OpenPCells

Technology Files Guide and Format

Patrick Kurth

2023-06-29

This is the official documentation of the OpenPCells project. It is split in several different files for clarity. This document provides an overview of the creation of technology files. If you are looking for a general overview of the project and how to use it, start with the user guide, which also contains a tutorial for getting started quickly. If you are looking for a guide and documentation on the creation of parametrized cells, consult the celldesign manual. If you want to now more about the technical details and implementation notes, look into the technical documentation.

1 Introduction

The pcells are defined in general layers (such as ”gate” or ”M1” or ”lastmetal”), which have to be translated into a specific technology for cell generation. Furthermore, vias and contacts have to be put into arrays, which requires knowledge about the technology. The technology-defining files (the “techfiles”) describe the required rules and layer mappings. In order to make it easy to start with techfiles they only need to contain what is required by the generated cell. That means for example that an inductor with only one metal layer only the mapping for that metal is required. If a cell is generated where some layers are missing in the techfile, you will see the following message (with the requested layer listed, in this case `metal(2)`):

```
generics: got NULL layer: generics.metal(2)
if this layer is not needed, set it to {}
```

This message shows you which layer has to be added to the techfile in order to make the cell run successfully. This guide will explain how to add these layers and which information is required.

2 Adding Technologies

A technology definition is a collection of files in one directory. These files get loaded by opc when requested as technology, the directory has to be within one of the paths in the technology search path (see the --techpath command line option). OpenPCells expects a technology to provide at least three files: config.lua, layermap.lua and vias.lua. All these files return one lua table, so a basic (empty technology) has the following content for all of these files:

```lua
return {}
```

Of course, this does not really make sense, so the following sections explore what (and when) to put into these files.

2.1 Config

The config is pretty simple, as it only requires a few keys set. Currently, only the number of metals in the stack is necessary for some calculations in specific cells, but the other options will be included in the project in the future. More options are also likely to come.

```lua
return {
    grid = 1, -- in nanometers,
    substrate_dopand = "p-substrate", -- dopand type of the substrate
    has_triple_well = true, -- support for tripe-wells
    is_SOI = false, -- silicon-on-insulator node
    FEOL_method = "active_plus_implant", -- method for specifying
       active (transistor) regions
    has_gatecut = false, -- special layer for cutting gates
    metals = 4, -- number of metals in the stack
}
```

The grid defines the granularity of the shapes, metals is the total number of available metal (and interconnect) layers in this technology node. substrate_dopand and has_triple_well configure the wells. is_SOI is a simple boolean switch whether this technology node is a silicon-on-insulator process. The front-end-of-line method (FEOL_method) configures the way active transistor regions are specified. Supported options are active_plus_implant, dedicated_active or asymmetric_active.

2.2 Layermap

The layermap includes information on the human-readable layer data as well as the stream numbers (virtuoso could also work just with the stream numbers, but often the layers have internal numbers that are NOT the stream numbers). Therefore every entry is a table containing a table for the layer and a table for the purpose:

```lua
-- example for metal 1
M1 = {
    name = "metal1",
    layer = {
        gds = { layer = 8, purpose = 0 },
        SKILL = { layer = "metal1", purpose = "drawing" },
        svg = { style = "metal1", order = 4, color = "0000ff" },
    }
},
-- example of an unused layer
notused = {},
```

The needed layers depends on the cells that are being used, but the program will also tell you when you are missing something. Therefore, you can also keep running it until it works. The 'opc' technology layer map contains most of the used layers and can be used as a reference.

The `name` field is optional, but helps when debugging cells. In fact, only one entry in the `layer` subtable for the corresponding export type is required.

2.3 Vias

The via rules file defines the via/contact geometries. In general, via regions (specified as rectangles within cells) are turned into arrays of individual cuts, defined by the via technology file. Therefore, the cut sizing must be given (`width` and `height`) as well as minimum space in `x`- and y-direction as well as outer enclosure (often different then inter-cut spacing).

Multiple entries are possible to accomodate the different ways of drawing vias. Furthermore, via entries can be restricted to certain maximum widths and/or heights of the via regions, with the keys `maxwidth` and `maxheight`.

```
viaM1M2 = {
    entries = {
        {
            width = 100, height = 100,
            xspace = 200, yspace = 200,
            xenclosure = 50, yenclosure = 50
        },
        {
            width = 100, height = 100,
            xspace = 200, yspace = 200,
            xenclosure = 100, yenclosure = 20
        },
    },
},
```

2.4 Summary

This already concludes the information on writing techfiles. There are some subleties here and there, but this document should be enough to get a basic technology mapping going. In general, the more advanced the used cells are, the more complex these files will get. This means on the other hand that only a few lines in each of these files are already enough to start using some of the cells and learning the opc workflow.