Адаптация DRC и LVS правил для техпроцессов ”тоньше” 110 нм под работу с открытыми инструментами

Микроэлектроника 2023

Уманский М.В., Елизаров С.Г., Лукьянченко Г.А, Черных Е.А.
ООО «Мальт систем»

MALT system

Содержание

РОССИЙСКИЙ ФОРУМ МИКРОЭЛЕКТРОНИКА 2023

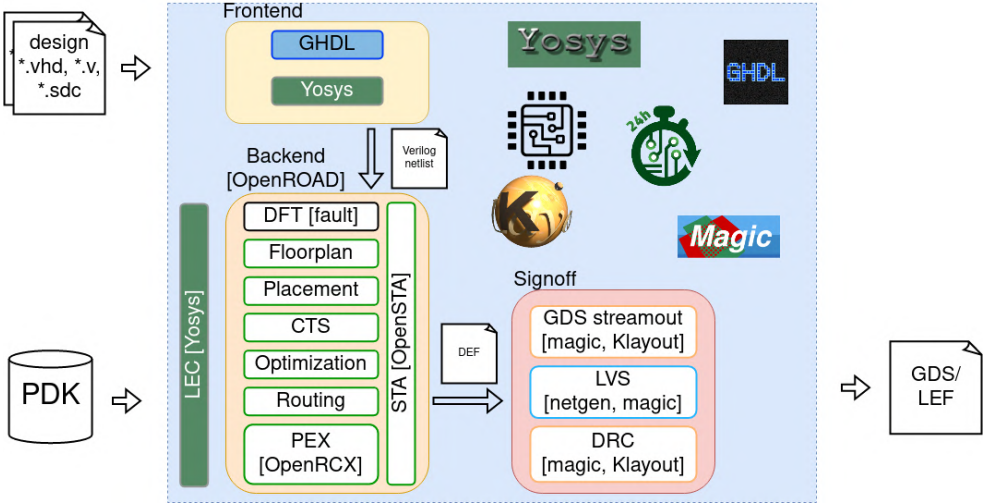
Введение

Введение

РОССИЙСКИЙ ФОРУМ МИКРОЭЛЕКТРОНИКА 2023

- Открытые инструменты достигли достаточного уровня для разработки реальных промышленных микросхем;
- Остро стоит вопрос адаптации отечественных и зарубежных коммерческих PDK для открытых инструментов;
- Одной из важнейших стадий адаптации PDK является портирование DRC и LVS скриптов;
- В открытых источниках много информации о использовании открытых САПР для техпроцессов 130 нм и ”толще”;
- В этом докладе мы рассмотрим пример проведения DRC и LVS проверок для технологического процесса 28 нм.

Открытый маршрут OpenLane



Адаптация PDK к маршруту

Из чего состоит PDK?

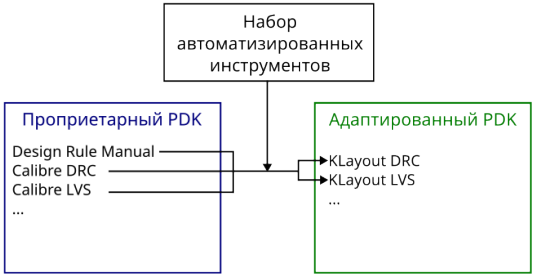
- DRM - Design Rule Manual;
- Скрипты для DRC (Design Rule Checking);
- Скрипты для LVS (Layout Versus Schematic);
- Скрипты для паразитной экстракции (PEX);
- Модели примитивов (devices);
- Библиотеки ячеек.

Открытые и закрытые форматы

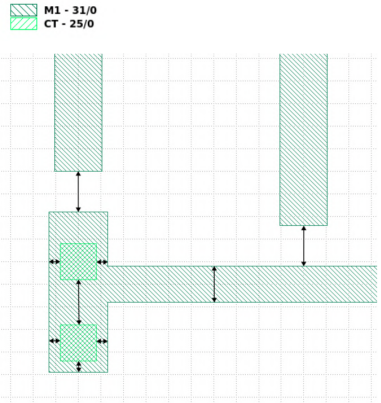
Элементы PDK	Форматы	Проприетарность
DRC-скрипты	SVRF	Да
LVS-скрипты	SVRF	Да
PEX-скрипты	Calibre xRC	Да
Модели для аналоговой симуляции	SPICE	Нет
Layout ячеек	GDS II	Нет
Нетлисты ячеек	SPICE	Нет
Результаты характеризации	Liberty	Нет
Цифровые модели ячеек	Verilog	Нет
Топология	LEF/DEF	Нет

В чем состоит адаптация PDK к маршруту

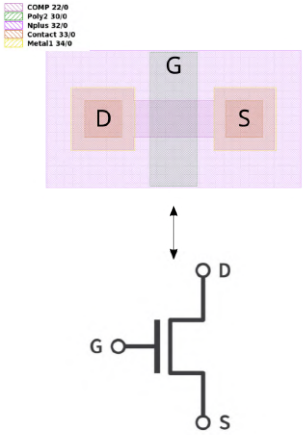
- Получение LVS и DRC скриптов, исполняемых открытыми инструментами;
- Написание технологических файлов для требующих этого инструментов;
- Генерация библиотеки ячеек и технологического LEF;
- Для упрощения процесса можно использовать автоматизированные инструменты.



DRC и LVS



DRC



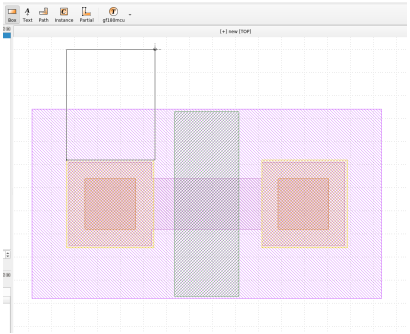
LVS

KLayout

Возможности KLayout

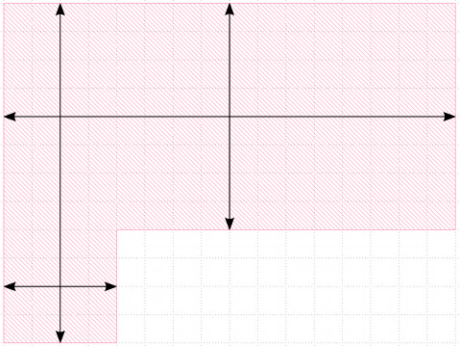
- Просмотр, генерация, редактирование GDS;
- GUI, а также Ruby и Python API;
- Параметризованные ячейки на Python (PCell);
- Движок для выполнения DRC и LVS;
- Обширная и подробная документация.

KLayout GUI & API

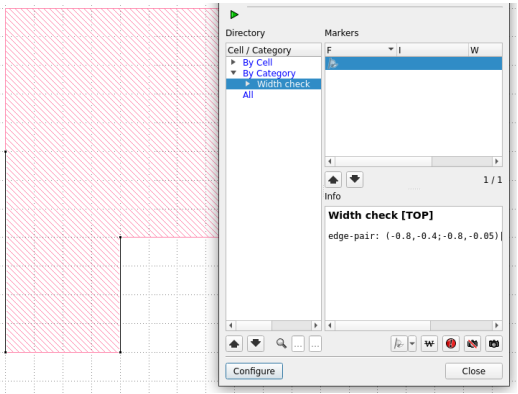


```
# Inserting a contact cell
cont_cell_index = layout.add_cell("contact")
cont_cell       = layout.cell(cont_cell_index)

# Inserting shapes now into the *contact* cell
cont_cell.shapes(contact).insert(pya.Box.new(0, 0, cont_size, cont_size))

# Contact array count and positions
nx = int((ld - (cont_size+cmp2cont+cont2ply))/(cont2cont+cont_size)) + 1
ny = int((w - (cont_size+2*cmp2cont))/(cont2cont+cont_size)) + 1
dx = (ld - nx * cont_size - (nx - 1) * cont2cont)*cmp2cont/cont_size
dy = (w - ny * cont_size - (ny - 1) * cont2cont)/2

# adding contact array and metals
# Left contacts
if not (w_changed == True and nf > 1) and (ld >= 440):
    cell.insert(pya.CellInstArray.new(cont_cell_index,
                pya.Trans.new(pya.Point.new(dx, dy)),
                pya.Point.new((cont2cont+cont_size), 0),
                pya.Point.new(0, (cont2cont+cont_size)), nx, ny))
    # Left metal
    cell.shapes(metal1).insert(pya.Box(-metal_violat, -metal_violat,
                               ld + metal_violat - (cont_size-2*cmp2cont),
                               w + metal_violat))
```

Width проверка

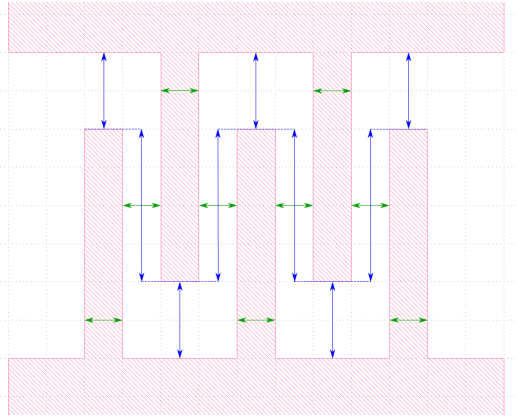


```
layer = input(1, 0)
layer.width(0.25).output(”Width check”)
```

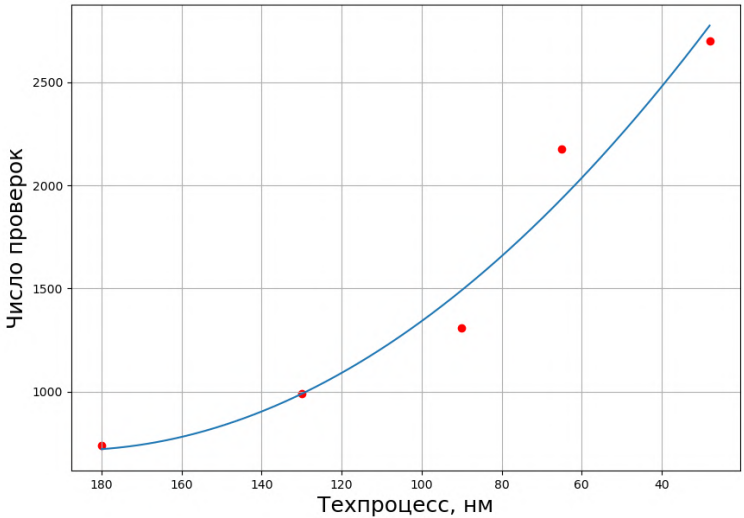


Более сложные проверки

```
tmp1 = L1.width(0.52).polygons
tmp2 = tmp1.uedges.and(
    tmp1.uedges.separation(L1.uedges, 0.52).edges
)
tmp3 = tmp2.uedges.separation(L1.uedges,
                    0.52).polygons.not_interacting(
    (tmp1.uedges.separation(L1.uedges,
        0.52).polygons.uedges & tmp2.uedges.with_length(1.3+EPS,
                                                        nil).uedges)
)
tmp4 = tmp1.outside(tmp3).interacting(
    (tmp1.uedges & tmp3.uedges).extended(-1.dbu, -1.dbu,
                                         1.dbu, 1.dbu)
)
tmp5 = tmp4.or(tmp3.outside(tmp4).interacting(
    (tmp3.uedges & tmp4.uedges).extended(-1.dbu, -1.dbu,
                                         1.dbu, 1.dbu))
)
tmp6 = L1.space(0.42).polygons
...
```



Сложность DRC для ”тонких” процессов

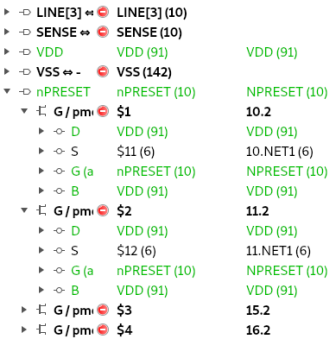


LVS в KLayout

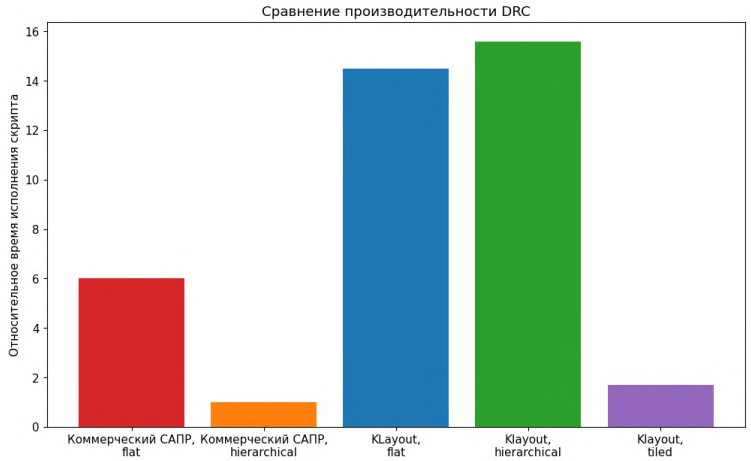
LVS Device Extractors

- Resistor extractors (resistor and resistor_with_bulk)
- Capacitor extractors (capacitor and capacitor_with_bulk)
- Diode extractor (diode)
- MOS transistor extractor (mos3 and mos4)
- Diffusion MOS transistor extractor (dmos3 and dmos4)
- Bipolar transistor extractor (bjt3 and bjt4)
- Device extractors and device classes

```
# 3.3V PMOS transistor outside DNWELL
logger.info("Extracting 3.3V PMOS transistor outside DNWELL")
extract_devices(mos4("pmos_3p3"),
{ "SD" => psd, "G" => pgate_3p3v,
"tS" => psd, "tD" => psd,
"tG" => poly2_con, "W" => nwell_con })
```



Сравнение производительности DRC KLayout с коммерческим инструментом



Выводы

Выводы

- Открытые САПР предоставляют развитый API, позволяющий реализовать DRC и LVS проверки даже для тонких PDK;
- Меньшая производительность открытых САПР компенсируется возможностью модификации их кода, в том числе точечных оптимизаций;
- KLayout обладает режимом исполнения DRC, который незначительно уступает коммерческим САПР;
- Мы уже применяем KLayout в разработке (в качестве GDS редактора), а OpenLane используем для прототипирования;
- Адаптация DRC и LVS скриптов для KLayout под требуемый техпроцесс позволит получить для него маршрут разработки цифровых микросхем полностью независимый от коммерческих САПР.

Спасибо за внимание!