosphere. The chamber gases recognized by the process simulator are listed in Section 7.1. Among them, `O2`, `H2O`, and `N2O` are the oxidizing gases. Additionally, mixing two or more gases can react to form an additional oxidizing gas in the chamber. The following reactions in the chamber are taken into account by the simulator.

$$2H_2 + O_2 \longrightarrow 2H_2O \tag{8.1}$$

$$O_2 + 4HCl \longrightarrow 2H_2O + Cl_2 \tag{8.2}$$

Gas flows of the product gases in the above reaction are determined from the gas flows of the reacting gases depending on which gas is the liming ingredient in the above reaction as follows.
If $f_{O_2}^{\text{ext}} > 0.5 f_{H_2}^{\text{ext}}$,

$$\begin{aligned} f_{O_2}^{\text{int}} &= f_{O_2}^{\text{ext}} - 0.5 f_{H_2}^{\text{ext}} \\ f_{H_2O}^{\text{int}} &= f_{H_2O}^{\text{ext}} + f_{H_2}^{\text{ext}} \\ f_{H_2}^{\text{int}} &= 0 \end{aligned} \tag{8.3}$$

else,

$$\begin{aligned} f_{O_2}^{\text{int}} &= 0 \\ f_{H_2O}^{\text{int}} &= f_{H_2O}^{\text{ext}} + 2f_{O_2}^{\text{ext}} \\ f_{H_2}^{\text{int}} &= f_{H_2}^{\text{ext}} - 2f_{O_2}^{\text{ext}} \end{aligned} \tag{8.4}$$

If $f_{O_2}^{\text{ext}} > 0.25 f_{\text{HCl}}^{\text{ext}}$,

$$\begin{aligned} f_{O_2}^{\text{int}} &= f_{O_2}^{\text{ext}} - 0.25 f_{\text{HCl}}^{\text{ext}} \\ f_{H_2O}^{\text{int}} &= f_{H_2O}^{\text{ext}} + 0.5 f_{\text{HCl}}^{\text{ext}} \\ f_{\text{HCl}}^{\text{int}} &= 0 \\ f_{Cl_2}^{int} &= 0.5 f_{\text{HCl}}^{\text{ext}} \end{aligned} \tag{8.5}$$

else,

$$\begin{aligned} f_{O_2}^{\text{int}} &= 0 \\ f_{H_2O}^{\text{int}} &= f_{H_2O}^{\text{ext}} + 2f_{O_2}^{\text{ext}} \\ f_{\text{HCl}}^{\text{int}} &= f_{\text{HCl}}^{\text{ext}} - 4f_{O_2}^{\text{ext}} \\ f_{Cl_2}^{int} &= 2f_{O_2}^{\text{ext}} \end{aligned} \tag{8.6}$$

In the above calculations, gas flows with the superscript *ext* are the exterior gas flows of the input gases (at the entrance of the chamber), whereas those with the superscript *int* are the interior gas flows in the chamber. If the given gas G is not reacting, the its interior flow is equal to the exterior gas flow ($f_G^{int} = f_G^{ext}$).

The oxidation rate is proportional to the partial pressure of the oxidizing gases. Partial pressure of various chamber gases is calculated by the following formula.

$$p_{\text{gas}} = p \cdot \frac{f_{\text{gas}}^{\text{int}}}{\sum_{\text{all-gases}} f_{\text{gas}}^{\text{int}}} \tag{8.7}$$

This value of the partial pressure is used for calculating the oxidation rate during oxidative annealing step.

8.3 Oxidation model

Oxidation rate at the surface of Silicon wafer is modeled by Deal-Grove model.

$$\frac{dx_{\text{ox}}}{dt} = \frac{B}{2x_{\text{ox}} + A} \tag{8.8}$$

Here, x_{ox} is the oxide thickness at any time t during annealing, A and B are the constant parameters defined below.

Using the above equation, the oxidation process may be divided into linear and quadratic regime.

Initially, when a thin oxide layer is present on Silicon surface, the oxidant gas molecules can readily diffuse through thin oxide layer and react with Si atoms at the surface. Since the diffusion through thin oxide layer is fast, oxidation rate is limited by the reaction rate at the surface, given by the linear rate constant $\frac{B}{A}$. In linear regime, oxide thickness is a linear function of the oxidation time.

As the oxide layer gets thicker, the oxidation rate gets slower and slower due to the limitations on diffusion through the thick oxide. This results in reduced oxidation rate. In this regime, oxide thickness is proportional to the square root of time. Hence, it is also called quadratic regime. The quadratic rate constant is denoted by B.

The quadratic rate constant B depends on the oxidation temperature. Two Arrhenius functions, applicable for low and high temperature oxidation processes, can be specified in the simulator, together with the temperature at which the transition from low to high temperature takes place. Expression for temperature dependence of B is given below.

$$B(T) = \begin{cases} B_{0,h} \cdot \exp\left(-\frac{B^E_{0,h}}{kT/q}\right), & \text{if } T > B_{\text{T,break}} \\ B_{0,l} \cdot \exp\left(-\frac{B^E_{0,l}}{kT/q}\right), & \text{otherwise} \end{cases} \tag{8.9}$$

The quadratic reaction rate B also depends on the partial pressure of the oxidizing gas. This dependence is given below.

$$B(p,T) = B(T)p_{gas}^{B_{p,dep}} \tag{8.10}$$

In the above defined expressions, B_h and B_l can be set as Arrhenius expressions by the `<param>` names `Bh` and `Bl`, respectively. The

parameters $B_{\mathrm{T,break}}$ and $B_{p,dep}$ are set by the `<param>` names `BTBreak`. Since the quadratic reaction rate signifies strength of diffusion process through the oxide, the parameter values are specific for the specific material. They are therefore set by using the following command.

```
SetParam*s0: {
  Command = "Oxide <gas> <param> <val>";
}
```

In the above command, `<gas>` can be any one of the oxidizing gases – `H2O`, `O2`, or `N2O`.

Similarly, reaction rate constant at Oxide-Silicon interface $\frac{B}{A}$ is given by the following expression.

$$\frac{B}{A}(T) = \begin{cases} B_{A,0,h} \cdot \exp\left(-\frac{B^E_{A,h}}{kT/q}\right), & \text{if } T > B_{\mathrm{AT,break}} \\ B_{A,0,l} \cdot \exp\left(-\frac{B^E_{A,l}}{kT/q}\right), & \text{otherwise} \end{cases} \tag{8.11}$$

In the above expression, $B_{A,h}$ and $B_{A,l}$ are set as Arrhenius expressions by the `<param>` names `BAh` and `BAl`, respectively. The threshold temperature $B_{\mathrm{AT,break}}$ is set by the `<param>` name `BATBreak`. Since the linear reaction rate constant $\frac{B}{A}(A)$ is interface dependent, the parameter values are specific of the material interface. They are set by using the following command.

```
SetParam*s0: {
  Command = "Oxide_Silicon <gas> <param> <val>";
}
```

8.4 Incorporating oxide in the structure

In Deal-Grove model, growth rate of the oxide layer on Silicon is given by Eq. 8.8. This equation has been implemented as an update equation for the oxidation rate in time-stepping iterations of annealing simulation. Mathematically it is given as follows.

$$\frac{x^i_{\mathrm{ox}}[q+1] - x^i_{\mathrm{ox}}[q]}{t[q+1] - t[q]} = \sum_{\mathrm{g}} \frac{B_{\mathrm{g}}(T[q], p[q])}{x^i_{\mathrm{ox}}[q] + A_{\mathrm{g}}(T[q], p[q])}$$

Here, $t[q]$ is the time from the beginning of the current anneal step and $x^i_{\rm ox}[q]$ is the oxide layer thickness at the vertex i at time $t[q]$. The oxidation parameters $B_{\rm g}(T[q], p[q])$ and $\frac{B}{A}(T[q])$ are computed for the anneal temperature $T[q]$ and the partial pressure of the oxidizing gas g. The oxidation rate due to each of the oxidizing gases in the chamber is added up to calculate the total oxidation rate. The above equation can be rearranged to calculate oxide layer thickness at the vertex i at the next time point $t[q+1]$ as follows.

$$x^i_{\rm ox}[q+1] = x^i_{\rm ox}[q] + (t[q+1] - t[q]) \cdot \left(\sum_{\rm g} \frac{B_{\rm g}(T[q], p[q])}{x^i_{\rm ox}[q] + A_{\rm g}(T[q], p[q])} \right) \tag{8.12}$$

Time integration in Eq. 8.12 is performed at each of the vertices at all the oxide-Silicon interfaces which have their oxide layer exposed to the gas. In solving the equation, oxide layer thickness at the beginning of annealing step ($x_{\rm ox}[0]$ at $t = 0$) is determined by computing minimum distance between the vertex at the oxide-gas interface the neighboring oxide layer

Time integration is also performed at each of the vertices at the Silicon-Gas interfaces. In this case, oxide layer thickness at the beginning of annealing step ($x_{\rm ox}[0]$) is set to the native oxide layer thickness on Silicon, i.e., 0.7nm.

Once annealing step is over, the oxide layer is incorporated into the structure by creating a new oxide region.

If oxidation is performed on bare Si surface, the above procedure would ignore dopant segregation into the oxide, due to the absence of initial oxide layer.

Chapter 9

Pseudo-Process Mode

In order to develop a viable process flow, it is necessary to put various process steps in a 'correct' order. Many-a-times, generation of a reasonably 'correct order' of the process steps involves beginning with a simple flow and performing multiple iterations over the course of development. A long process simulation time (due to multiple long anneal steps and implantation steps) could be a road-block for an iterative process-flow development. Process development of 3D structures becomes impractical.

'Pseudo-process mode' circumvents this drawback by replacing long anneal and implantation steps with their equivalent shapes purely for visualization purpose.

When 'pseudo-process mode' is active, the geometry-modifying process steps - `Etch` and `Deposit` are executed normally to create the desired final device from an initial structure (wafer).

`Implant` step is registered in the process-flow. Additionally, a sheet is placed wherever implantation takes place. This sheet demarks a *doped* zone.

Similarly, `Anneal` step is registered in the process-flow. In addition to that, all the doped zones in the device structure are expanded by using 'Offset' algorithm. This expansion length is specified by the user. It corresponds to increase of the diffusion length due to the annealing step. Average doping in the doped region is proportionately decreased.

At any point in the process flow, calling `saveDeviceAsSTEP(filename)` function on the process object stores the device structure to the STEP file together with the region names and material information. Additionally, all the doped zones are also stored for visualization purpose. Use of 'pseudo-process mode' is described with an example below.

9.1 Python file

Python script file in Chapter 2 is rewritten to activate and utilize pseudo-process mode as follows.

```
import process as pr
import numpy as np

# define process object
dioPro = pr.Process (Name="Diode", ProcessStepsFile="",
                  MeshType="FEM", PseudoProcessMode=True)

xmin = -0.24
xmax = 0.24
ymin = -0.2
ySiTop = 0.2
ymax = 1.0
zmax = 0
zmin = 0
posRef = [pr.position(xmin, ymin, zmin),
          pr.position(xmax, ymax, zmax)]
meshPar = {"MaxAreaBulk" : 0.01, "MaxAreaInterface" : 0.005}
# define RefWins
dioPro.setRefWin (Name="RefAll", Shape="Rectangle", Position=posRef,
                  NumericParams=meshPar, NumericListParams={})

posRef = [pr.position(xmin, ymin, zmin),
          pr.position(xmax, ySiTop, zmax)]
dioPro.setRefWin (Name="RefSi", Shape="Rectangle", Position=posRef,
                  NumericParams={}, NumericListParams={})
```

```
# define Regions
dioPro.setRegion (Name="RegBack", RefWin="RefAll", Material="Gas",
                  NumericParams=meshPar)

dioPro.setRegion (Name="RegSi", RefWin="RefSi", Material="Silicon",
                  NumericParams=meshPar)

# MESH GENERATION IS NOT NECESSARY FOR PSEUDOPROCESS MODE.
# dioPro.createDeviceMesh ()
# dioPro.saveMeshData ()

dioPro.preprocessDevice ()

# Save device created so far
dioPro.saveDeviceAsSTEP("InitStructure.stp")

# start process steps
dioPro.deposit (Name="DepOx", Material="Oxide",
                Type="Anisotropic", Thickness=0.4)

dioPro.saveDeviceAsSTEP("AfterDepOx.stp")

# define RefWin for Mask
arrTmp=[pr.position(xmin*0.6, 0., 0.),
pr.position(xmax*0.6, 0., 0.)]
dioPro.setRefWin (Name="OxEtMaskWin", Shape="Segment",
                  Position=arrTmp, NumericParams={},
                  NumericListParams={})

# define Mask for etching oxide
dioPro.mask (Name="OxEtMask", RefWinList=["OxEtMaskWin"])

# etch oxide
dioPro.etch (Name="EtOx", Material="Oxide", Type="Anisotropic",
             Depth=0.5, EtchMask="OxEtMask")

dioPro.saveDeviceAsSTEP("AfterEtOx.stp")
```

```
# implant Boron
dioPro.implant (Name="Impl1",Type="Ana", ImplantSpecies="Boron",
                Energy=90, Dose=1E12)

dioPro.saveDeviceAsSTEP("AfterImplB.step")

# strip oxide
dioPro.etch (Name="StripOx", Material="Oxide", Type="Strip")

dioPro.saveDeviceAsSTEP("AfterStripOx.step")

# set global parameters
dioPro.setParam (Name="s0", Material = "Silicon",
     Dopant = "BActive", Defect = "IntNeutral",
     ParamName = "Dstar", Charge = "0",
     ParamValue = "Arrhenius 0.0121 3.341")

# set numeric parameters for annealing step.
# create two python maps, one with string parameters and
#        one with numeric parameters
numPar = { "initstep" : 1E-4, "minstep" : 1E-7, "maxstep" : 10.0,
      "Pressure" : 1., "RelativeError" : 1.E0, "TolRes" : 1.0E-1,
      "TolStep" : 1.0E-1, "MaxIterations" : 15,
       "MaxInnerIterations" : 5, "UndampedIterations" : 0}
strPar = { "TimeStepping" : "BE"}
dioPro.setAnnealPar (Name="s1", NumericParMap=numPar,
                     StringParMap=strPar)

# perform annealing
TempProfile = [800., 950., 950., 800.]
TimeIntervals = [20., 5., 20.]
Gases = {"N2" : [20., 20., 20.]}
dioPro.anneal (Name="ann1", Temperature=TempProfile,
               Time=TimeIntervals, GasFlows=Gases,
               NumericParMap={"DiffusionLength" : 0.05})
dioPro.saveDeviceAsSTEP("AfterAnneal.step")
```

```
# dioPro.doProcessing ()
# dioPro.saveDevice ()
```

The above python script can be run using the following line.

```
>> python example_pro.py
```

The script is described in detail below.

9.1.1 Activate pseudo-process mode

Pseudo-process mode is activated by setting `PseudoProcessMode` to boolean `True` in the constructor of `process` object.

Various `RefWins`, `Regions`, and other entities are added just like before to generate initial structure. Mesh generation using `createDeviceMesh` and `saveMeshData` functions is not necessary for pseudo-process simulations. However, mesh generation is required to perform simulations using `doProcessing` function.

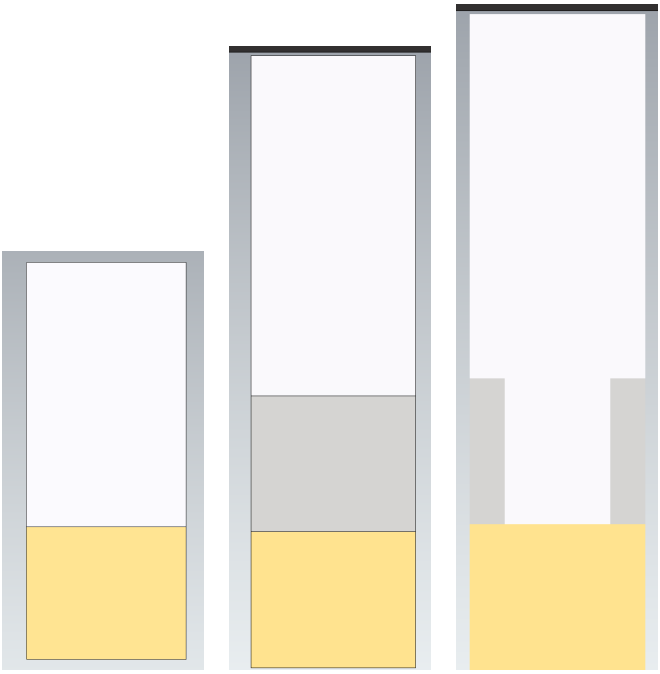
9.1.2 Saving STEP files

In the above code, STEP file is stored after each process step by calling `saveDeviceAsSTEP` function on `dioPro` object. The desired file-name is provided as an input parameter. The stored files can be opened by any of the STEP viewers, such as `cadassistant` by OpenCascade.

9.2 Results of pseudo-process simulation

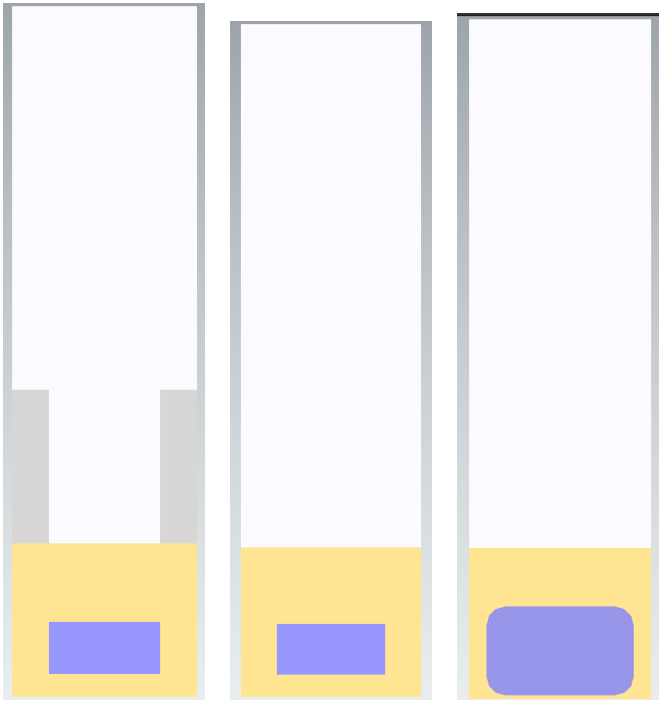
The above pseudo-process simulation runs in a fraction of a second, as opposed to the real process simulation which runs in a few 10s of minutes. The device structure stored after every step is displayed in Fig. 9.1 and Fig. 9.2. Additionally, clicking on 'Model Browser' button displays names of the regions along with the materials (see Fig. 9.3). Names of the as-implanted or annealed doped regions are also shown.

Notice, that after implantation a blue rectangle appears in Silicon region. This 'in vague terms' represents the doped zones. Blue and red colors represent p- and n- doped zones. Also, notice that the doped zone size increases after annealing. This vaguely represents diffusion process. This diffusion length during annealing is given by a numeric



(a) Initial Structure (b) After oxide deposition (c) After masked oxide etch

Figure 9.1: STEP files saved at the end of first three steps in the python script given at the beginning of this chapter. While color - background gas. Grey color - oxide. Yellow color - Silver.



(a) After implantation (b) After oxide mask removal (c) After annealing

Figure 9.2: STEP files saved at the end of last three steps in the python script given at the beginning of this chapter. While color - background gas. Grey color - oxide. Yellow color - Silver.

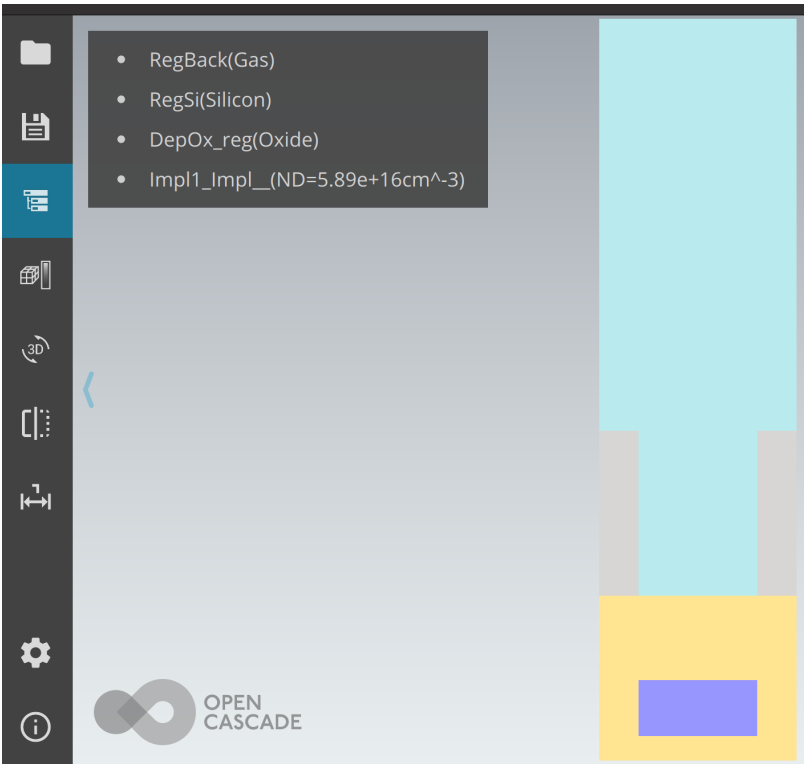


Figure 9.3: 'Cadassistant' program window showing the names of the regions and the doped zones.

parameter named `DiffusionLength` to along with `anneal` command. Diffusion length provided here is only 'incremental diffusion length' due to the `anneal` step in μm. Total diffusion length of any given doped zone is the sum of the 'incremental diffusion lengths' coming from all the `anneal` steps undergone by the doped zone.

Chapter 10

Graphical User-Interface

Process flow can also be created using a graphical-user-interface (GUI) of the process generator. The GUI can be opened using the following line -

```
processgen &
```

The above command opens a GUI window which consists of a 'Menu-bar', a 'Tool-bar', a structure drawing board, and a side-pane. The 'Menu-bar' contains various functions as drop-down menus, while the 'Tool-bar' contains some of the more common functions as short-cut buttons. The 'side-pane' contains a stack of different widgets. Each of the widgets contains an information on the structure, steps in the process flow, GDS layout to structure conversion, python script for generation, etc.

10.1 Generating Initial Structure

A quick look at the `processgen` GUI tool (see Fig. 10.1) and the `structuregen` GUI tool shows, that both the GUIs are identical. This is because, `processgen` GUI includes all the functionalities of `structuregen` GUI together with additional functions. A process simulation requires creating an initial device structure. This task is

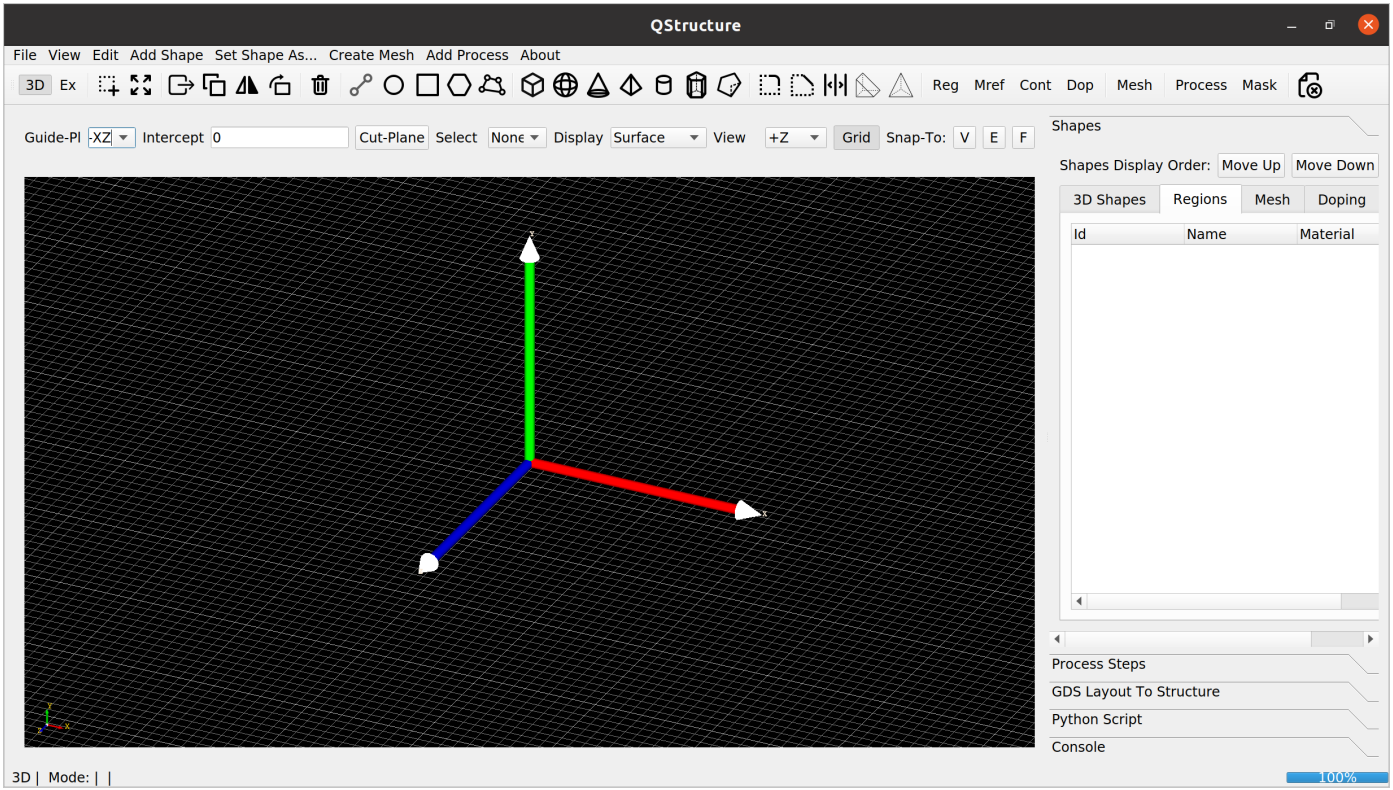


Figure 10.1: Graphical-user-interface of the process generation tool.

performed by `structuregen`like GUI. Please refer to the *Graphical User-Interface* chapter in *Structure Generation and Meshing User Manual* for further information on various functionalities of the GUI tool.

Functions which are unique to the `processgen` GUI tool are described below.

10.2 File-Menu

Functions of the file menu are similar to those in the structure generation tool, except -

- Open loads a previously saved process generation file (*.qprc) or a structure generation file (*.qstr) to the tool. If the file is a structure generation file, the structure stored in it serves as an initial structure.
- Save save a process generation file (*.qprc) only.

10.3 View menu

Functions of the view menu are similar to those in the structure generation tool, except there are two more buttons Process and Mask in `View` menu. Their functions are as follows.

10.3.1 Process

When clicked, it activates 'process' mode. New process steps can be added only when the process mode is active. Once activated, the process mode cannot be turned off. Therefore, it is necessary to generate initial structure before activating the process mode.

10.3.2 Mask

When clicked, it activates mask drawing mode. A 2D mask `RefWin` in XZ-plane can be created in this mode. A mask is generated from the 2D RefWin by clicking Add Process → Create Mask button.

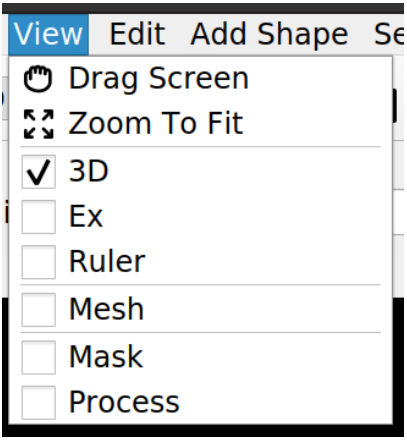


Figure 10.2: View menu.

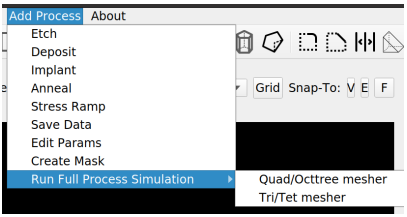


Figure 10.3: 'Add Process' menu.

10.4 Add Process menu

Various functions in the 'Add Process' menu are shown in Fig. 10.3. These functions are active *only* when the Process is active. Each of these functions assist the user in adding a new process step with the required set of parameters. Their usage is described below.

10.4.1 Etch

Clicking 'Etch' function opens 'Etch dialog box' which is shown in Fig. 10.4. The dialog box provides fields to setup various input

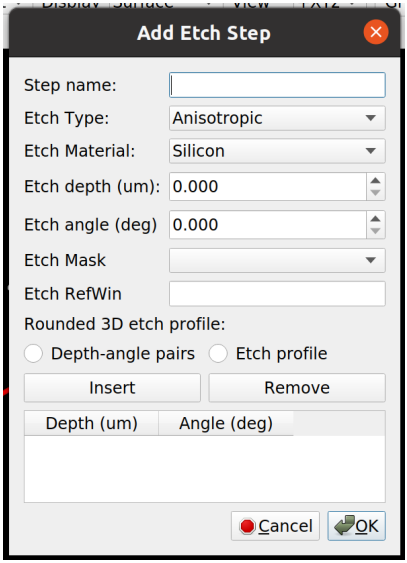


Figure 10.4: 'Etch' dialog box.

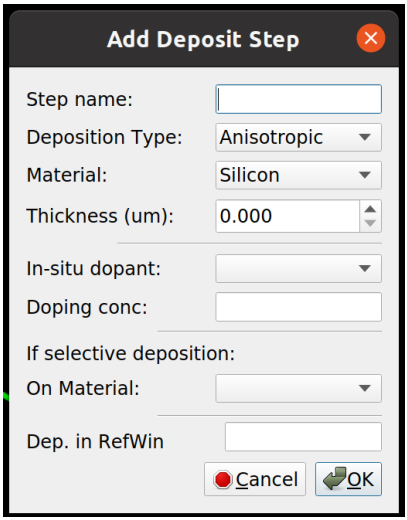


Figure 10.5: 'Deposit' dialog box.

parameters for simulating etch step. Masked etch can be performed by selecting appropriate etch mask. If no mask is selected, unmasked etch-step is performed. Similarly, etch `RefWin` is specified by giving name of the previously defined `RefWin` in the text field. Detailed description of various parameters corresponding to etch-step is given in Chapter 4 Sec. 4.2. Click Ok to add the etch-step at the end of the process-flow, otherwise no etch-step is added.

10.4.2 Deposit

Clicking 'Deposit' function opens 'Deposit dialog box' as shown in Fig. 10.5. The dialog box provides fields to setup various input parameters for simulating deposit-step. For custom deposition of a material in a user-defined RefWin shape, specify the name of the previously defined `RefWin` in the text field 'Dep. in RefWin'. Also, in-situ doping can be set by specifying dopant-type and dopant concentration. Detailed description of various parameters corresponding to etch-step is given

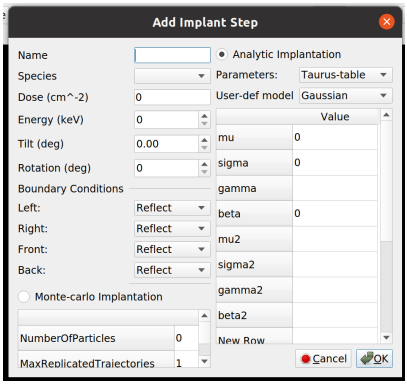


Figure 10.6: 'Implant' dialog box.

in Chapter 4 Sec. 4.1. Click Ok to add the deposit-step at the end of the process-flow.

10.4.3 Implant

Clicking 'Implant' function opens 'Implant dialog box' as shown in Fig. 10.6. The dialog box provides fields to setup various input parameters for simulating implantation-step. If *analytic implantation* method is selected, the implantation parameters are read from stored implant-tables. If user-defined parameters are to be used, the user can set the distribution-type and corresponding parameters in the table below. Detailed description of various parameters corresponding to etch-step is given in Chapter 5. Click Ok to add the implantation-step at the end of the process-flow.

10.4.4 Anneal

Clicking 'Anneal' function opens 'Anneal dialog box' as shown in Fig. 10.7. The dialog box provides fields to setup various input parameters for simulating anneal-step. Detailed description of various parameters corresponding to etch-step is given in Chapter 7. Click Ok to add the anneal-step at the end of the process-flow.

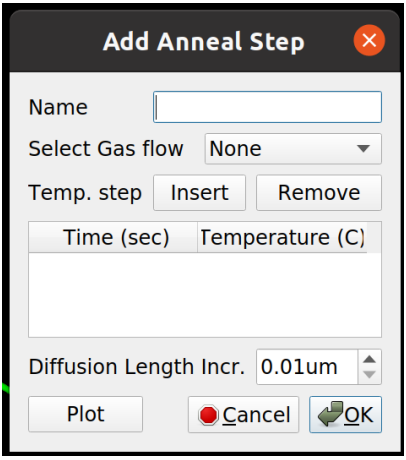


Figure 10.7: 'Anneal' dialog box.

User can specify ramp-time (in sec.) and corresponding ramp-temperature (in degC) in the table. To add a new temperature step at the end of the table, click Insert. To delete a row, click on the row and then click Remove. Note, that time-points must be listed in increasing order of time. Temperature is linearly interpolated between two consecutive time-points.

Gas-flow rate at any given time-point is specified in the successive columns. Select gas name in the drop-down box 'Select Gas-flow'. If the gas-name columns is already present, it will be highlighted. Else a new column with the selected gas-name is added at the end.

To visualize gas flows and temperature-ramp, click Plot. It will open a graph which plots gas flows and temperature-ramps.

Specify approximate diffusion length increment due to the given anneal step. This value is used *only* for quick visualization purpose. Exact diffusion length can be calculated after simulating the anneal step.

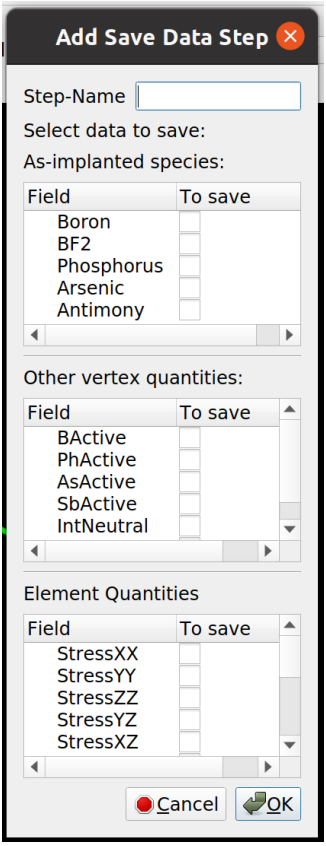


Figure 10.8: 'Save Data' dialog box.

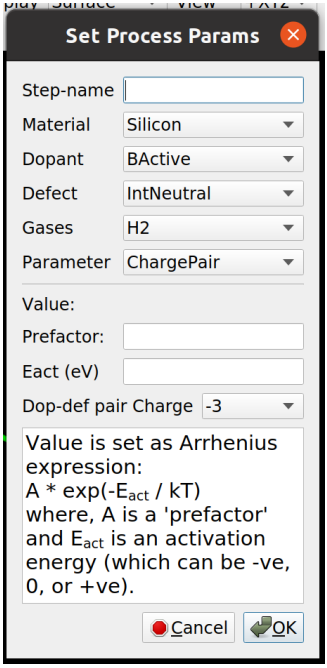


Figure 10.9: 'Save Data' dialog box.

10.4.5 Save Data

Clicking 'Save Data' function opens 'Save Data dialog box' as shown in Fig. 10.8. All the available 'As-implanted species', 'Vertex Quantities', and 'Element Quantities' are listed in three different lists. Please tick the boxes on the quantities which need to be stored. A save-step is added after last step of the process flow.

10.4.6 Edit Params

Clicking 'Set Process Params' function opens 'Set Process Params dialog box' as shown in Fig. 10.9. The dialog box provides input parameters which are needed to set a specific input parameter specific to the given material, dopant, defect, or gases. Dopants, defects,

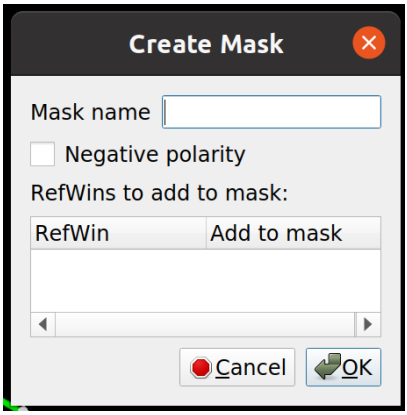


Figure 10.10: 'Create Mask' dialog box.

or gases field can be empty (not all can be empty at the same time). Material field must contain a valid material name. All the editable process simulation parameters corresponding to the specified combination of material, dopant, defect, or gases are listed in the drop-down list. Select the parameter. Then give its value as a prefactor and activation energy of an Arrhenius. If the parameter is just a number, activation energy can be set to zero. Click Ok to add the edit-parameter-step at the end.

10.4.7 Create Mask

Clicking 'Create Mask' function opens 'Create Mask dialog box' as shown in Fig. 10.11. Set name of the mask. The dialog box provides a list of mask `RefWins`. Select the `RefWins` to be added to the current mask. Click Ok to add a new mask to the process generator.

10.4.8 Run Full Process Simulation

Clicking on this option opens a new sub-menu with two options. They are as follows.

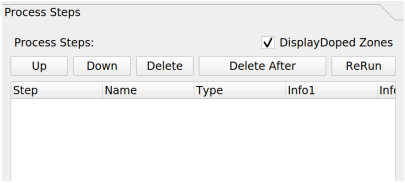


Figure 10.11: 'Process Steps' tab on the side pane

- `Quad/Octtree mesher` : Use Quad- (2D) or Oct- (3D) Tree mesher for process simulation.
- `Tri/Tet mesher` : Use Triangle (2D) or TetGen (3D) mesher for process simulation.

When clicked, full process simulation is performed from initial structure till the last process step. Once completed, the final device structure is displayed in the drawing window. Console output is displayed in the text box in the 'Console Tab'.

10.5 Process Steps Tab

Process steps tab on the side-pane shows the list of processes added to the process flow. First column states the step (etch, deposit, anneal, implant, etc.) and second column states the process name. If the `type` of the process step is specified, it is listed in the third column. Fourth and fifth columns provide additional information about the process step as follows. Sixth column states unique id corresponding to the step.

Process steps are executed in the same order as shown in the list. In order to move a step up or down, click on the process step and click Up or Down buttons, respectively. To delete a step, click on the step and click Delete. To delete all the process steps *after* a specific step, click on the step and click Delete After. After editing the process flow, to update the structure in the drawing board based on the process flow, press ReRun. It performs 'Pseudo-process simulation', not full process simulation. Different 'doped zones' are

Table 10.1: Additional information provided in fourth and fifth columns

Step	**Info1**	**Info2**
Etch	Material	Depth (μm)
Deposit	Material	Thickness (μm)
Implant	Species	Dose (cm^{-3})
Anneal	Max. Temperature	Anneal-type
Set-parameter	Dopant	Defect

created during pseudo-simulations. If 'Display doped zones' check-box is checked, then 'doped-zones' are also displayed together with the structure. Unchecking the check-box hides the 'doped-zones'.

Appendix A

Notation and Acronyms

Acronyms

BC Boundary Condition

MC Monte-Carlo

Bibliography