OpenPCells

Technical Documentation and Implementation Notes

Patrick Kurth

June 7, 2021

1 Technology Mapping

Mapping from generic cell descriptions to technology-specific data has to perform several steps:

- resolve relative metal numbering
- split up via stacks
- translate via rectangles to via arrays
- map all remaining[1] layers

Figure 1 shows the technology translation from generic to specific layers. The process is discussed in detail further on in this document. This example technology has 7 metal layers (labeled "M1" to "M7"), therefor "M-2" points to "M6". Cell developers have to have a possibility of specifying

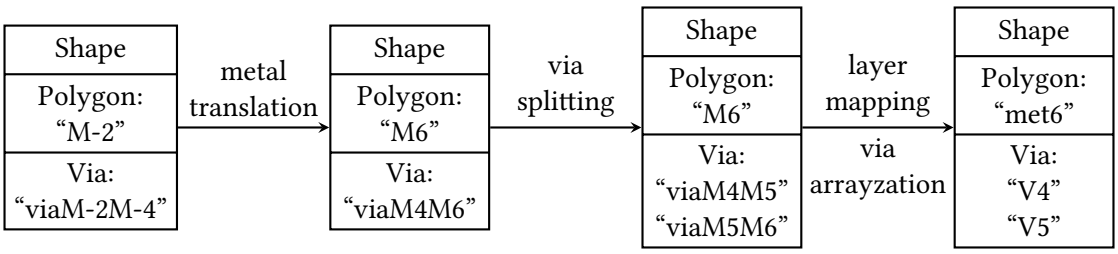

Figure 1: Technology translation

generic layers (such as "gate"), since there can not be any technology-specific information in the cells. This is achieved by using the generics module. The structures provided by this are crucial for the technology translation. Table 1 shows the currently available generics.

[1] The via translation already generates technology-specific layers.

generics	Meaning	Action
metal	Any metal (including local interconnects)	Resolve relative numbering
via	A via between two metals	Remove dummy shape and place cuts
contact	A conctact from the first metal to a semiconductor layer	Remove dummy shape and place cuts
other	Anything that is nothing of the above	Perform regular layer mapping
mapped	Already mapped layer (real process layer)	Not to be used by cells, generated by technology translation

Table 1: Available generics

1.1 Metal Numbering

For some cells like inductors it is customary to specify things like *last metal* or a metal relative to another. This has to be resolved for further processing, which is done in this step. Currently, only negative numbers (such as "M-1") are being processed into something like "M8" (depending on the total number of metals in the technology).

1.2 Via Splitting

It is allowed the create via stacks, that is vias with non-adjacent metals. These have to split up into several shapes before via arrayzation.

1.3 Via Arrayzation

Via geometries can't be inside generic PCells, since these vary from technology to technology. For this reason, only rectangular areas where vias are to be placed in a cell are specified. The technology translation then must create the actual via shapes, as shown in figure 2. For this process, the technology file has to define the size of the individual cuts (width and height), the spacing in x- and y-direction as well as the minimum enclosure of the surrounding layers (metals). The generated cuts fullfill the following equations:

$$\text{width} = N_x \cdot W + (N_x - 1) \cdot S_x + 2 \cdot E_x$$
$$\text{height} = N_y \cdot H + (N_y - 1) \cdot S_y + 2 \cdot E_y$$

For the generation of the cuts, these equations are solved for N_x and N_y.

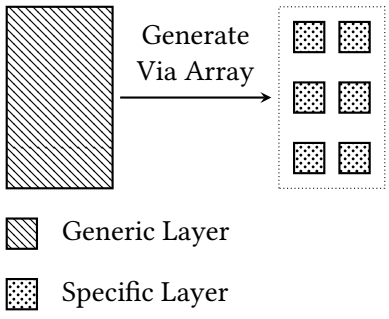

Figure 2: Example of via arrayzation

Figure 3: Possible topologies for rectangle union

1.4 Layer Mapping

The stage of layer mapping is the last of the technology translation, as some earlier stages also create to-be-mapped layers (for instance contacts will always generate instances of "M1"). Layer mapping is a simple process, as no shapes have to be modified. Here only "mapped" and "other" layers remain, where the former are naturally ignored.

2 Cell Parameterization

Cell parameters are defined in the `parameters()` function via the `pcell` module. It uses one state to store all parameters, so everytime a layout is created, the parameters have to be defined first. A different approach to this would be to store all loaded parameters and retrieve the right ones when a cell layout is created. Since currently there are no performance issues, no storing is made as code complexity increases (one has to check if loaded parameters exist etc.).

3 Shape Reduction

The generation process generates a lot of shapes that overlap partially or fully. Certain combinations of shapes can be merged, reducing the overall complexity (while maintaining the functionality of the layout, of course). Currently, a simple algorithm to merge rectangles is implemented, as rectanges usually make up the main part of a layout for integrated circuits. Union of polygons is much more complex, logic- and computation-wise.

The implemented algorithm simply checks if two rectangles can be merged into a rectangle by looking for certain topologies. All possible cases are shown in figure 3. Two independent checks are run for x and y, only x topologies are shown.