Firebird Software User Manual

Version 2.2

Richard Neill Finn

November 11, 2024

Contents

List of Code Listings

1 Introduction

This document serves as the user manual for Firebird - the lightweight, FEniCS based thermal calculation tool for cryogenic integrated circuits, developed as a final year project by Richard Neill Finn in 2024. It offers guidelines to the installation, configuration and usage of the tool. This manual is satisfactorily comprehensive, though not complete and will not cover **all** possible use cases. However, the most common ones are discussed, along with possible mistakes the user can make and how to avoid them.

Any resources not developed by the original author have been referenced and can be viewed using the provided links. Support for tools not written as part of Firebird is the responsibility of the authors of that particular product. Please visit their respective websites for any enquiries.

2 Installation and Configuration

This section covers the installation instructions for using Firebird.

2.1 Prerequisites

In order to start using Firebird, the following listed items are required. Note that Firebird can be run on any machine with the correct installations, though installing and running on a Unix based Operating System (MacOS or Linux) is recommended. This allows Firebird to make use of some parallel computing frameworks, such as PETSc and MPI.

- Python[1]
- FEniCS (recommend using conda installer)[2]
- GMSH[3]

The following packages are required for FEniCS operations to be performed. These can be installed using a package manager such as Homebrew[4] (if on MacOS and Linux), using `pip install` commands, or Conda[5]. A full list is given in Appendix A.

- fenics-basicx - 0.7.0+
- fenics-dolfinx - 0.7.3+
- fenics-ffcx - 0.7.0+
- fenics-ufl - 2023.2.0+
- gmsh - 4.13.1+
- meshio - 5.3.5+
- mpi4py - 3.1.6+
- petsc4py - 3.20.5+

[1] https://www.python.org/downloads/
[2] https://fenicsproject.org/download/
[3] https://gmsh.info
[4] https://brew.sh/
[5] https://anaconda.org/anaconda/conda

3 Usage of the tool

This section covers the usage of Firebird and possible issues that can arise during usage.

3.1 Required files

Firebird was written to analyse existing IC layouts. As such, it expects the user to provide it with a file that contains the layout information, along with two additional files that specify some simulation parameters and layout data.

3.1.1 GDS file

GDSII files contain the layout and are written in a layered fashion. Physical layers on the IC chip layouts are represented by layer numbers in a GDS file. GDS files also contain `cells`, which are like blueprints for other elements which are added (and often repeated) over the main layout. Firebird requires the layout to be provided by a GDSII file, saved in the `.gds` format. A typical GDSII file is written in binary, through when exported and viewed in a ASCII format as text, it should look similar to what is shown here:

```
BGNSTR 10/18/2024 19:37:22 10/18/2024 19:37:22
STRNAME JJ-Resist

BOUNDARY
LAYER 1
DATATYPE 0
XY -83000: -24000
-83000: 24000
-73000: 24000
-73000: -24000
-83000: -24000
ENDEL
```

Listing 1: Example output from GDSII file

When opened in an appropriate design tool, such as KLayout[6], the output should visually appear as something similar to what is shown here in Figure 1.

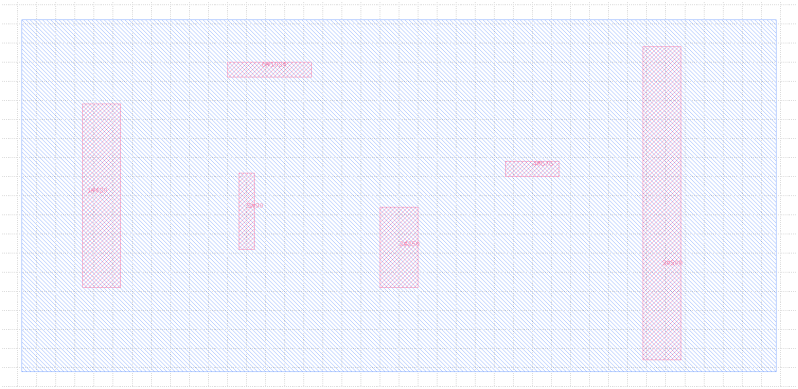


Figure 1: Example output from a GDS file that Firebird expects

3.1.2 Layer Definition File

The layer definition file is used in conjunction with the GDS file to provide Firebird with the necessary information to identify the different layers and their properties. This file is expected to be written and saved in XML format, a common file type used to mark data in a hierarchical way. XML works by using tags to nest different layers and material properties of that specific layer. An example of the expected XML data is shown in Listing 2 (full example in Appendix **??**).

As seen in the example code, each layer's material properties need to be listed. The properties of interest are the **Layer Numbers**, **Material type**, **Thermal conductivity**, **Specific Heat Capacity** and the **Density**. While other properties can be listed, Firebird will only use the ones mentioned here.

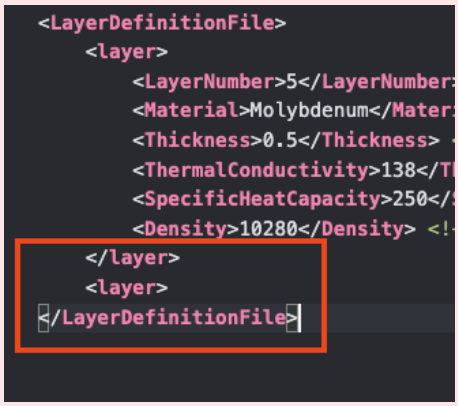
```
<ThermalConductivity>138</ThermalConductivity> <!-- W/mK-->
<SpecificHeatCapacity>250</SpecificHeatCapacity> <!-- J/kgK-->
<Density>10280</Density> <!-- kg/m3-->
```

Listing 2: Example output from layer definition file

[6] https://www.klayout.de/build.html

Possible error!

Ensure that there are no floating tags or any absences of closing tags. This will cause Firebird to be unable to parse the file, cause an error, and the program will ask if it needs to restart. If this happens, check the content of the XML file and run the program again. In the example on the right, the second `<layer>` tag has no closing tag `</layer>` indicating another layer in the layout.



3.1.3 Parameter file

The parameter file is used to specify simulation parameters that Firebird requires. These have default values written into Firebird, but is meant to allow the user to customize their output and simulation. Below in Listing 3 is an example of the parameters expected in a parameter file:

```
delta=0.1
substrate=5
resist=1
iterations=10
ambient=12
s_sense=2.0
```

Listing 3: Example output from parameter file

Suppose one wants to simulate n seconds worth of temperature change over the chip layout. One can imagine dividing this time frame into smaller segments, each representing the temperature profile at that specific time instance. This is where the `delta` and `iterations` parameters are used.

- The `delta` value sets the time between time segments. This can be thought of as difference in time from one frame to the next.
- The `iterations` value sets the number of time frames the output will

produce. This allows the user to set which specific instances in time they would like to capture. This number directly sets the number of output files that will be generated.

As an example, suppose one wants to observe changes every 20 microseconds, over a time frame of 0.1 milliseconds:

$$0.1 \text{ milliseconds} = 100\mu s$$
$$100\mu s \div \text{increments of } 20\mu s = 5 \text{ iterations}$$
$$\implies \texttt{delta} = 20, \quad \texttt{iterations} = 5$$

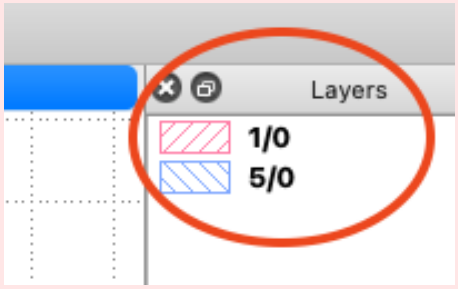
Possible error!

Ensure that the neither the number of iterations, nor the delta value is negative. This will stop the simulation setup and the program will ask if it needs to restart. If this happens, check the values in the parameter file and run the program again.

The `substrate` and `resist` values represent the layer numbers of the substrate and resistive layers. These must coincide with the layout file data.

Possible error!

Ensure the layer numbers listed in the parameter file match what was designed in the layout file. The layer numbers can be seen in KLayout (in a default layout) in the top right corner of the screen:



The `ambient` value sets the (ambient) temperature of the surrounding environment. This value is used in the determination of the boundary conditions when formulating the PDE problem over the chip layout. In a practical sense, this would be determined by the cooling method used, although it has a defualt value of 4K.

The `s_sense` value determines the characteristic length of the mesh vertices when meshing over the substrate. This value directly influences the number of mesh elements that will be generated and thus also affects the rendering time (which differs between machines with different hardware). It is used

when placing a `Point` in GMSH to determine the mesh density around that point.

Warning!

Setting this value too low can cause a **severe** increase in the rendering time. The default value is the recommended value, which is $\frac{1}{10}$th the value of the smallest dimension of the substrate. If this value is set too high, the mesh resolution may not capture the temperature gradient accurately.

3.2 Running Firebird in terminal

Firebird is a command line-based program that needs to be run within a FEniCS-enabled Python environment.

3.2.1 Directory and Environment

If all the required files are in place, the first step is to open a command terminal. In the guide below, the program is run in a Ubuntu terminal, though the command actions are the same regardless of the Operating System (the syntax and command calls may differ). Enter the following command to get started by navigating to the current working directory where all the required files (setup files) are present:

```
rnfinn@Neill-ZenBook:~$ cd Academics/2024/Skripsie/main/
```

Figure 2: Navigating to the correct directory. The parameter, layout and layer definition files are in the `main` folder.

Next, run the following command to set up the environment variables required to use Conda, allowing Conda commands to be enabled in the terminal session.

```
rnfinn@Neill-ZenBook:~/Academics/2024/Skripsie/main$ source ~/miniconda3/bin/activate
```

Figure 3: Changing the source

After enabling the commands, run the following Conda command to activate the FEniCS environment. Note that in this case, the environment is called

`fenix-env` as to differentiate it from other FEniCS related names, though this name is arbitrary:

```
(base) rnfinn@Neill-ZenBook:~/Academics/2024/Skripsie/main$ conda activate fenix-env
(fenix-env) rnfinn@Neill-ZenBook:~/Academics/2024/Skripsie/main$
```

Figure 4: Activation of the FEniCS environment

Note the change in environment as listed on the left hand side of the output. If this is successful, the setup for running in the correct environment is complete and a simulation can be run.

3.2.2 Running a simulation

Running a simulation requires calling the main `Firebird.py` file, along with passing the required setup files as arguments. An example of a call to run a simulation can be seen here:

```
$ python Firebird.py skripsieTest-Large.gds largeLayers.xml largeParams.txt
```

Figure 5: Calling Firebird to run a simulation

Possible error!

Ensure that the files passed in this command are in the correct order as shown in Figure 5. Help on which file is expected can also be found by running "`python Firebird.py –help`" in the terminal window.

The time it takes for a simulation to runs depends on the parameters that were passed to it in the appropriate file. An increase in the number of steps or the mesh density will cause the simulation to run for longer. This is influenced by the machine it is running on as well, as a faster CPU and improved memory will lead to faster simulation times.

3.3 Output files

The generated output files are saved in the (legacy) VTK format and can be opened by any compatible viewer. A popular viewing program such as ParaView[7] can be used to visualize the output, though the file contents can be inspected using a text editor, such as Notepad++.

Each VTK file has headings that mark the start of a section of data. These headings are (from top to bottom):

- Points (grid points indicating the node locations)
- Mesh cells
- Cell types
- Temperature values

An example of the content found in the VTK file is shown here, which is a section containing the scalar values for the temperatures at mesh nodes. The number following the `POINT_DATA` text is the number of nodes.

```
POINT_DATA 34935
SCALARS temperature float 1
LOOKUP_TABLE default
4.0
4.0
-1.3063084956876307
-0.42526550141847375
-1.222068762724175
-1.2934752944816945
0.16268253680358083
4.0
0.550868958569401
0.21475545580817065
0.34774848305623424
4.0
-0.9244261727223888
-0.07559256493456679
```

Listing 4: Example content from output VTK file

[7] https://www.paraview.org/

When viewed in ParaView, the scalar temperature values are selected for viewing by default and should appear similar to Figure 6.

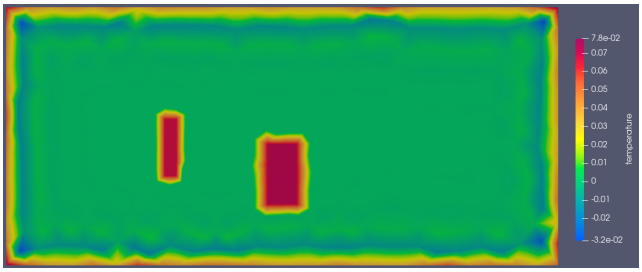


Figure 6: Example of temperatures visualized in ParaView

A Appendix A

Package	Version
aiohttp	3.9.5
aiosignal	1.3.1
async-timeout	4.0.3
attrs	23.2.0
Brotli	1.1.0
certifi	2024.8.30
cffi	1.16.0
charset-normalizer	3.3.2
click	8.1.7
contourpy	1.1.1
cycler	0.12.1
fenics-basix	0.7.0
fenics-dolfinx	0.7.3
fenics-ffcx	0.7.0
fenics-ufl	2023.2.0
fonttools	4.53.1
frozenlist	1.4.1
gdspy	1.6.13
gmsh	4.13.1
h2	4.1.0
h5py	3.11.0
hpack	4.0.0
hyperframe	6.0.1
idna	3.7
importlib_resources	6.4.0
kiwisolver	1.4.5
loguru	0.7.2
markdown-it-py	3.0.0

```
matplotlib          3.7.3
mdurl               0.1.2
meshio              5.3.5
mpi4py              3.1.6
msgpack             1.0.8
multidict           6.0.5
munkres             1.1.4
numpy               1.24.4
packaging           24.1
petsc4py            3.20.5
pillow              10.4.0
pip                 24.0
platformdirs        4.2.2
pooch               1.8.2
pycparser           2.22
Pygments            2.18.0
pyparsing           3.1.2
PySocks             1.7.1
python-dateutil     2.9.0
pyvista             0.44.1
requests            2.32.3
rich                13.9.2
scooby              0.10.0
setuptools          69.5.1
six                 1.16.0
slepc4py            3.20.2
tqdm                4.66.5
typing_extensions   4.12.2
unicodedata2        15.1.0
urllib3             2.2.2
vtk                 9.3.1
wheel               0.43.0
wslink              2.1.1
yarl                1.9.4
zipp                3.19.2
zstandard           0.23.0
```