

アナログ開発ツールの使い方 (OpenRule1um)

森 瑞紀 (Mizuki Mori) , <https://github.com/3zki>

今村 謙之 (Noritsuna Imamura)

Rev.1

アジェンダ

- 開発ツールの紹介、PDKの中身
- アナログLSIの設計フロー
- アナログLSI設計デモンストレーション (CMOSインバータ)

アナログLSIの設計フロー

どのような設計をするか、どのような開発ツールを使用するか

アナログLSIの設計フロー

1. 回路図（テストベンチ）を描く
2. シミュレーションをする
3. 回路図を基にレイアウトを描く
4. レイアウトを検証する (DRC, LVS)
5. レイアウトを基に寄生成分を考慮したシミュレーションをする OpenRule1um では未対応
6. フレームに載せる

そもそもLSI回路設計に必要なものとは？

- **プロセスデザインキット PDK**
 - シミュレーションモデルライブラリ (SPICE)
 - 検証ツール用ルールファイル (DRC, LVSなど)
 - スタンダードセルライブラリ
 - レイアウト (GDS)
 - ネットリスト(SPICE)
 - 自動配置配線ルール(LEF)
 - Verilogシミュレーションライブラリ (LIB, V)
 - 一部指定されたレイアウト (フレームなど)
- PDKで指定された各種ツール (EDAツール)
 - 回路図エディタ or Verilogコンパイラ
 - 回路図エディタ : xschem, Glade, LTspice, KiCAD
 - レイアウトエディタ or 自動レイアウトツール
 - レイアウトエディタ : klayout, Glade
 - SPICEシミュレータ or Verilogシミュレータ
 - SPICEシミュレータ : ngspice
 - 検証ツール
 - 検証ツール : klayout

PDKの場所

公式レポジトリ

- Glade(回路図&レイアウト), KiCAD(回路図)用PDK
<https://github.com/MakeLSI/OpenRule1um>
- Klayout用PDK (レイアウトエディタ)
<https://github.com/mineda-support/OpenRule1um>
- Qflow用PDK (デジタル自動配置配線)
<https://github.com/mineda-support/OR1>
- Glade, KiCAD, Klayout用スタンダードセル
https://github.com/MakeLSI/OpenRule1um_StdCell
- LTspice用PDK (回路図 ,スタセルシンボル)
https://github.com/MakeLSI/OpenRule1um_StdCell_LTspiceSymbol
- OR1実測データ (Spiceモデルあり)
<https://github.com/MakeLSI/Measure>

ISHI-KAIオリジナル

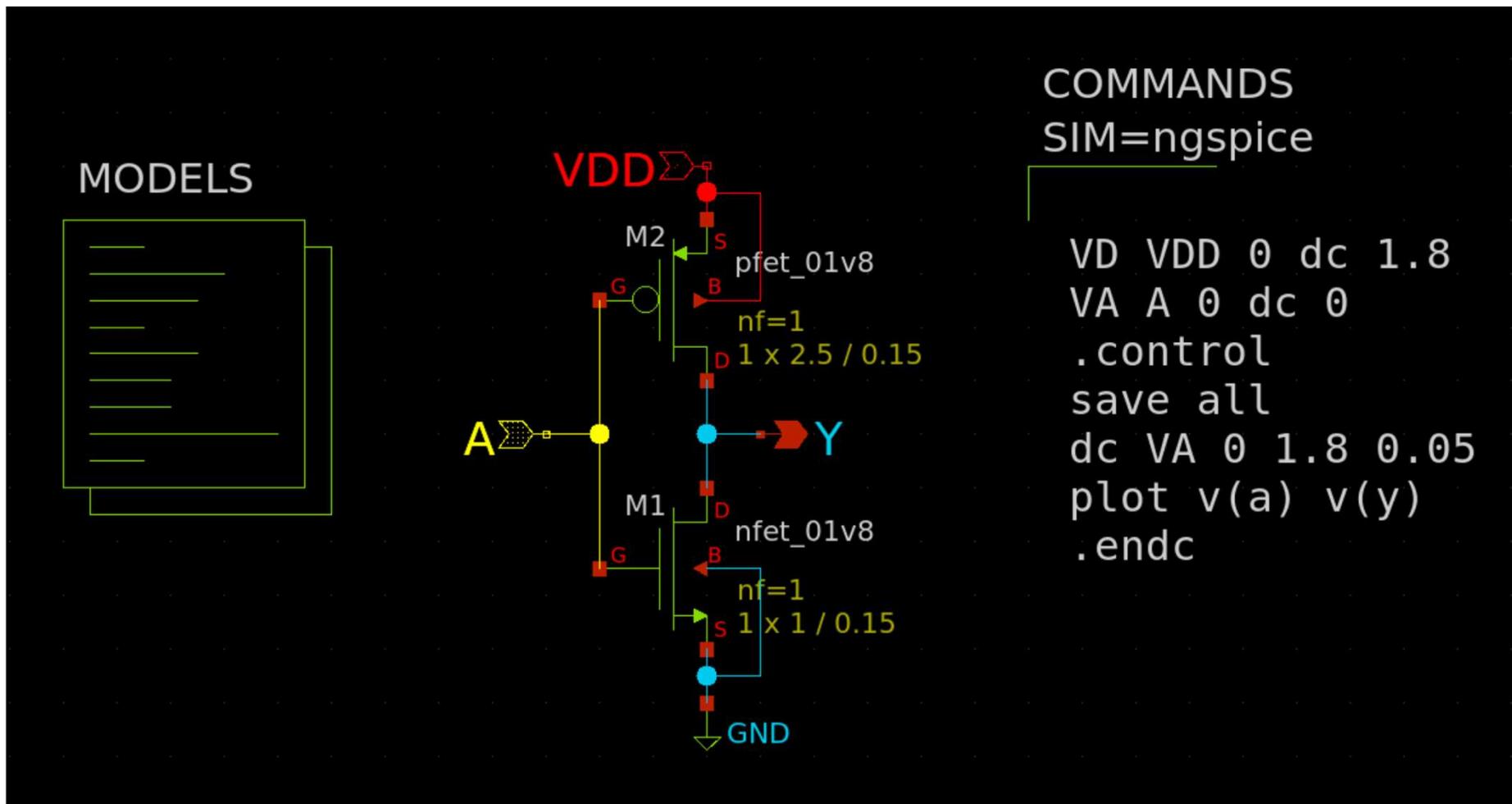
- Xschem, Klayout用PDK **[NEW!]**
https://github.com/ishi-kai/OpenRule1umPDK_SetupEDA
- Xschem用PDK (開発ベータ版)
https://github.com/3zki/OpenRule1umPDK_Xschem

ローカルでのセットアップ方法

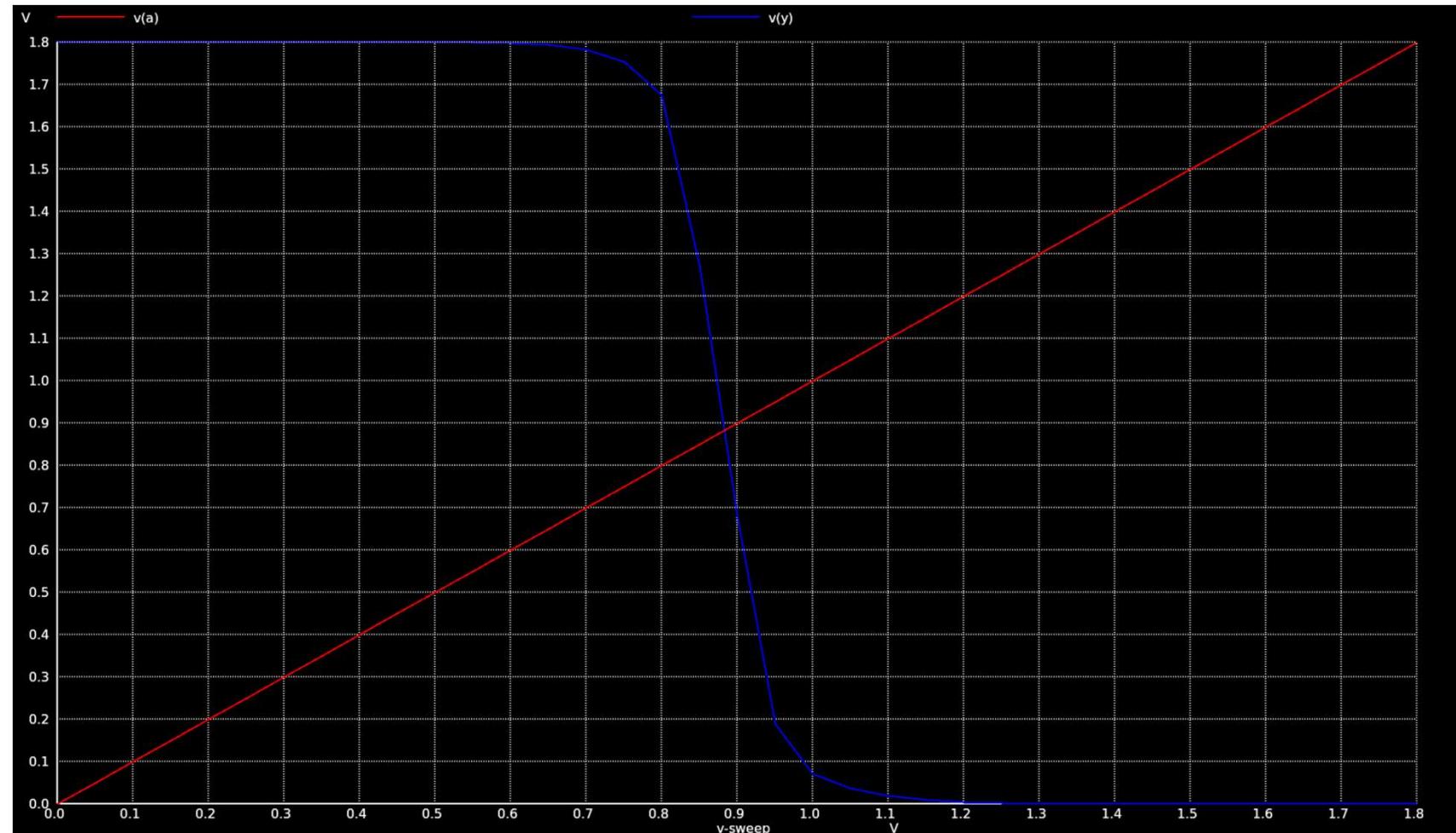
https://github.com/ishi-kai/OpenRule1umPDK_setupEDA

- ここに必要なものが全てが入っています。

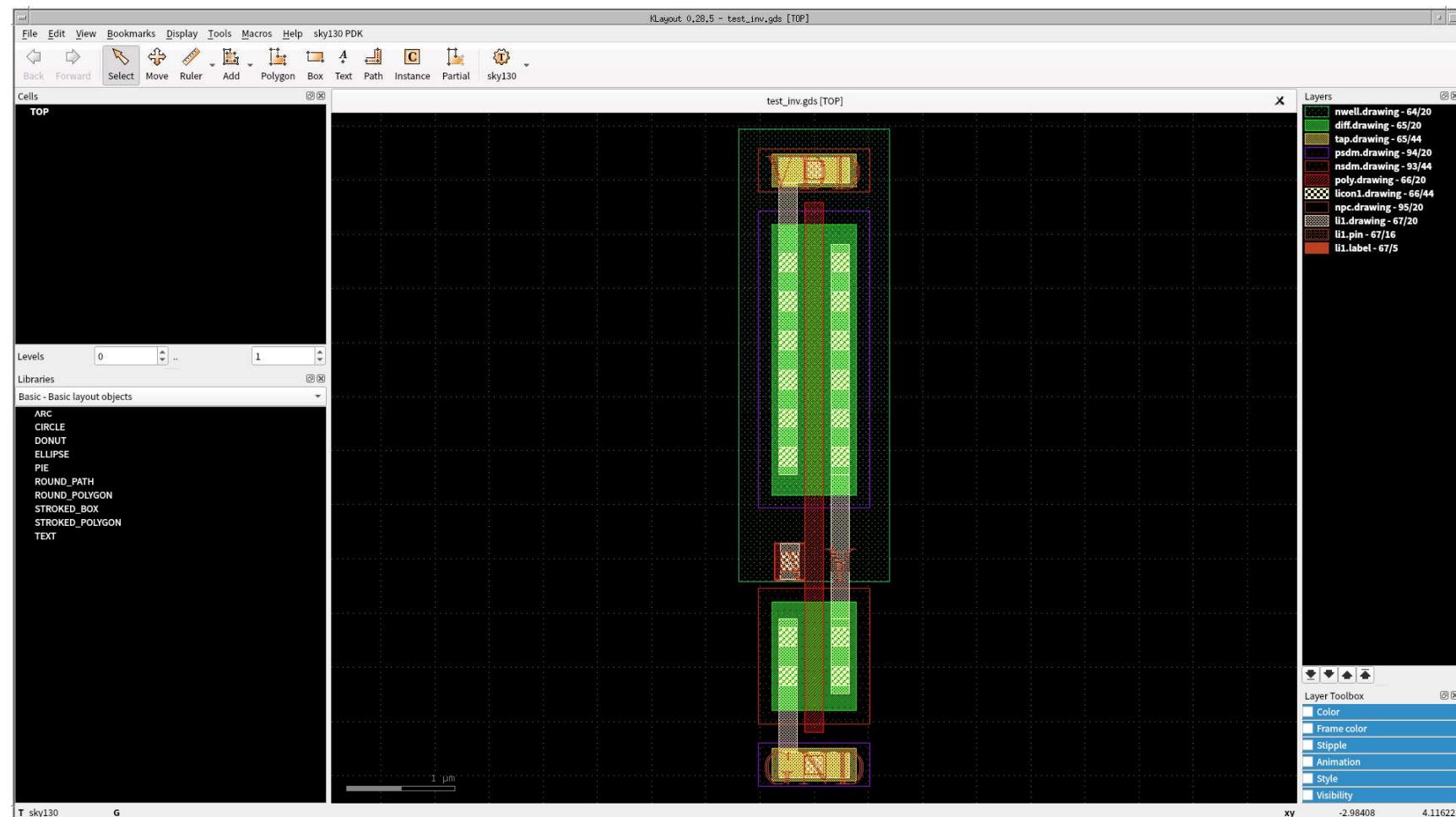
回路図(ベンチマーク)を描いて…



論理検証をして…

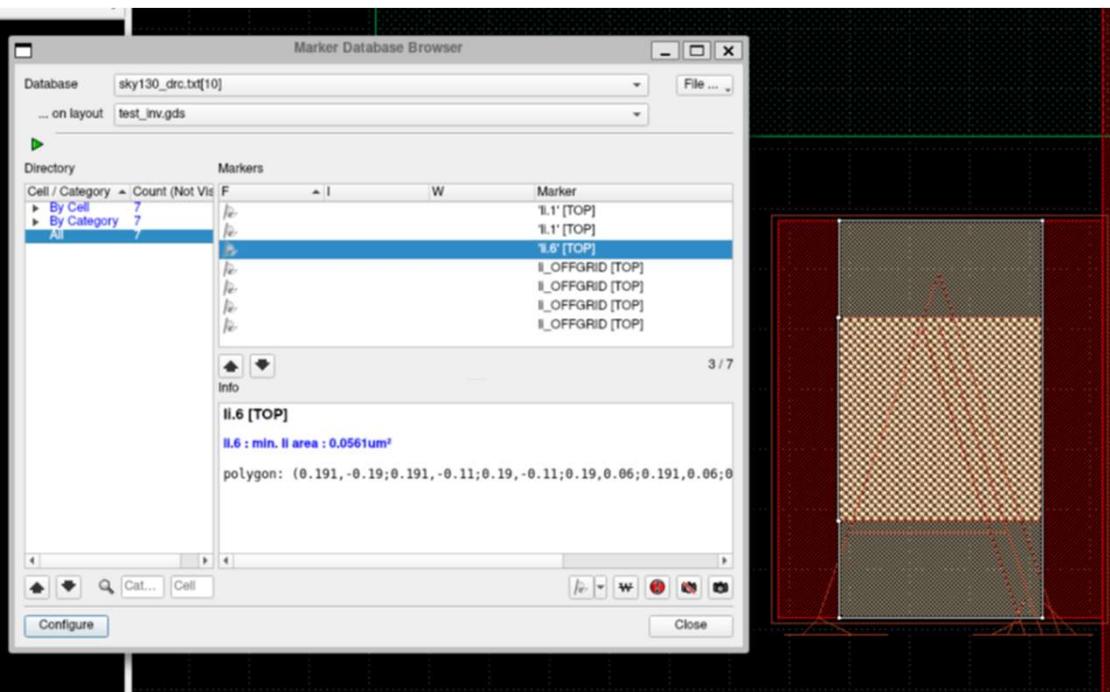


レイアウトを描いて…

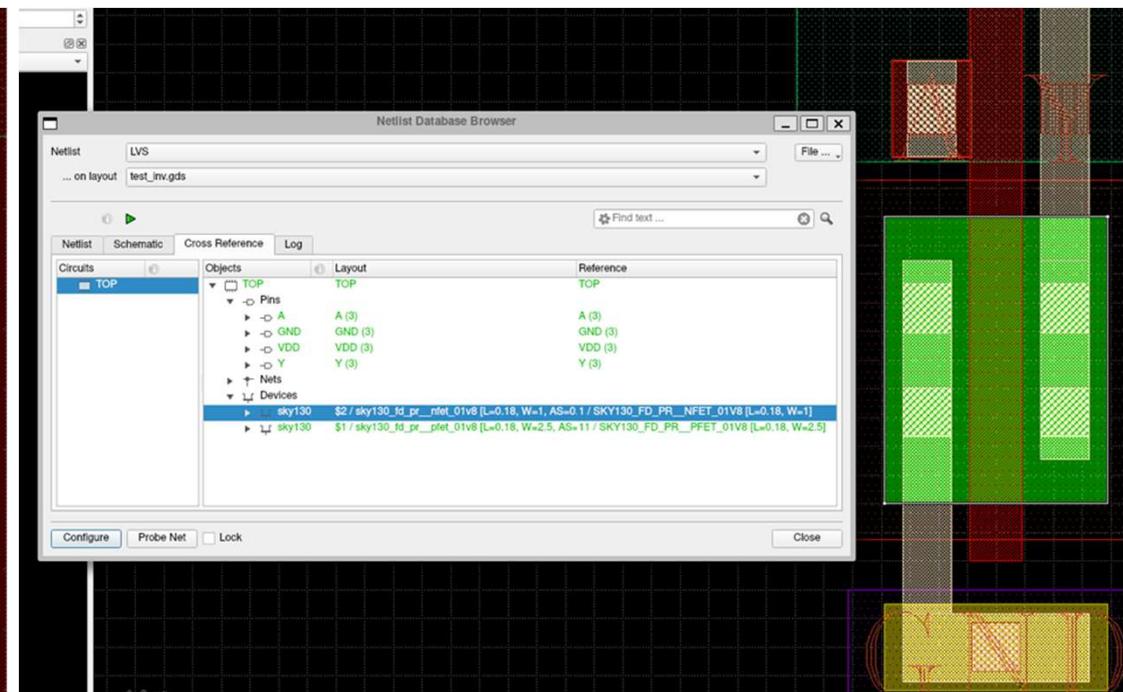


レイアウトを検証して…

DRC



LVS



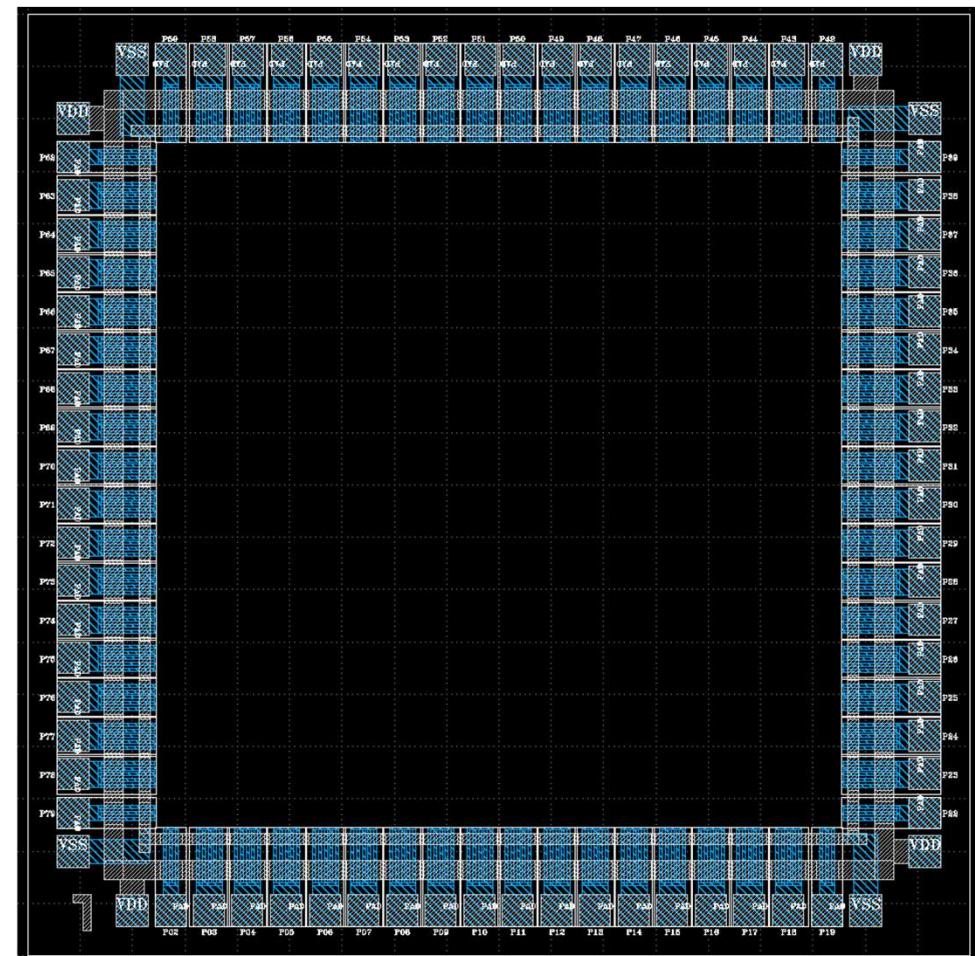
完成したら指定のフレームに回路を載せる（テープアウト対象者のみ）

指定のフレームにレイアウトを載せる

- 80ピン(ユーザが使用可能なピンは72ピン)
 - 自分が使用するピンについてはピンリストを参照



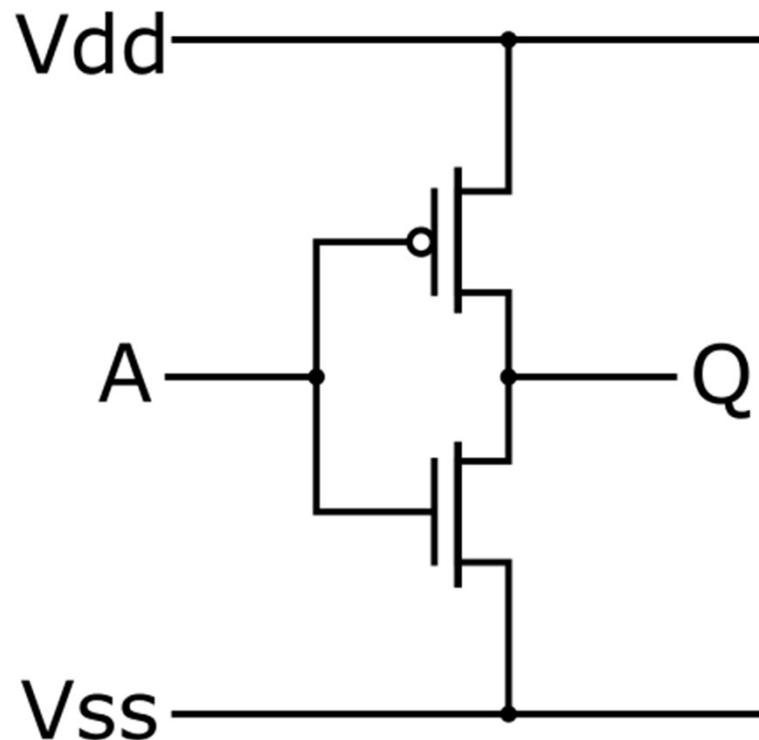
チップサイズ1.8×1.8(mm)
LQFP 80pin 対応モデル
80PAD(座標固定)



アナログLSI設計デモンストレーション

CMOSインバータ作成

CMOSインバータ



入力A	出力Q
L	H
H	L



入力A	出力Q
V_{ss}	V_{dd}
V_{dd}	V_{ss}

アナログLSIの設計フロー

1. 回路図を描く
2. シミュレーションをする
3. 回路図を基にレイアウトを描く
4. レイアウトを検証する
5. レイアウトを基に寄生成分を考慮したシミュレーションをする
6. (フレームに載せる)

寄生成分データが
無いため割愛

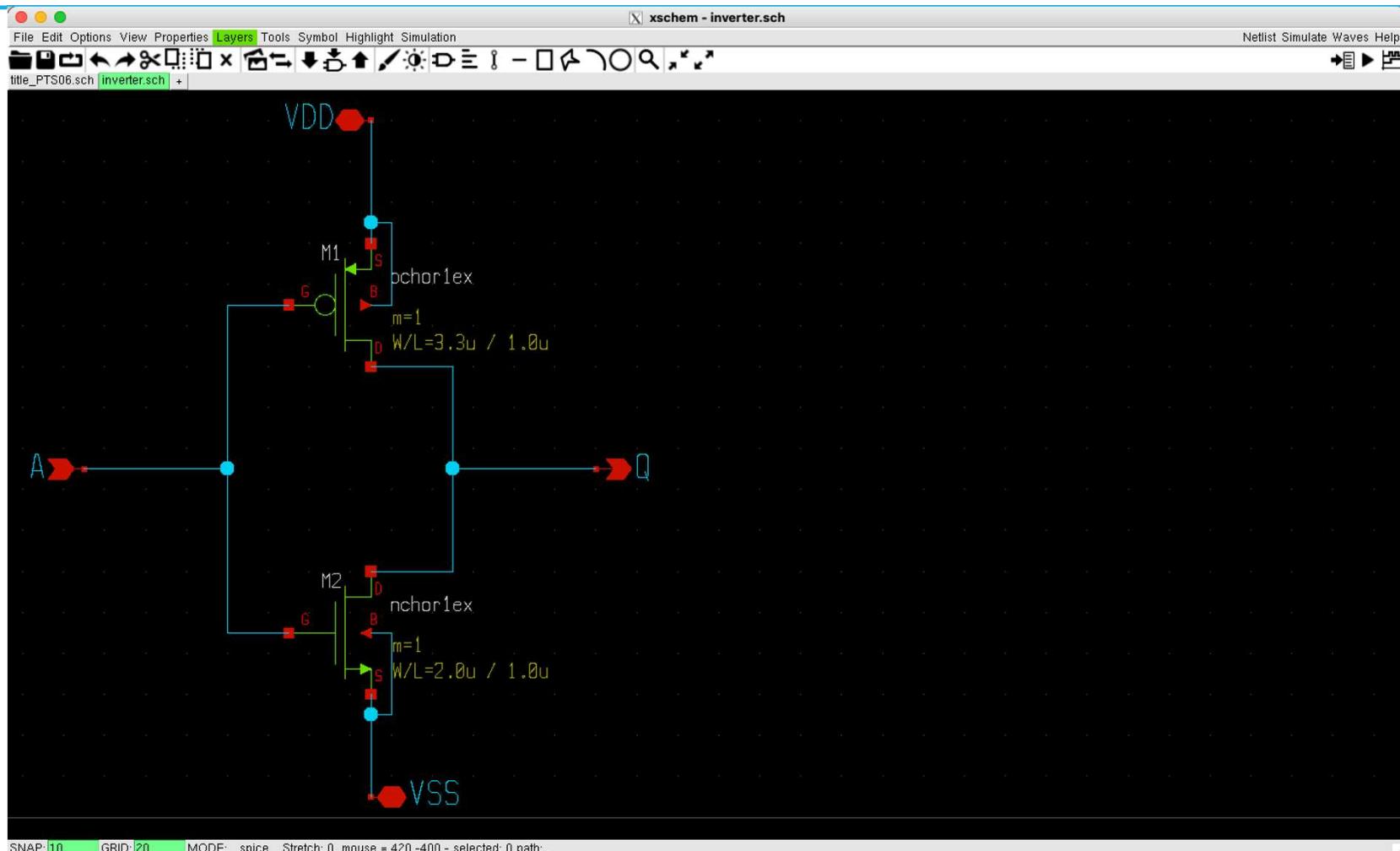
xschem使用上の留意点

- デバイスのシンボルは一部のみ
 - モデル名を変更する必要がある場合もある（P-FET,N-FETでは不要）

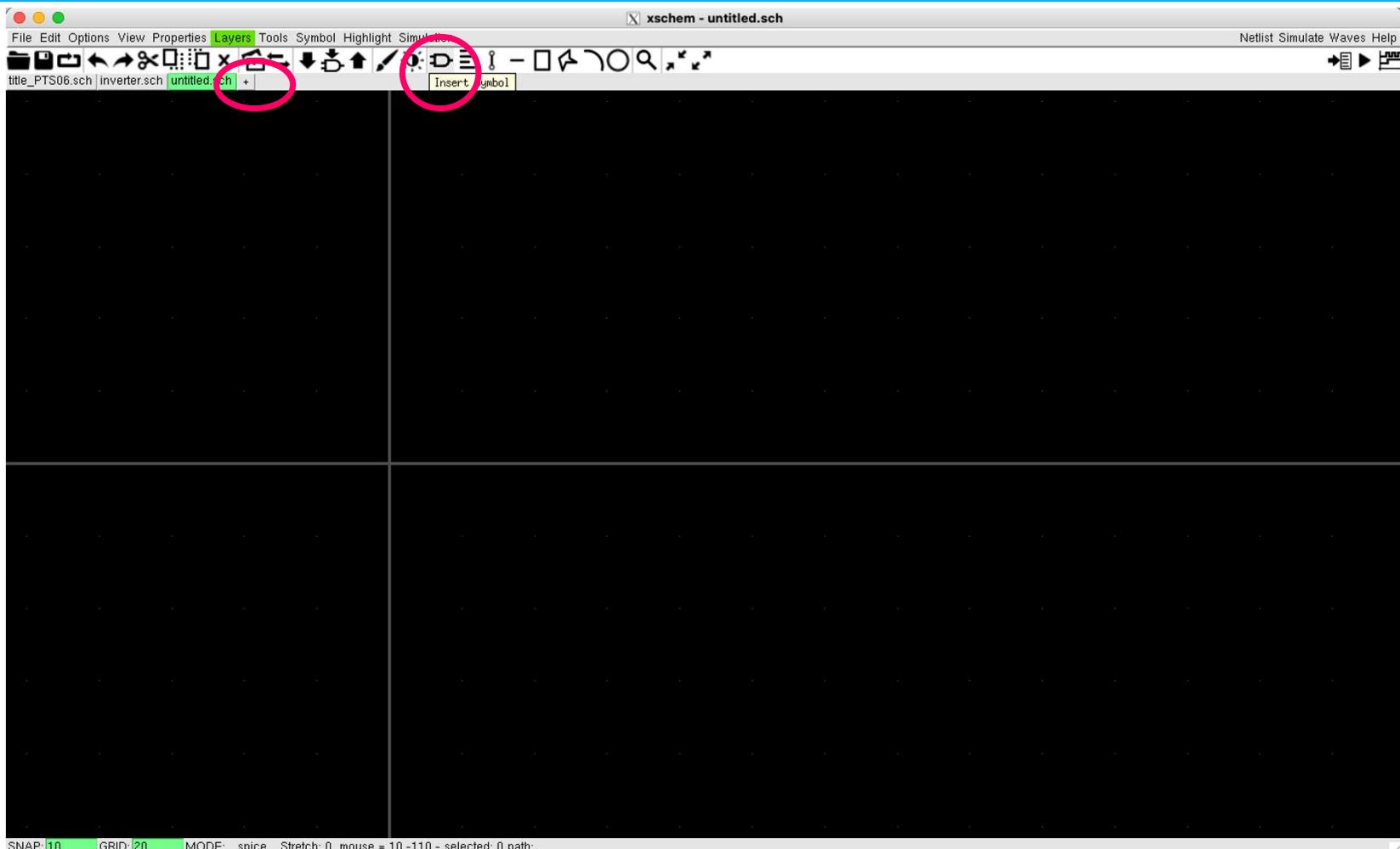
論理検証

- インバータでは入力に対して反転した出力が確認できれば良い
 - 入力 0 V → 出力 Vdd V, 入力 Vdd V → 出力 0 V
 - DC解析でCMOSインバータの動作を見てみる
- DC解析 電圧を変動させたときの定常応答
- tran解析 時間を変動させたときの過渡応答
- AC解析 周波数を変動させたときの定常応答

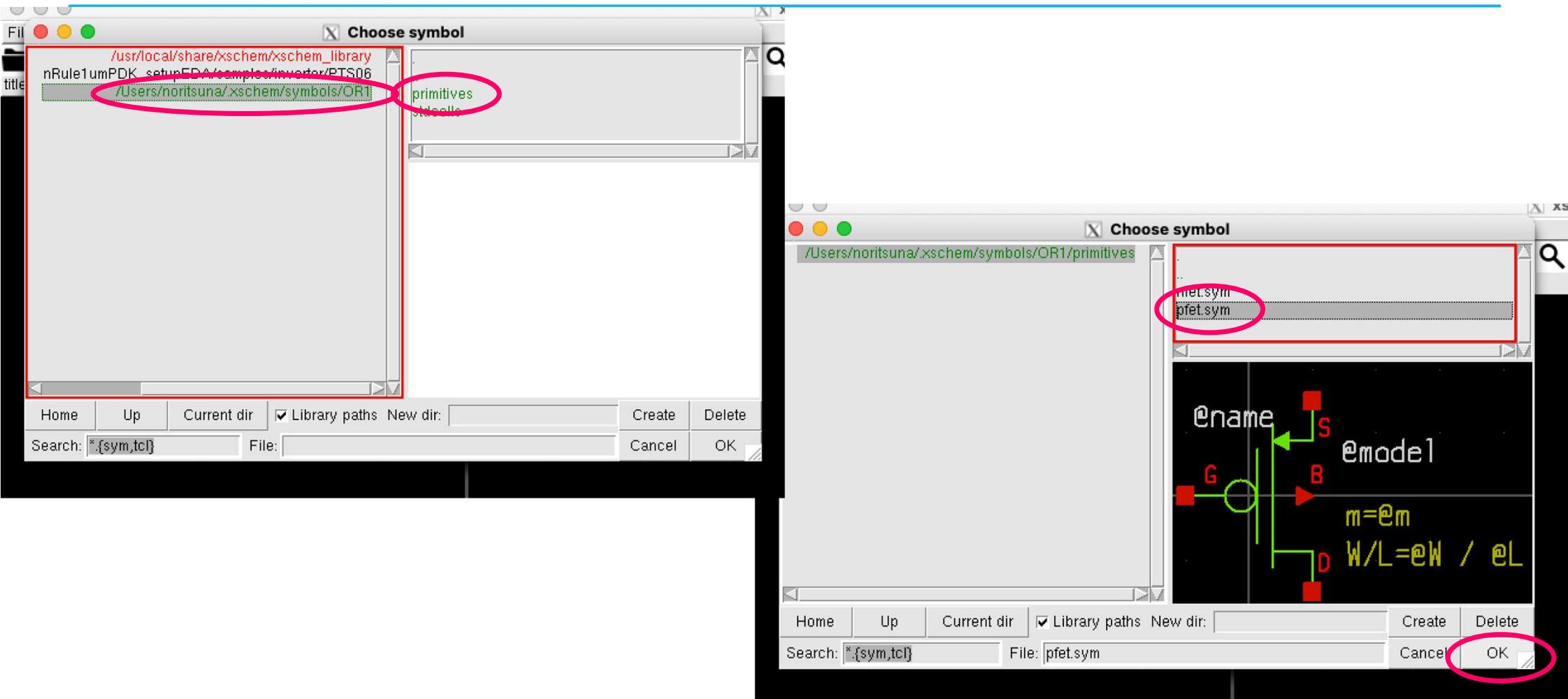
CMOSインバーター回路の例



xschem使い方：新規回路図とMODULEウィンドウ



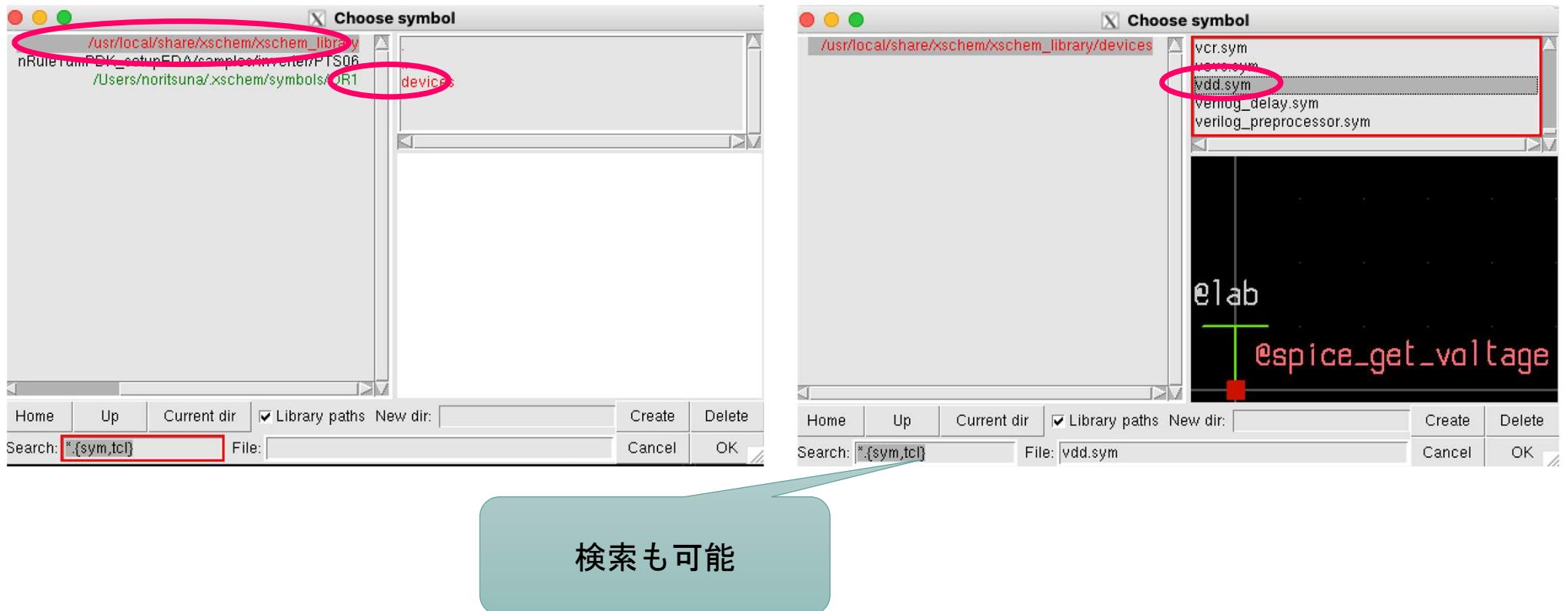
xschem使い方：OR1のMODULEの選択：P-FET



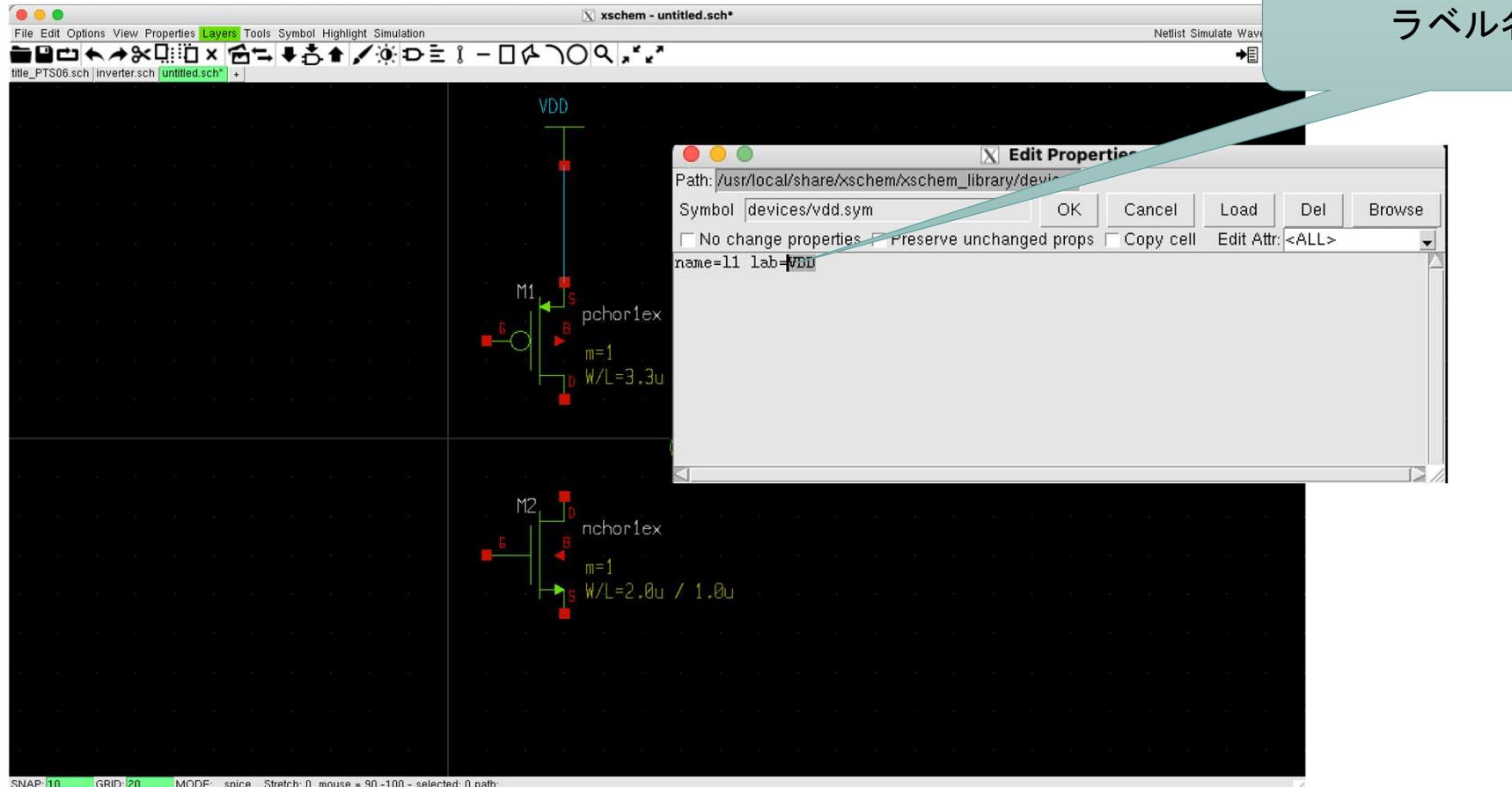
xschem使い方：OR1のMODULEの配置とプロパティ



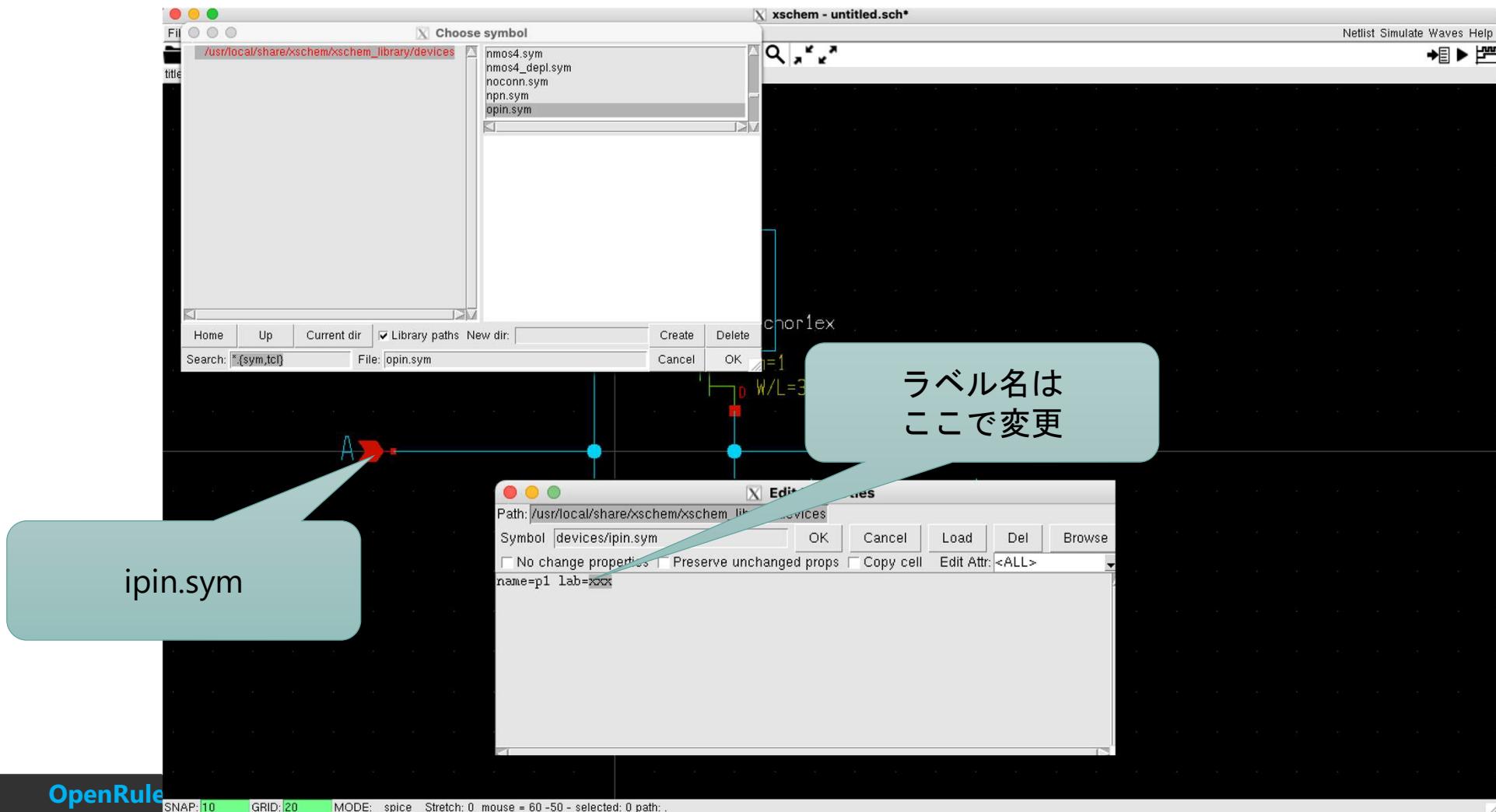
xschem使い方：標準のMODULEの選択：VDD



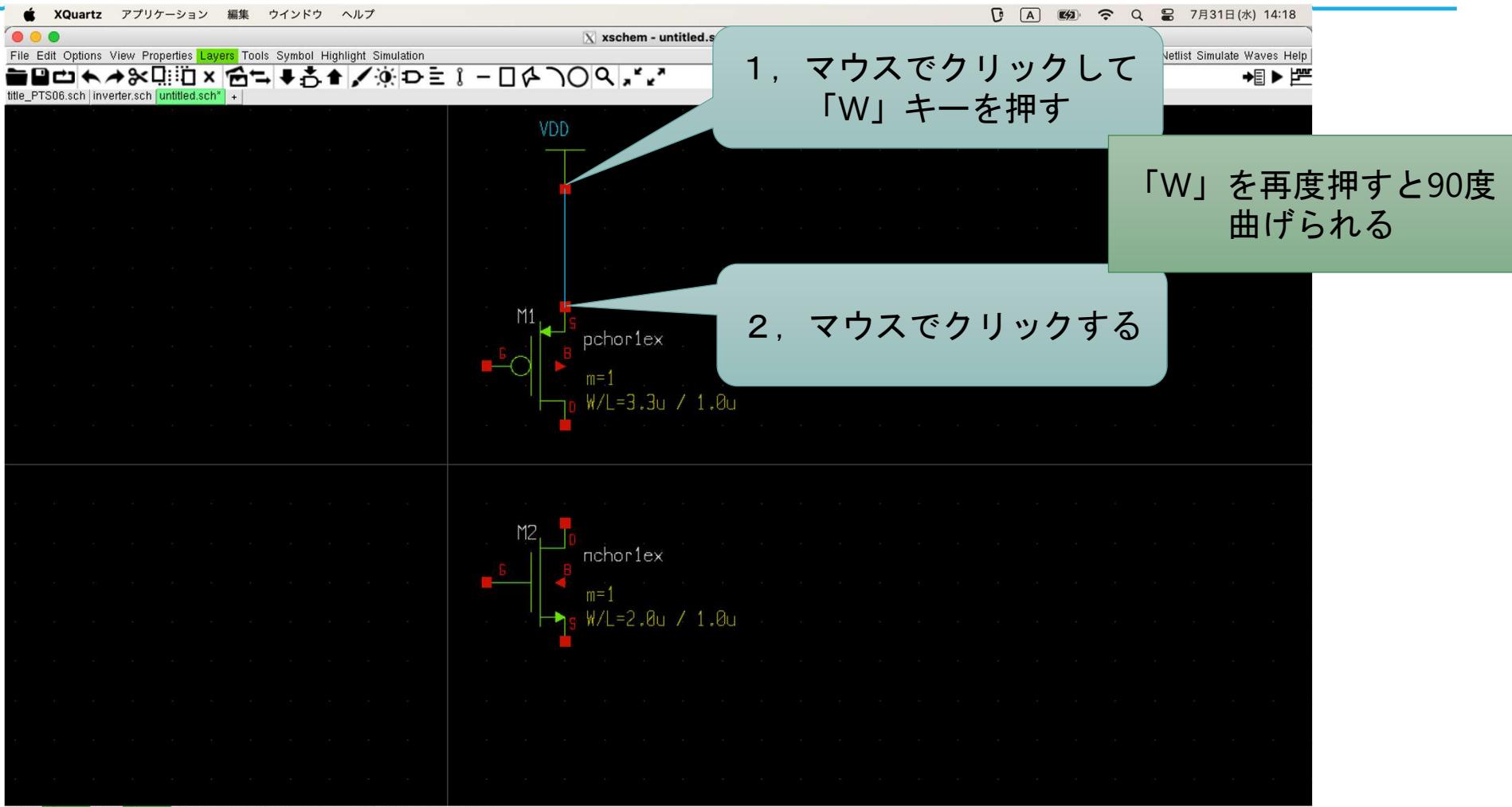
xschem使い方：標準のMODULEの配置とプロパティ：VDD



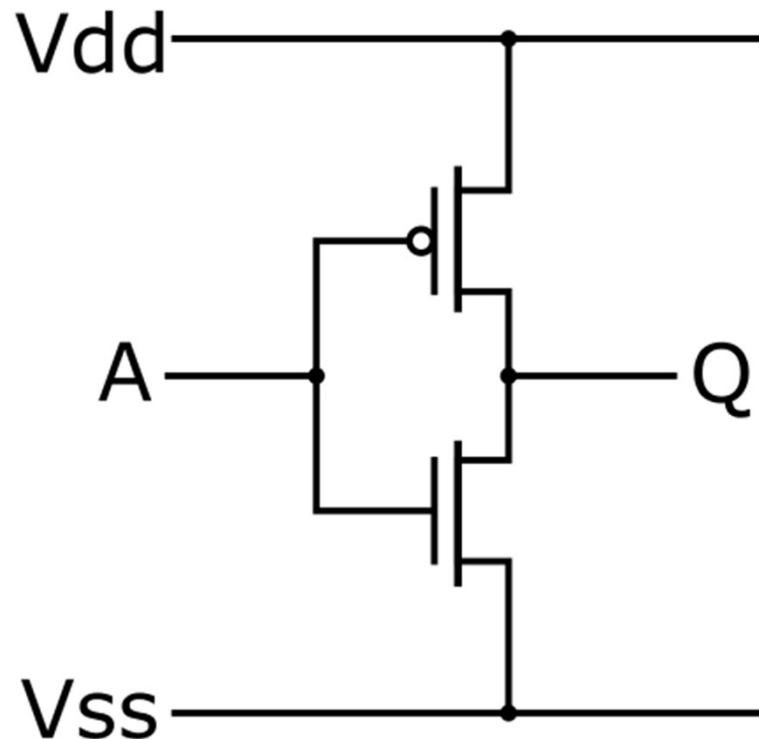
xschem使い方：標準のMODULEの選択：ipin, opin, iopin



xschem使い方： wireの引き方



ハンズオン：CMOSインバータを作る

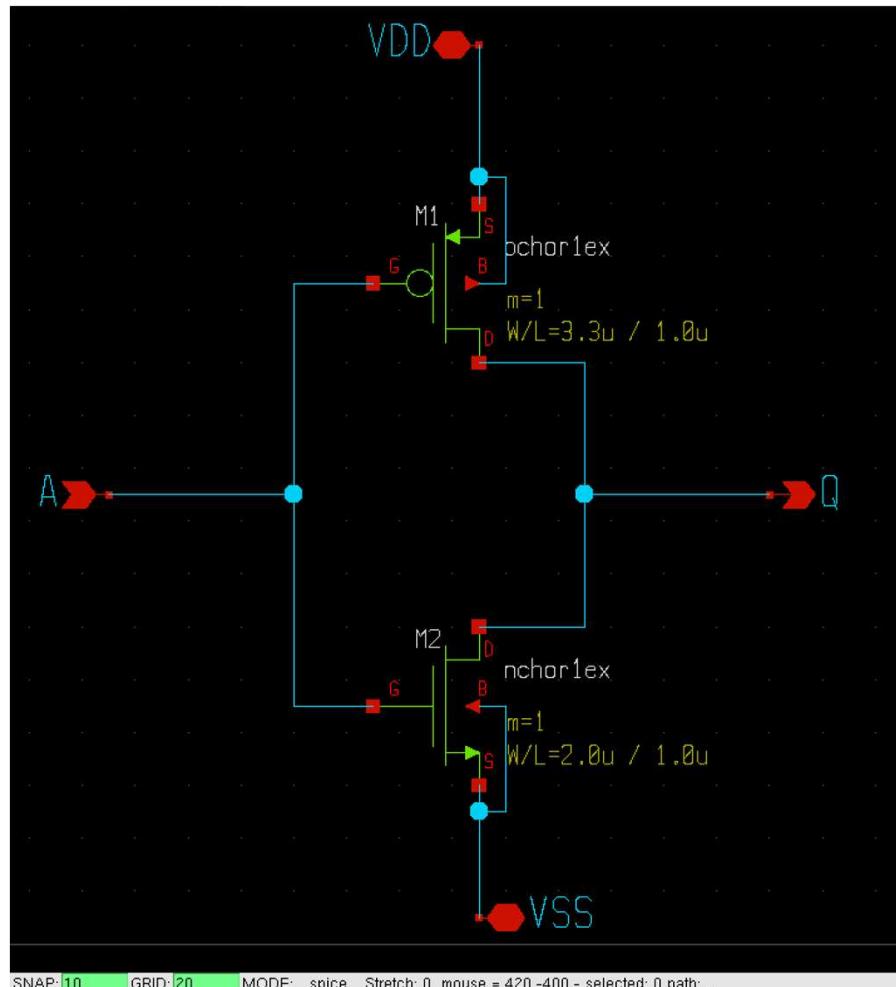


入力A	出力Q
L	H
H	L



入力A	出力Q
V_{ss}	V_{dd}
V_{dd}	V_{ss}

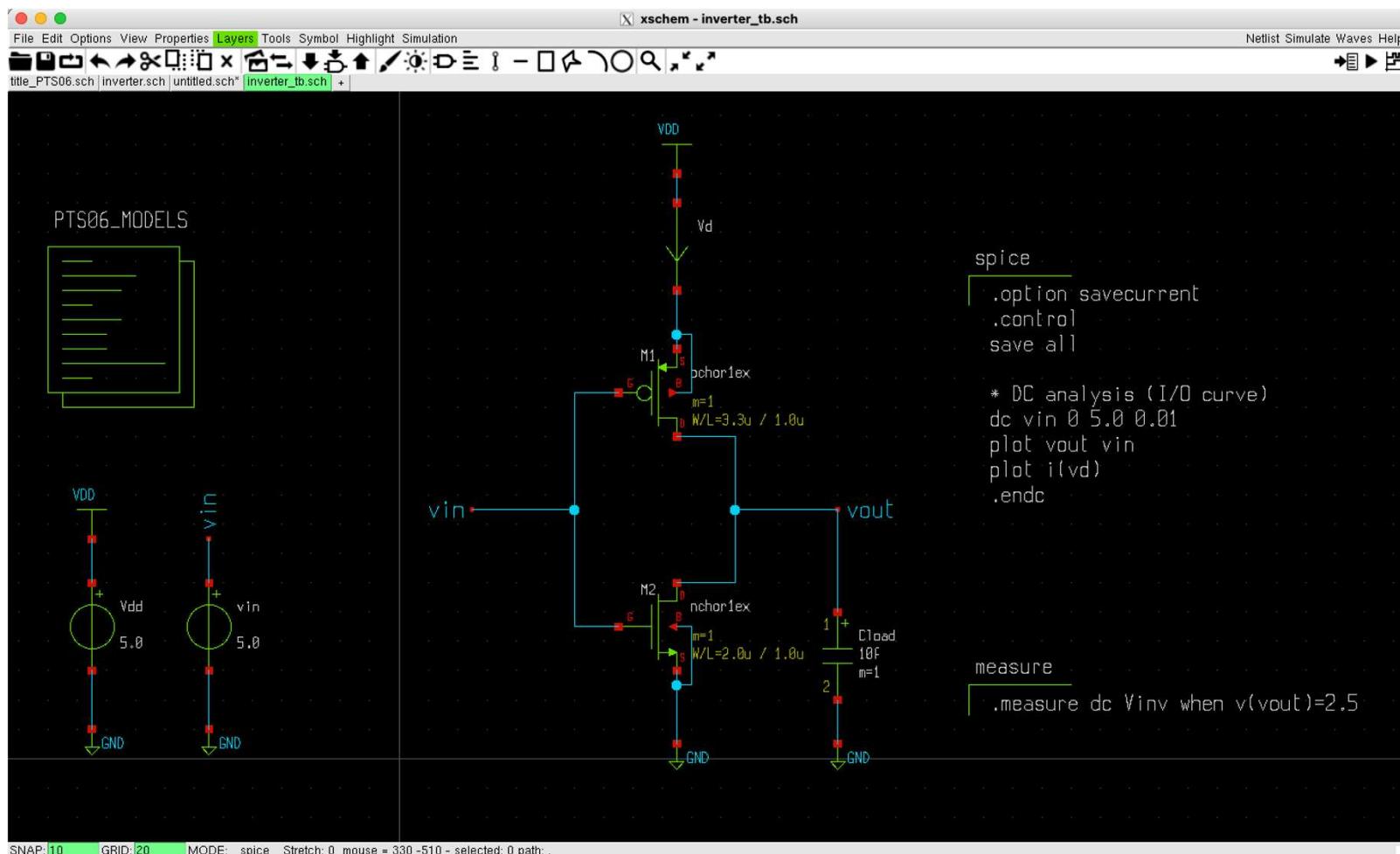
ハンズオン：CMOSインバータのサンプル



アナログLSIの設計フロー

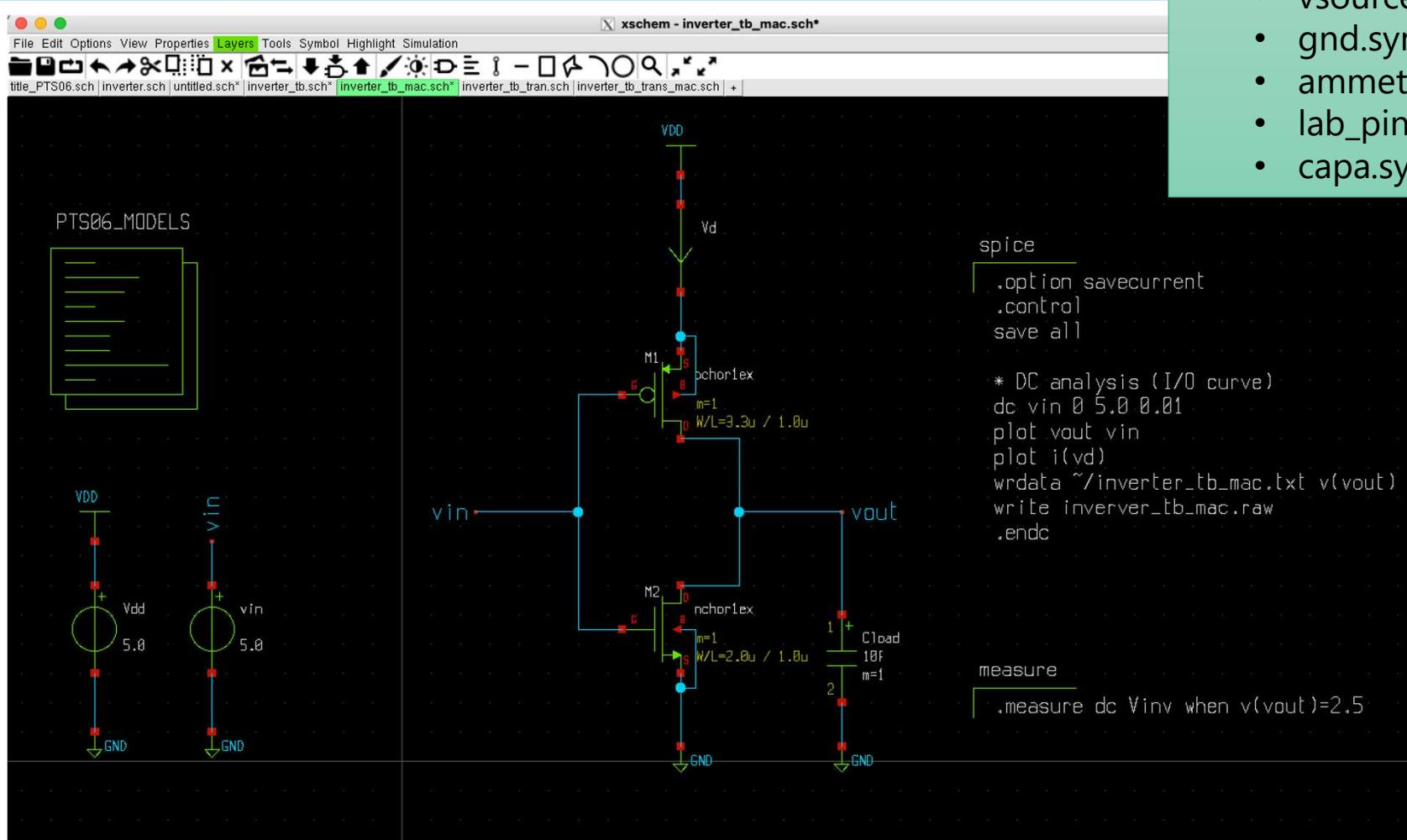
1. 回路図を描く
2. シミュレーションをする
3. 回路図を基にレイアウトを描く
4. レイアウトを検証する
5. レイアウトを基に寄生成分を考慮したシミュレーションをする
6. (フレームに載せる)

xschem : ベンチマーク例



SNAP: 10 GRID: 20 MODE: spice Stretch: 0 mouse = 330 -510 - selected: 0 path: .

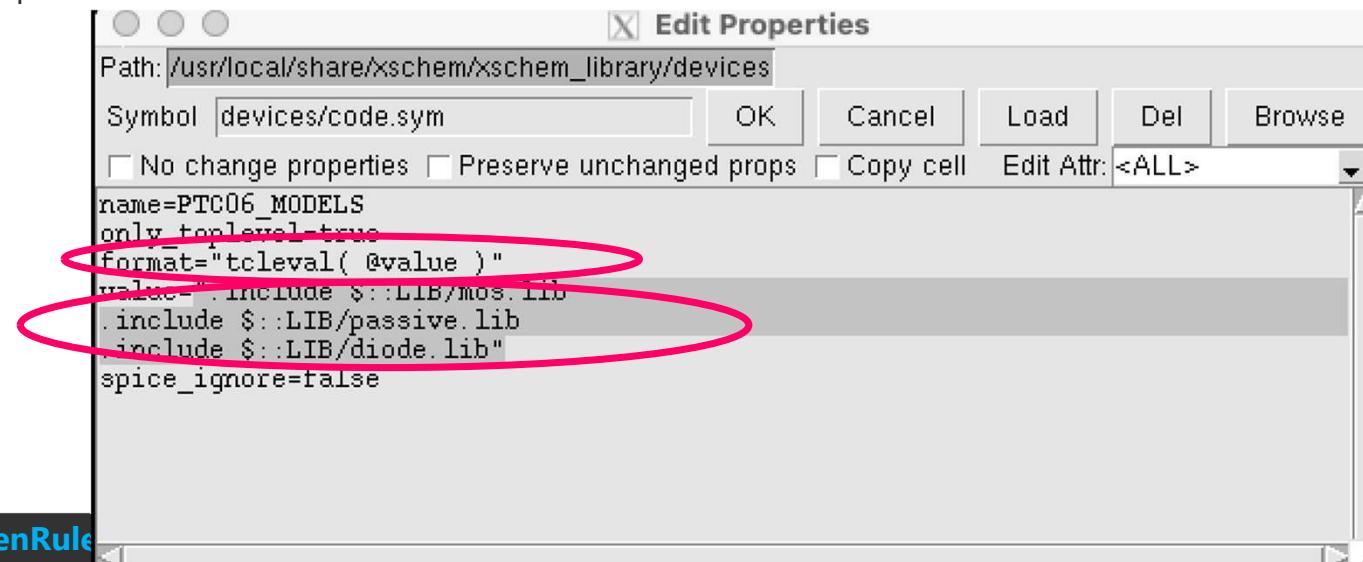
xschem : ベンチマーク



- キーワード
 - code.sym
 - code_shown.sym
 - vsource.sym
 - gnd.sym
 - ammeter.sym
 - lab_pin.sym
 - capa.sym

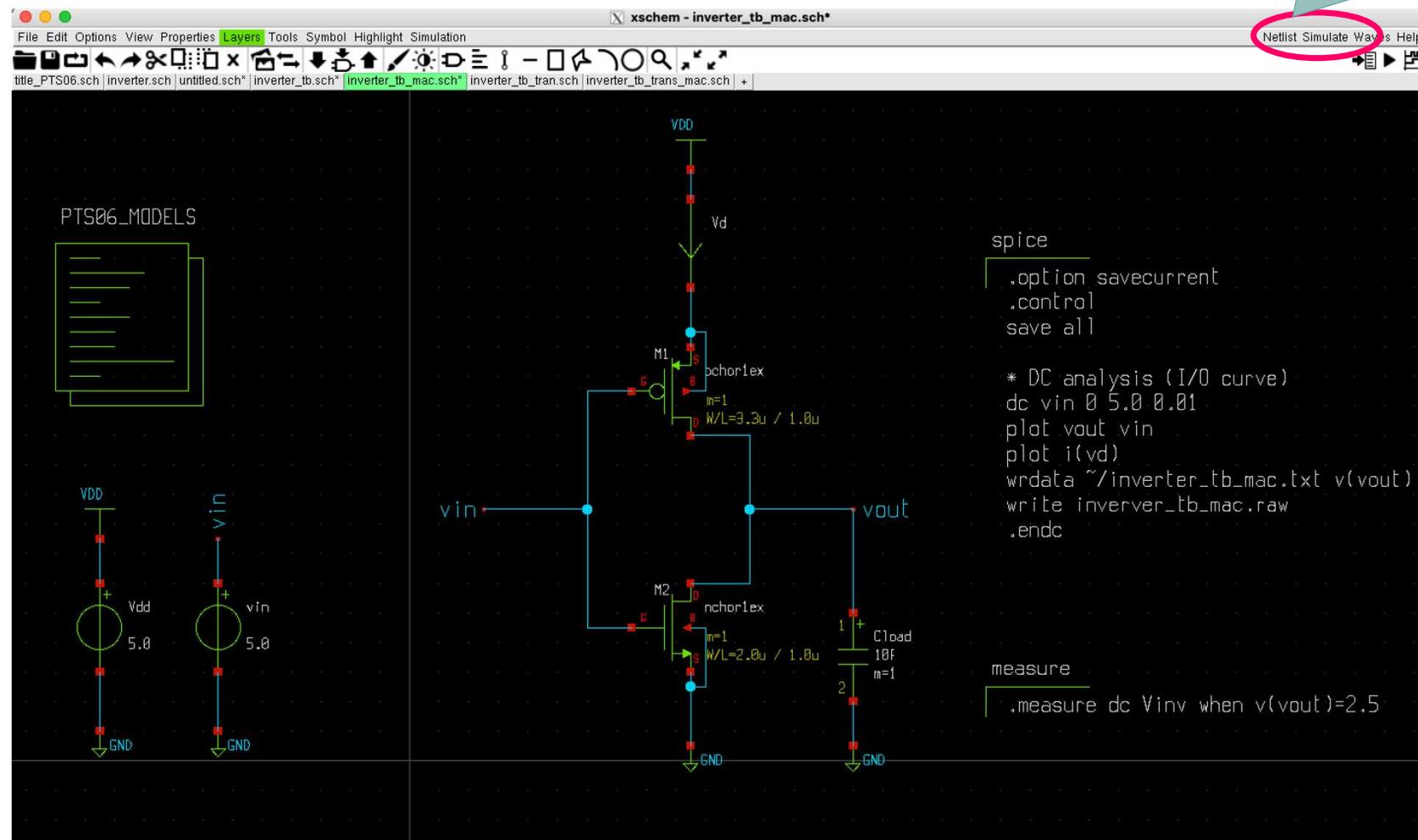
xschem : PTS06_MODELSを記述する際の注意点x

- code.symを使用する際、以下の行を追記する必要がある
 - format="tcleval(@value)"
- includeは絶対パス（フルパス）である必要がある。相対パスは使えない。
 - 「\$::LIB」は特殊パスのため利用可能

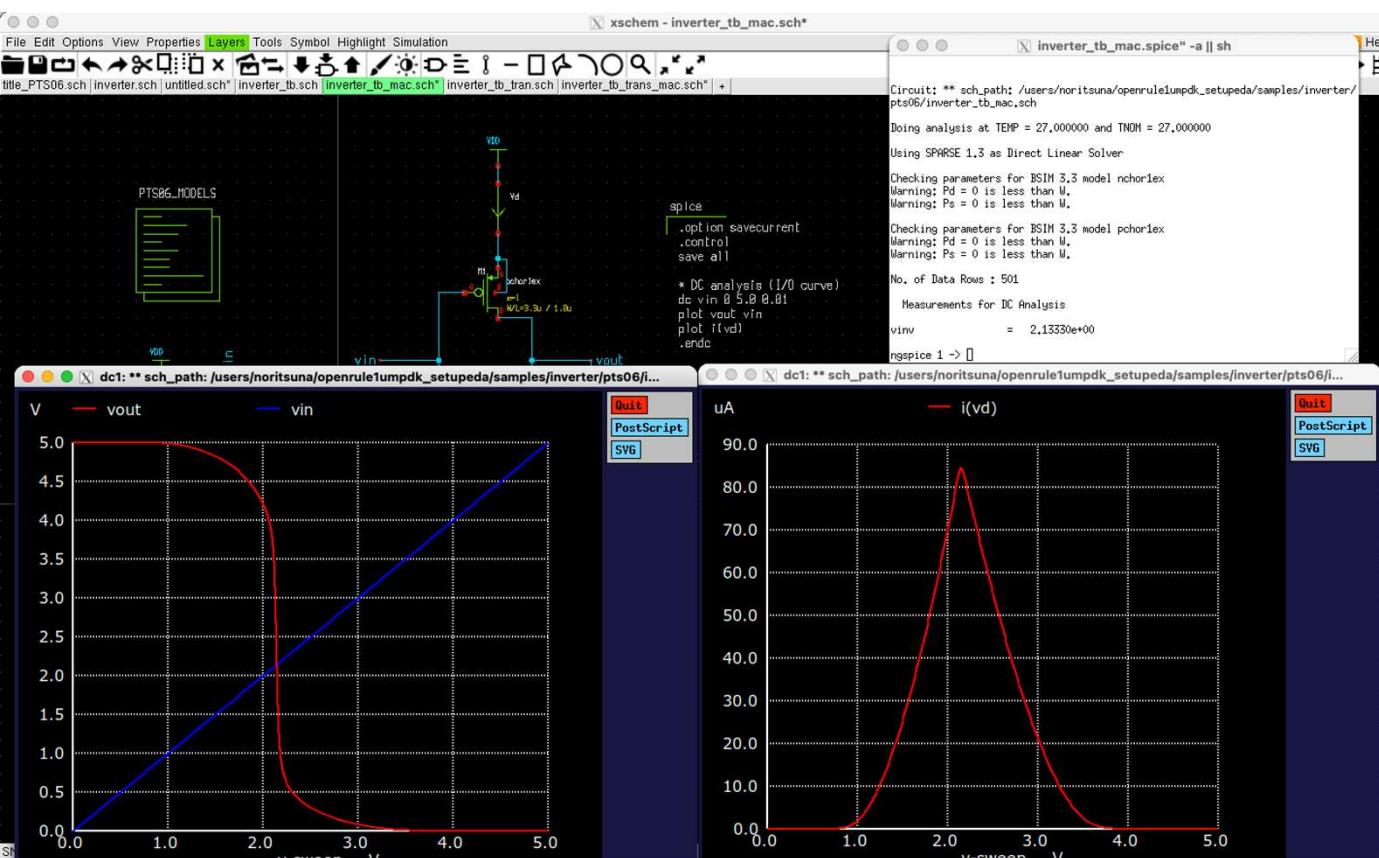


ngspice : シミュレーションの実行

netlist→simulateの
順にクリック



ngspice : シミュレーション結果



入力A = 0 V → 出力Q = 5.0 V

入力A = 5.0 V → 出力Q = 0 V

- インバータとして機能している

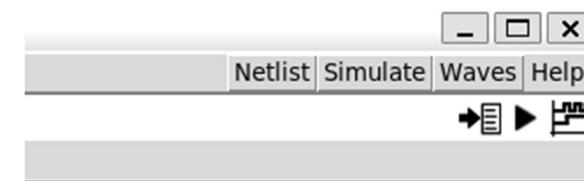
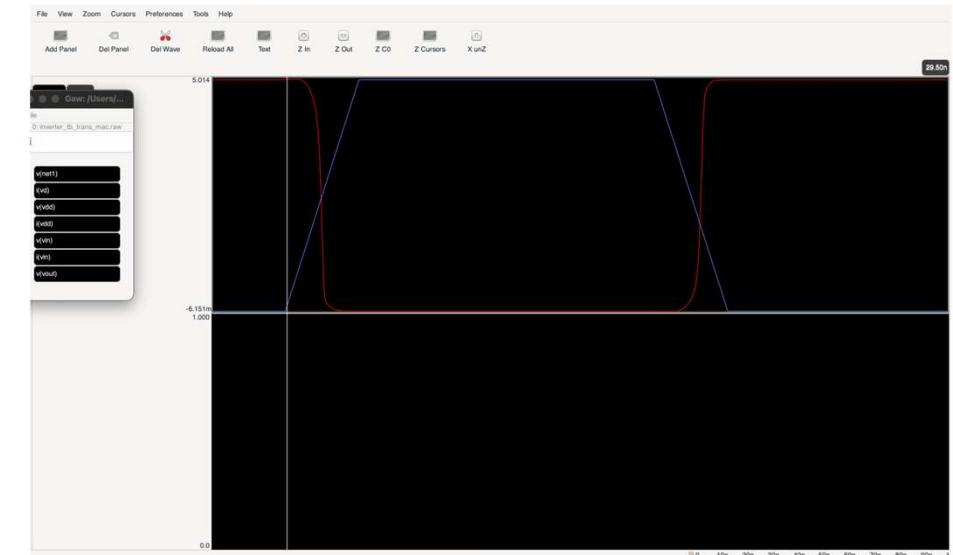
ngspice : wrdata と gawによる波形表示

wrdataを使用するとテキスト出力ができる

```
0.0000000e+00 3.2999999e+00
5.0000000e-02 3.2999999e+00
1.0000000e-01 3.2999997e+00
1.5000000e-01 3.2999990e+00
2.0000000e-01 3.29999960e+00
2.5000000e-01 3.29999842e+00
3.0000000e-01 3.29999375e+00
3.5000000e-01 3.29997564e+00
4.0000000e-01 3.29990878e+00
4.5000000e-01 3.29968137e+00
5.0000000e-01 3.29900073e+00
5.5000000e-01 3.29728006e+00
```

▪
▪
▪

writeでrawファイルを出力すると gaw で波形表示ができる



xschem 右上の
Wavesから起動できる

ngspice : 過渡応答、遅延時間

- 過渡応答から遅延時間を見たい場合はtran解析を用いる
- 出力段に負荷を設定しないと正しい過渡応答が見れない
 今回は入力段・出力段共に何も接続しない状態を見る

ngspice : 過渡解析ベンチマーク例

xschem - inverter_tb_trans_mac.sch

File Edit Options View Properties Layers Tools Symbol Highlight Simulation

Netlist Simulate Waves Help

title_PTS06.sch | inverter.sch | untitled.sch* | inverter_tb.sch | inverter_tb_mac.sch | inverter_tb_tran.sch | inverter_tb_trans_mac.sch | +

PTSO6_MODELS

spice

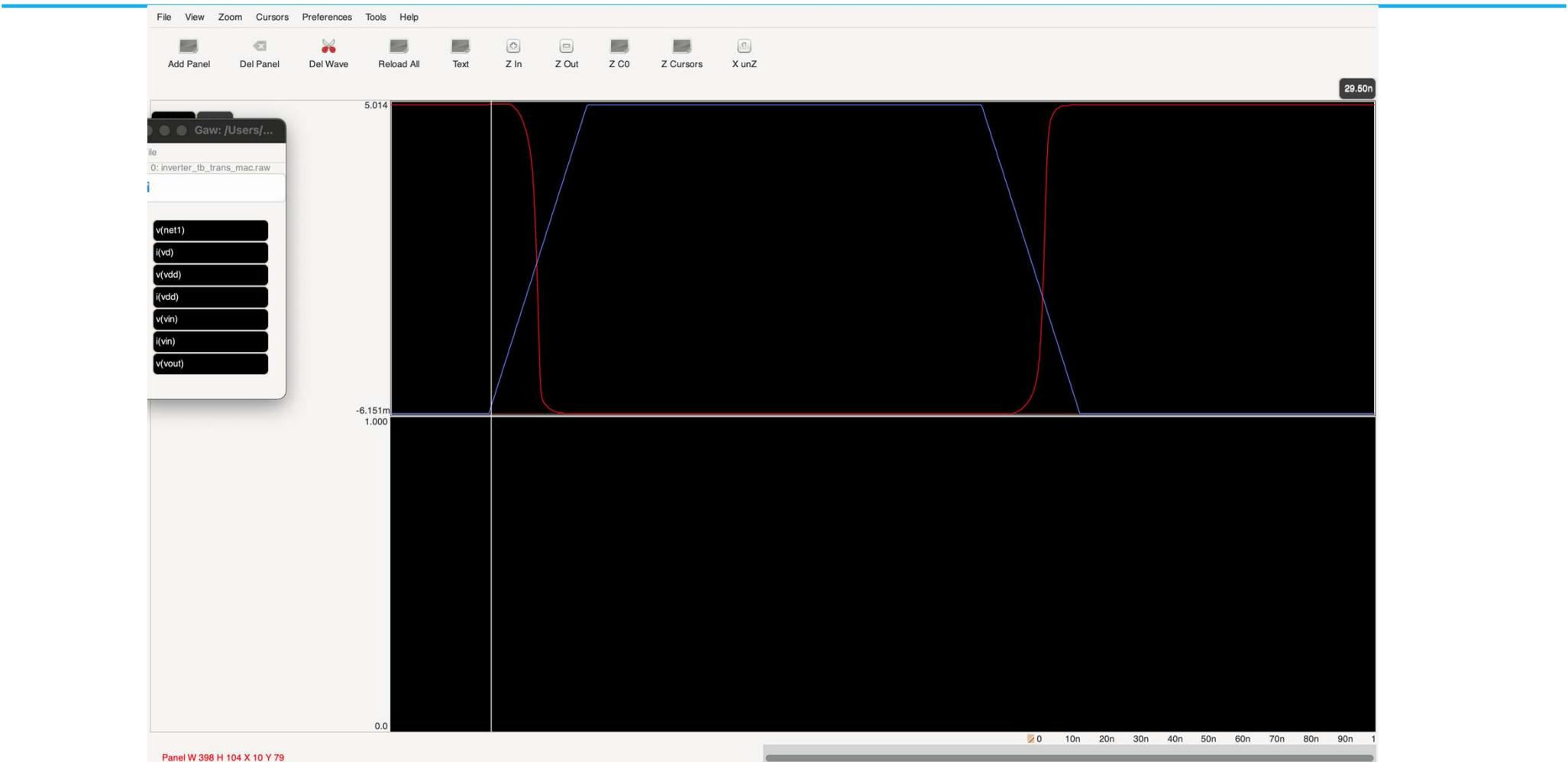
```
.option savecurrent  
.control  
save all  
  
* Tran analysis  
tran 0.1n 100n  
plot vout vin  
plot i(vd)  
.endc
```

measure

```
.measure tran td_r trig v(vin) val=2.5 Fall=1 targ v(vout) val=2.5 rise=1  
.measure tran td_F trig v(vin) val=2.5 rise=1 targ v(vout) val=2.5 fall=1  
.measure tran trise trig v(vout) val=0.83 rise=1 targ v(vout) val=4.17 rise=1  
.measure tran tfall trig v(vout) val=4.17 fall=1 targ v(vout) val=0.83 fall=1
```

Op SNAP: 10 GRID: 20 MODE: spice Stretch: 0 mouse = 550 -480 - selected: 0 path: .

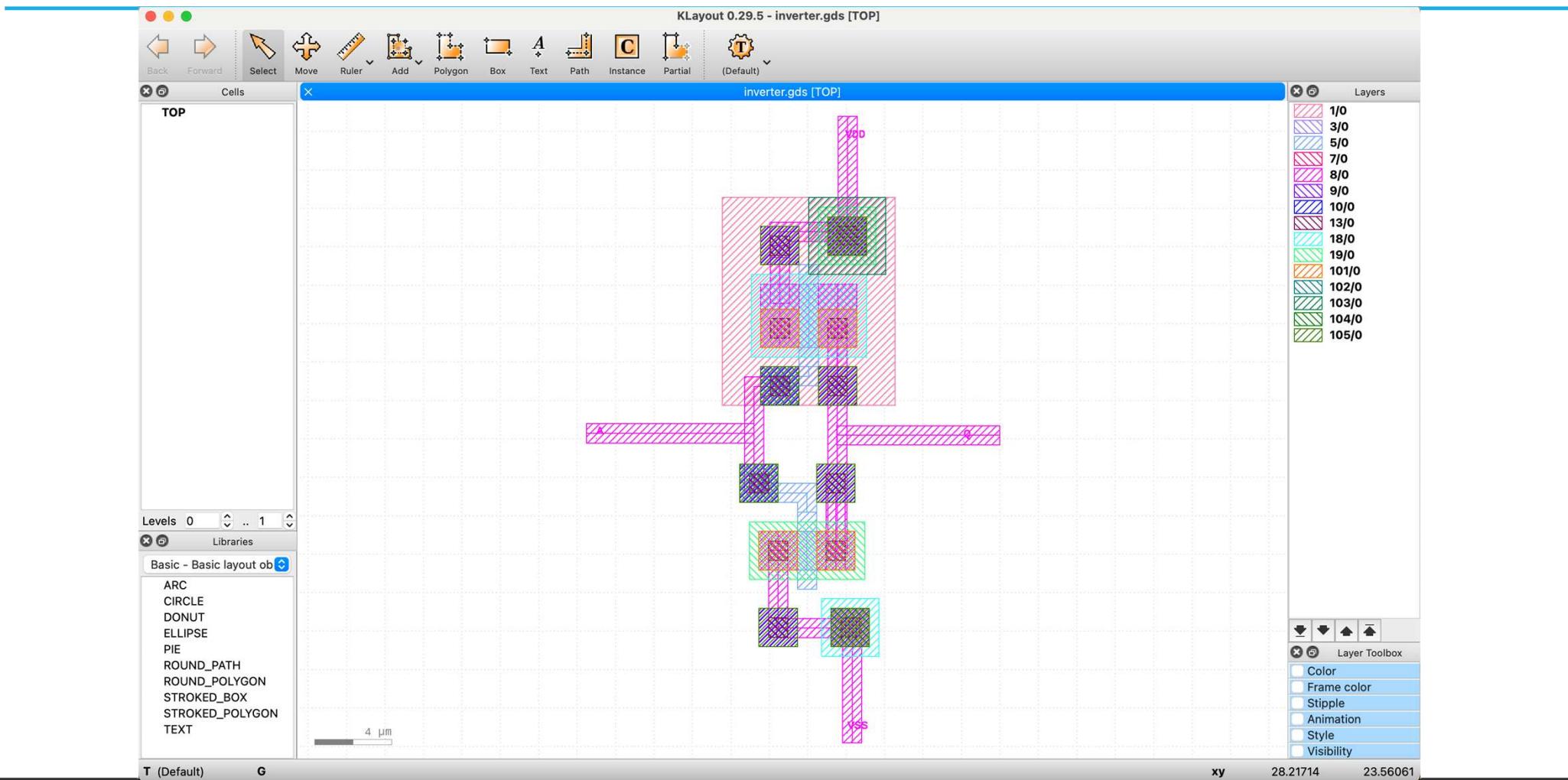
ngspice : 過渡解析シミュレーション結果



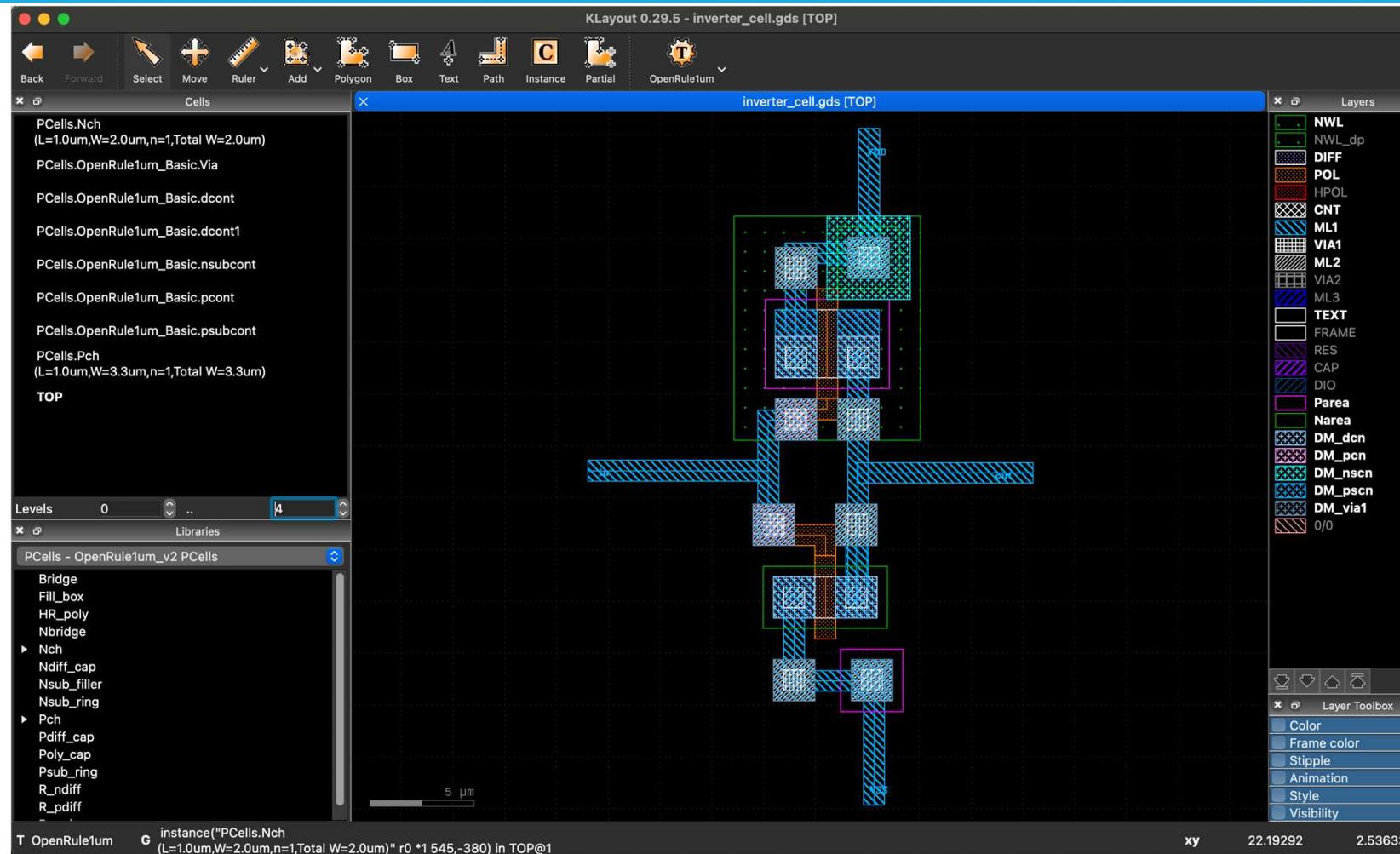
アナログLSIの設計フロー

1. 回路図を描く
2. シミュレーションをする
- 3. 回路図を基にレイアウトを描く**
4. レイアウトを検証する
5. レイアウトを基に寄生成分を考慮したシミュレーションをする
6. (フレームに載せる)

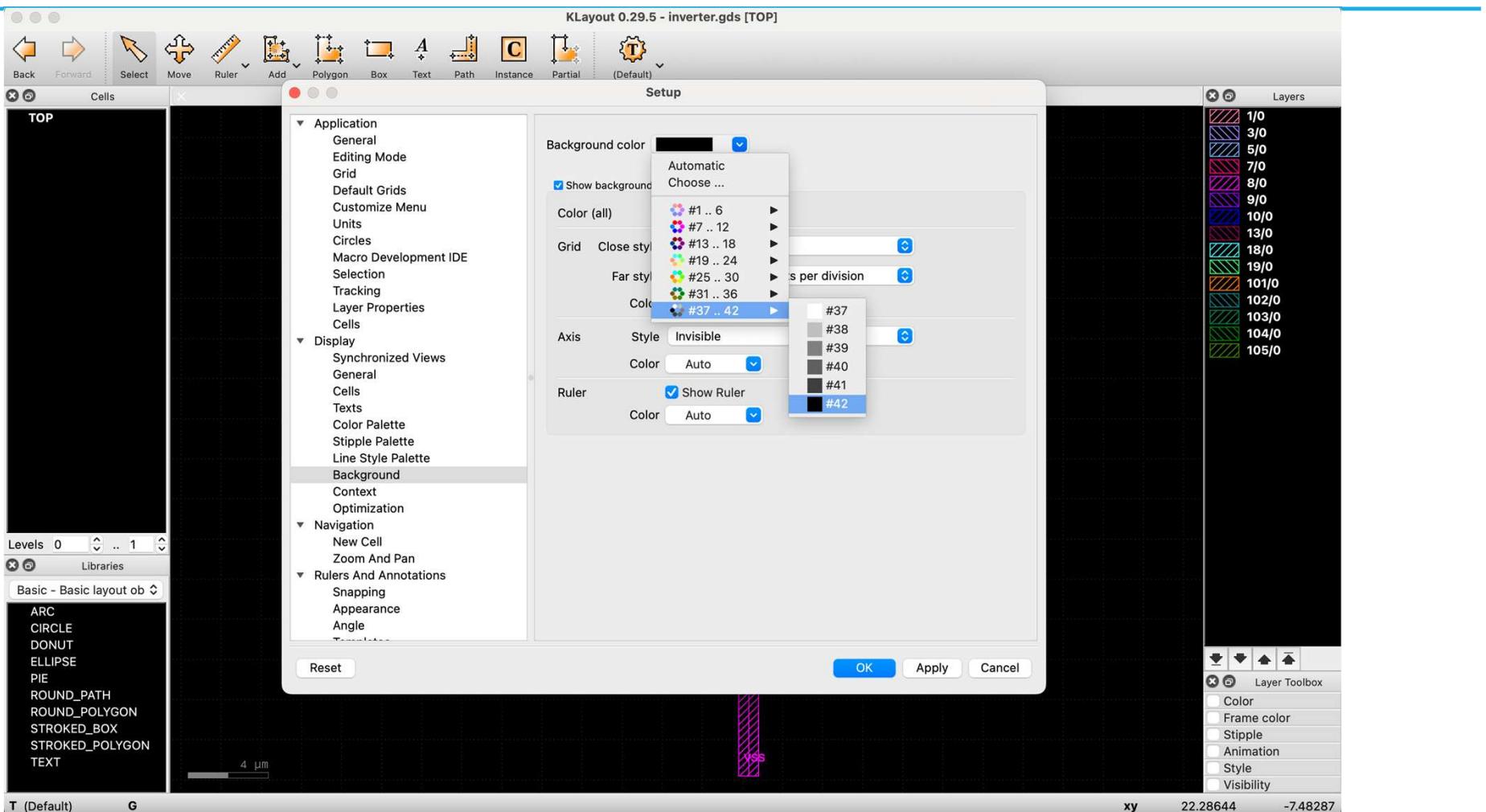
KLayout : 公式PDKのレイアウト画面



KLayout : バックグラウンドが「黒」のレイアウト画面

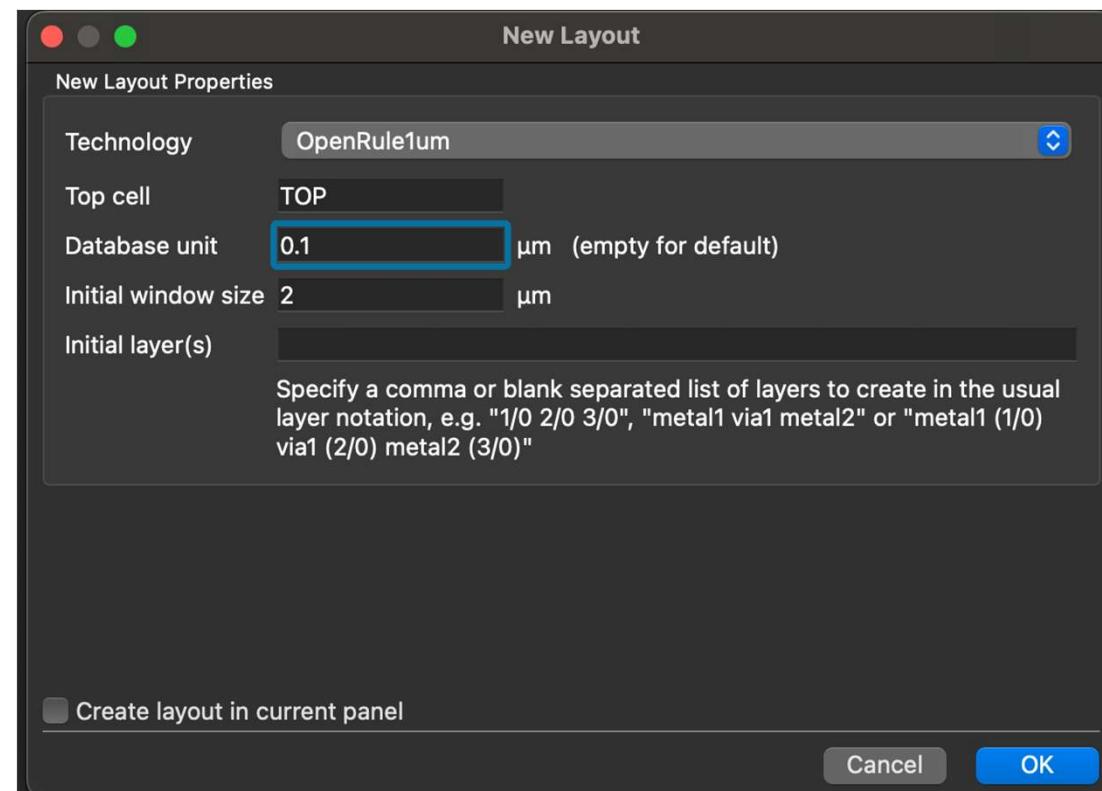


KLayout : バックグラウンドを「黒」にする



Klayout : レイアウトを描く上での注意

- Technologyを OpenRule1umにする
- Database unit を 0.1にする

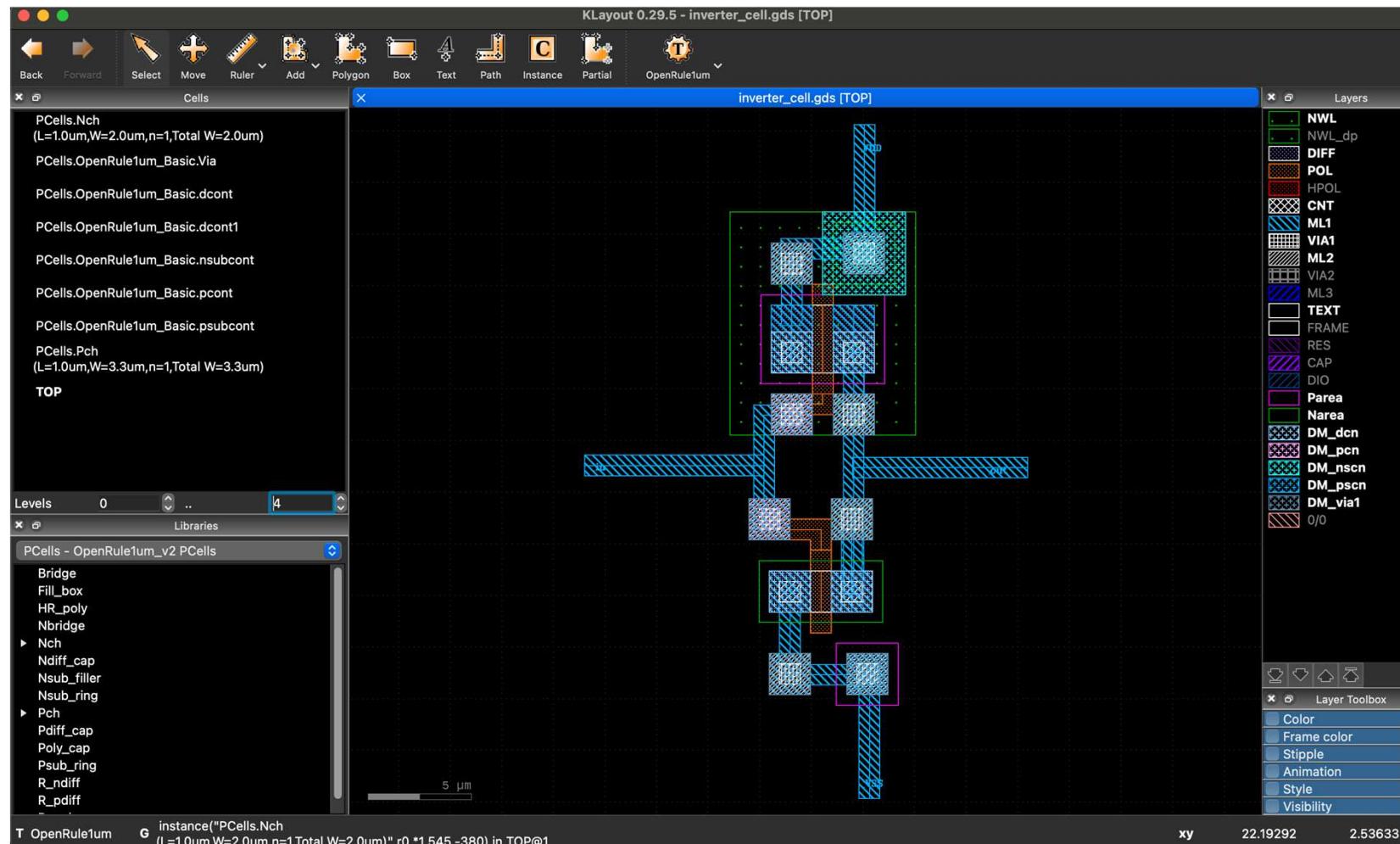


Klayout : デザインルールの場所

- 必ずデザインルールを守る

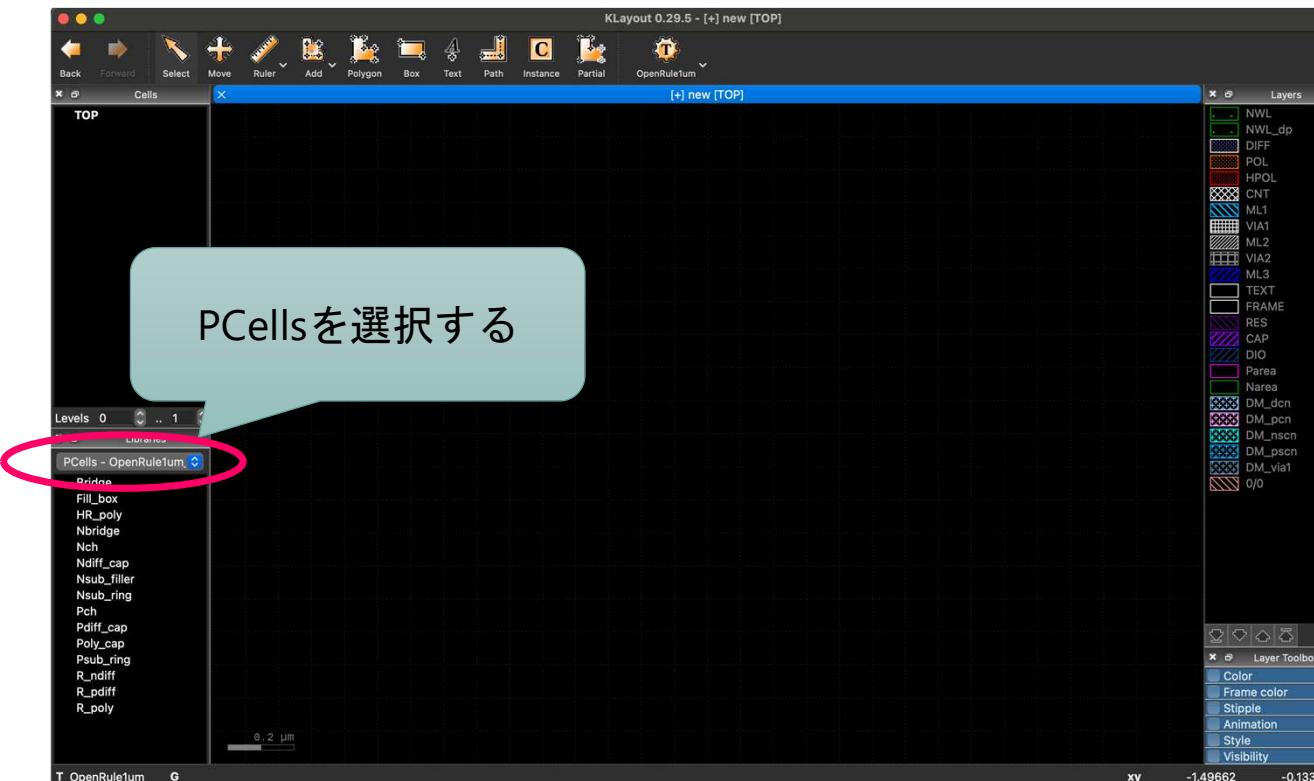
<https://github.com/mineda-support/OpenRule1um>

KLayout : 完成サンプル



KLayout : P-Cellの使い方

- P-Cell（パラメータ化セル）とは、EDAで使用される再利用可能で構成可能なブロックのことです。P-Cellは、集積回路のレイアウトパターンを特定の設計要件に応じて調整可能なパラメータで作成することができます。これにより、各プロジェクトごとにレイアウト全体を再設計する必要がなくなり、設計プロセスの効率と柔軟性が大幅に向上します。



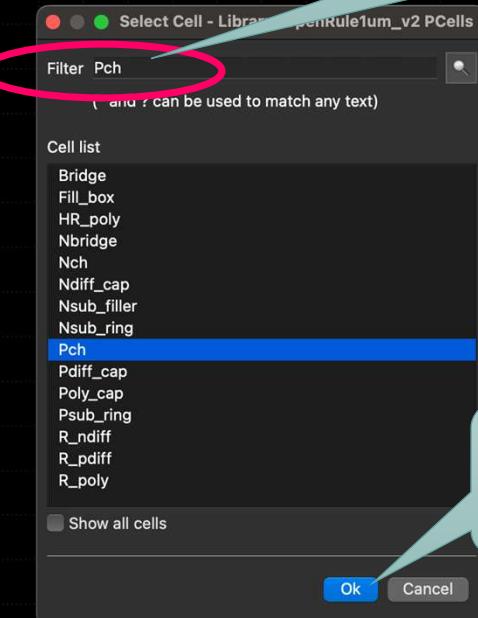
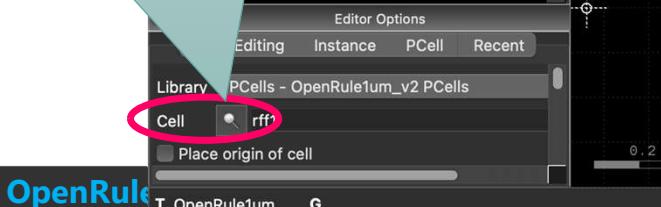
Klayout : P-Cellの使い方

1, Instanceをクリック

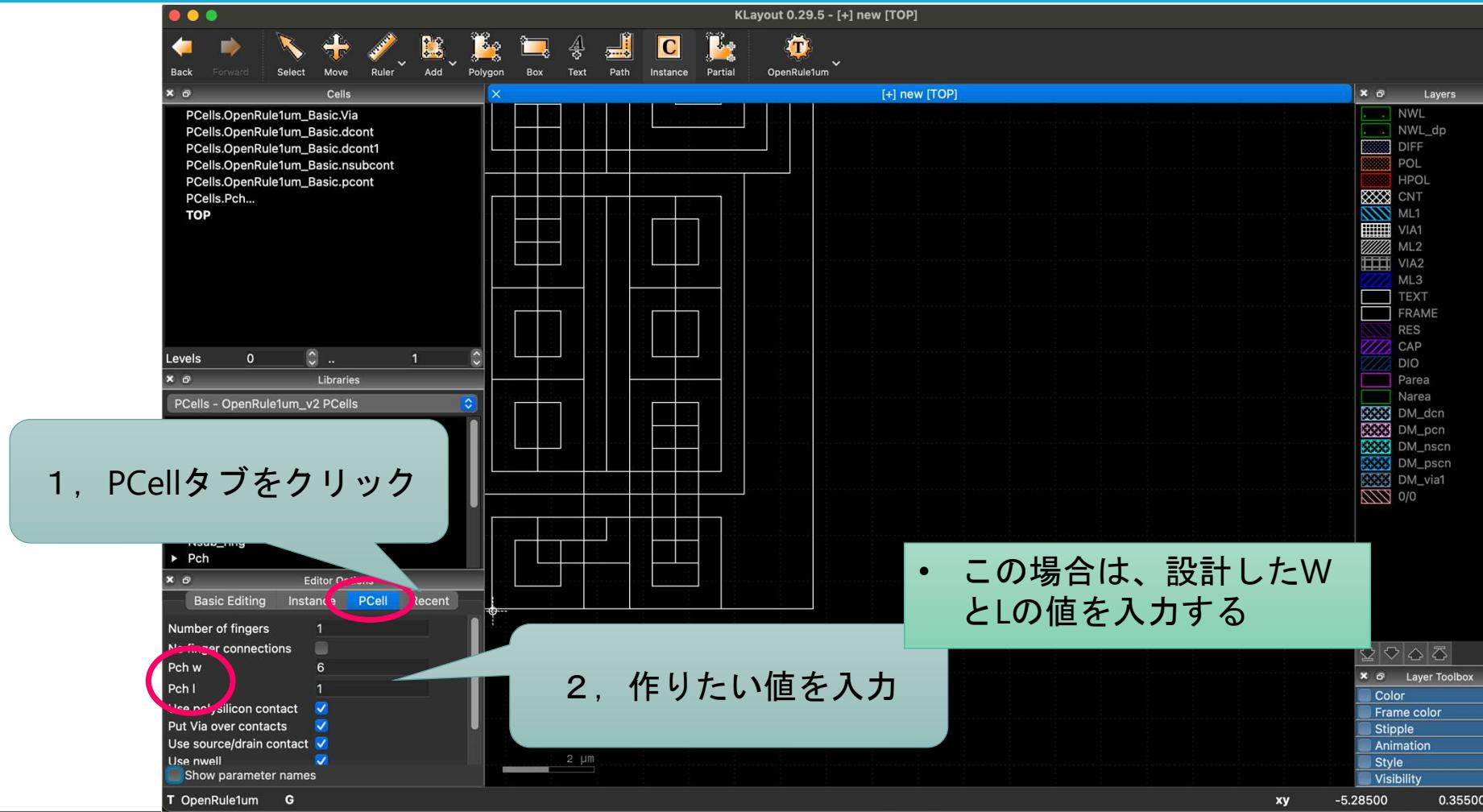
3, 目当てのPcellの名前
を入力する

4, OKをクリック

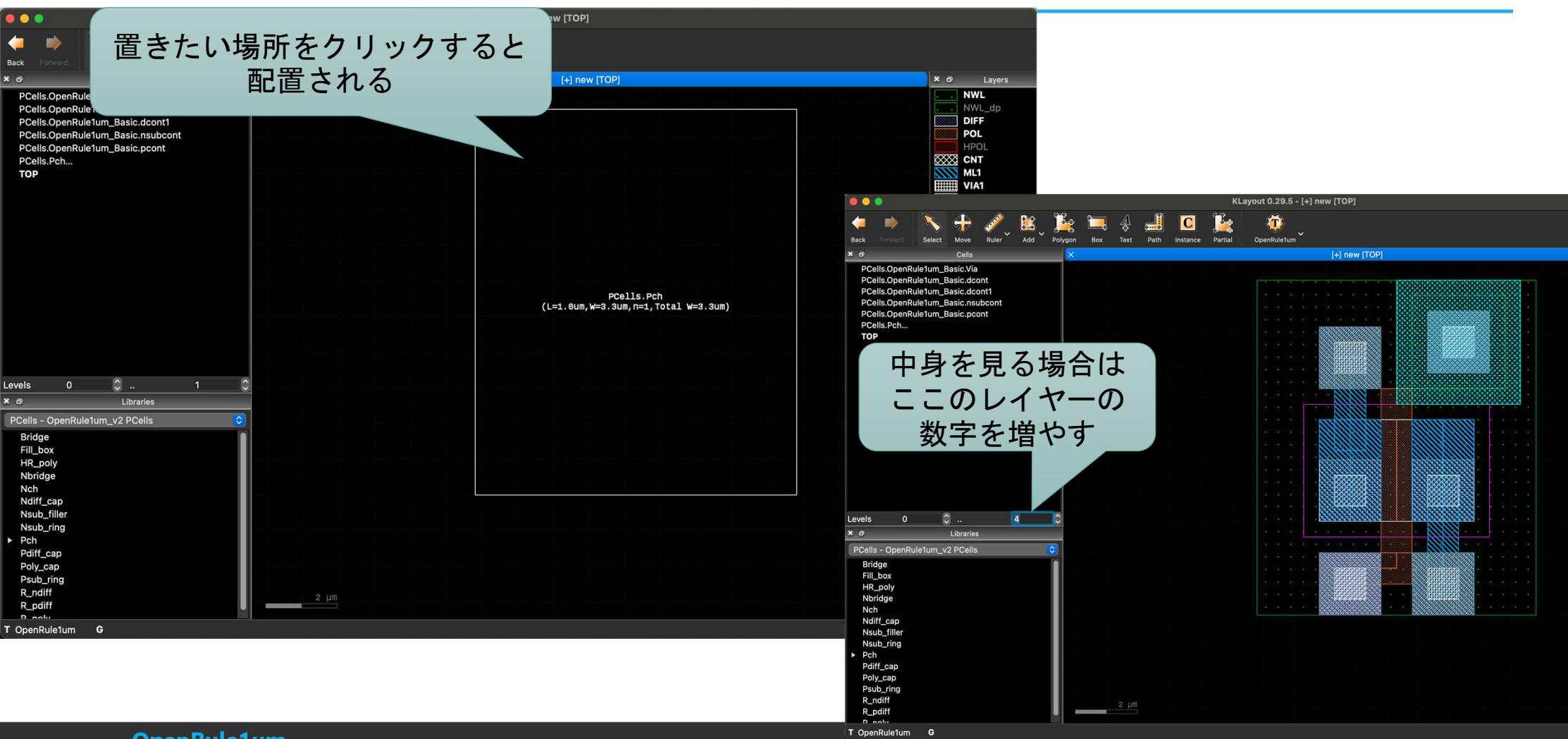
2, 虫眼鏡をクリック



KLayout : P-Cellの使い方

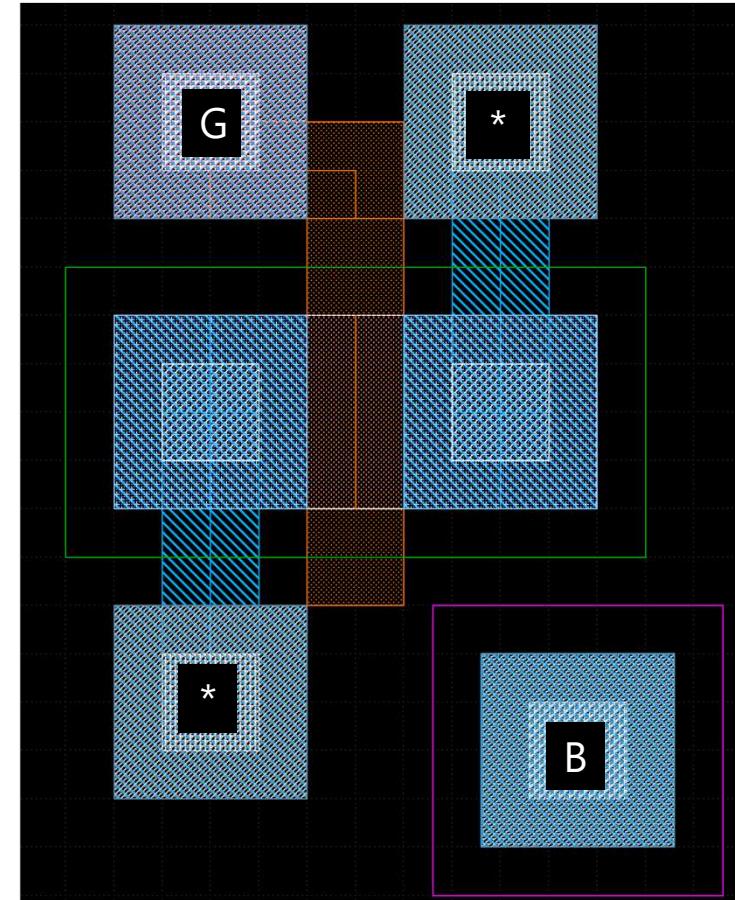
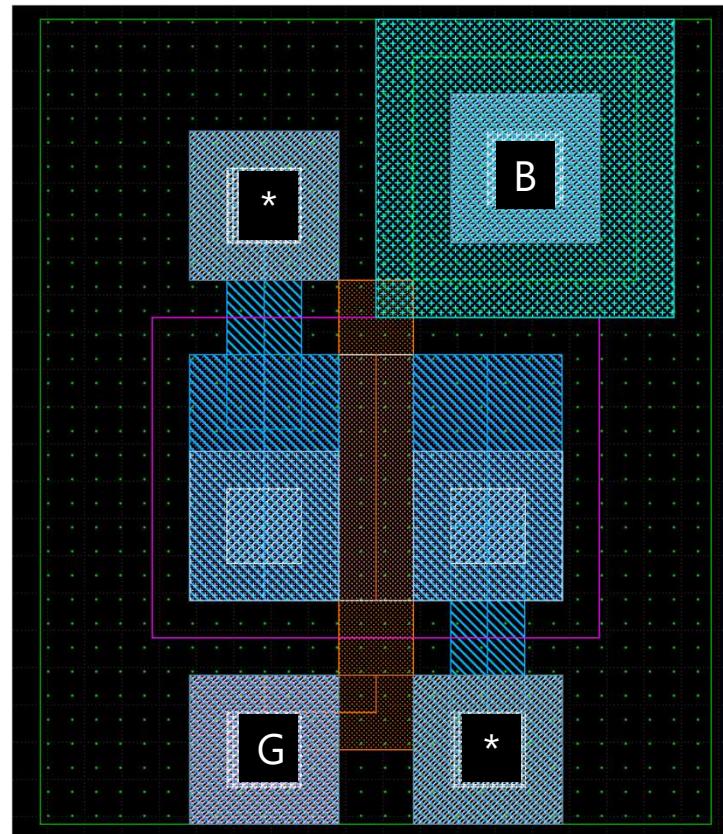


Klayout : P-Cellの使い方



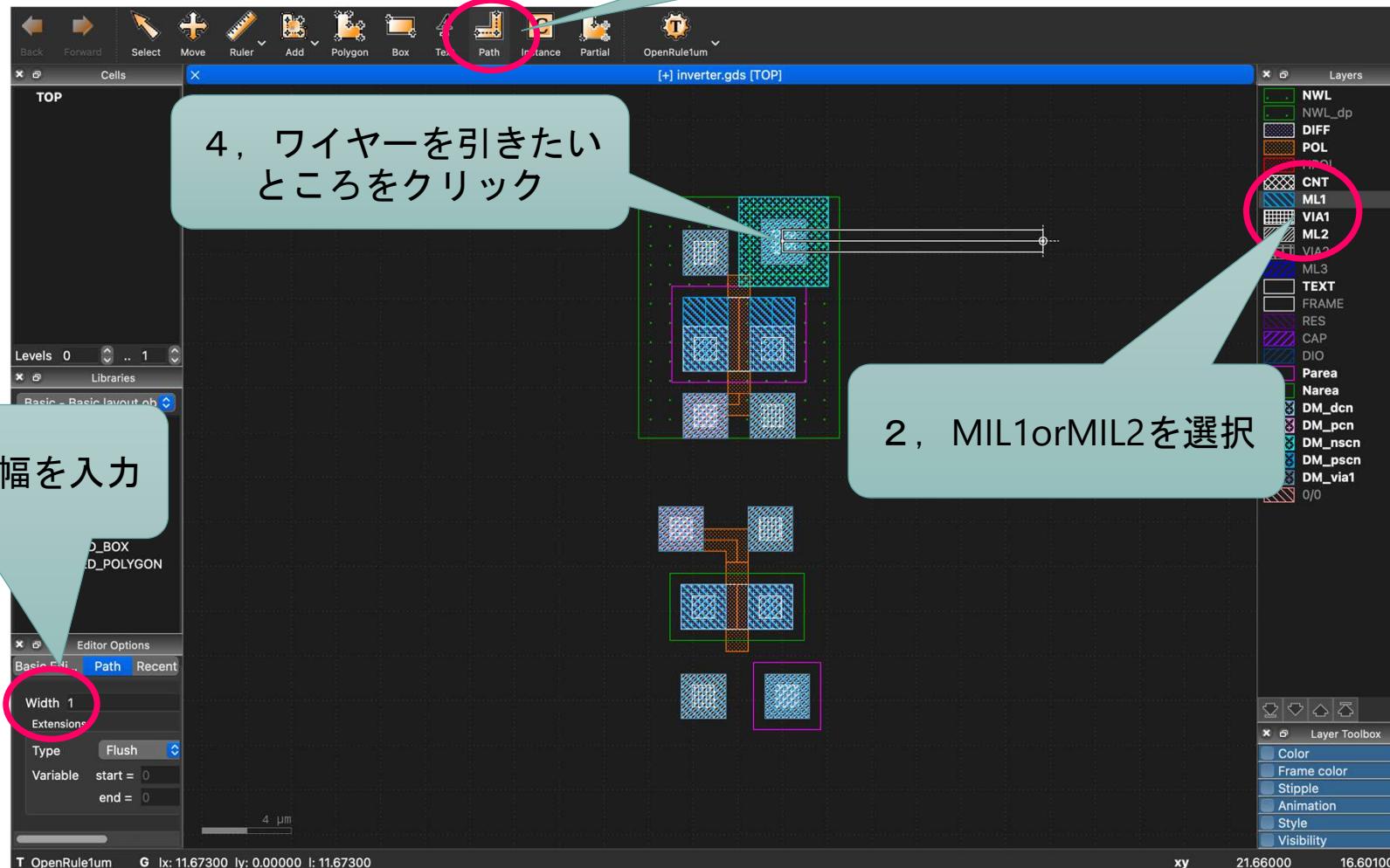
Klayout : タップ付 PFET と NFET

- [*] どちらをドレインにしてどちらをソースにするかは任意。

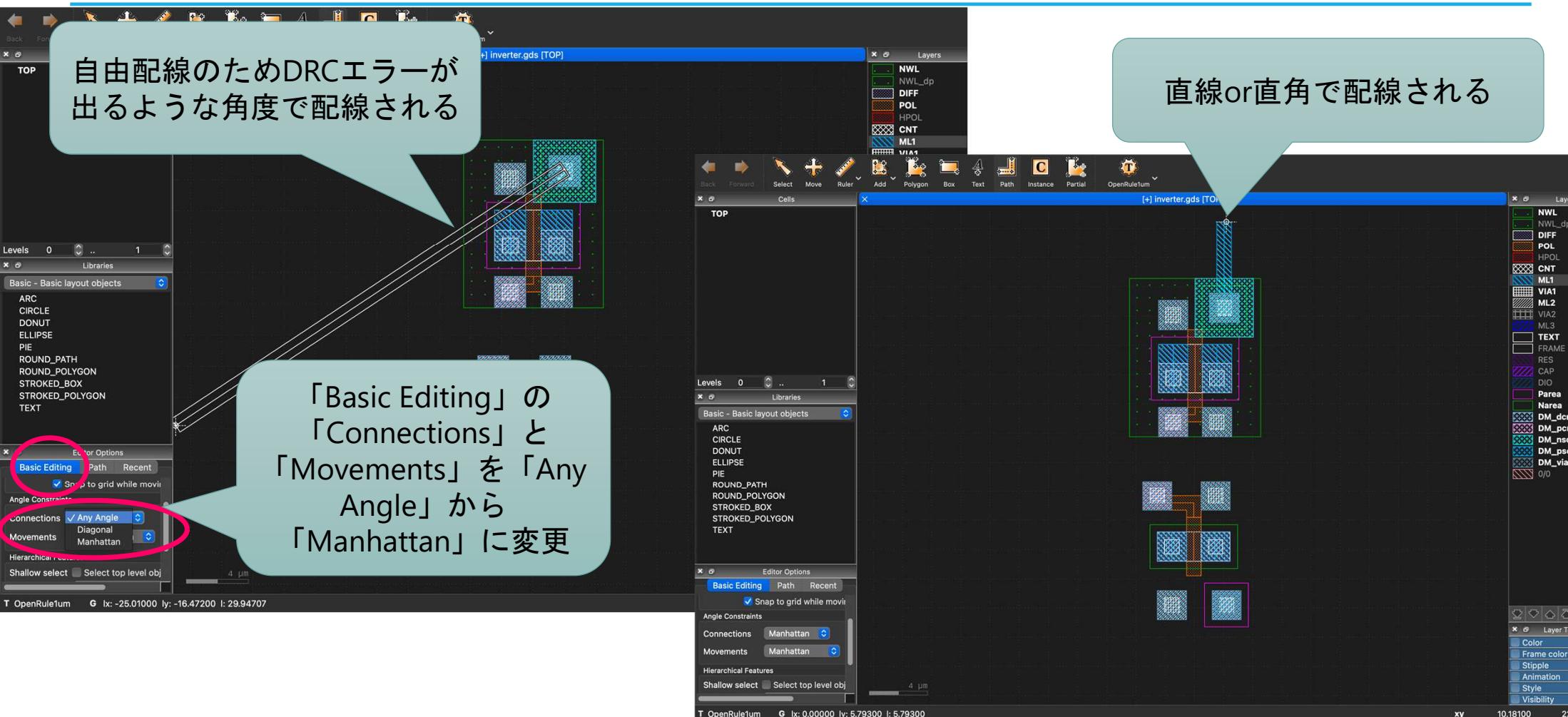


Klayout : ネットリスト（ワイヤー）の引き方

1, Pathをクリック



Klayout : ネットリスト（ワイヤー）の設定

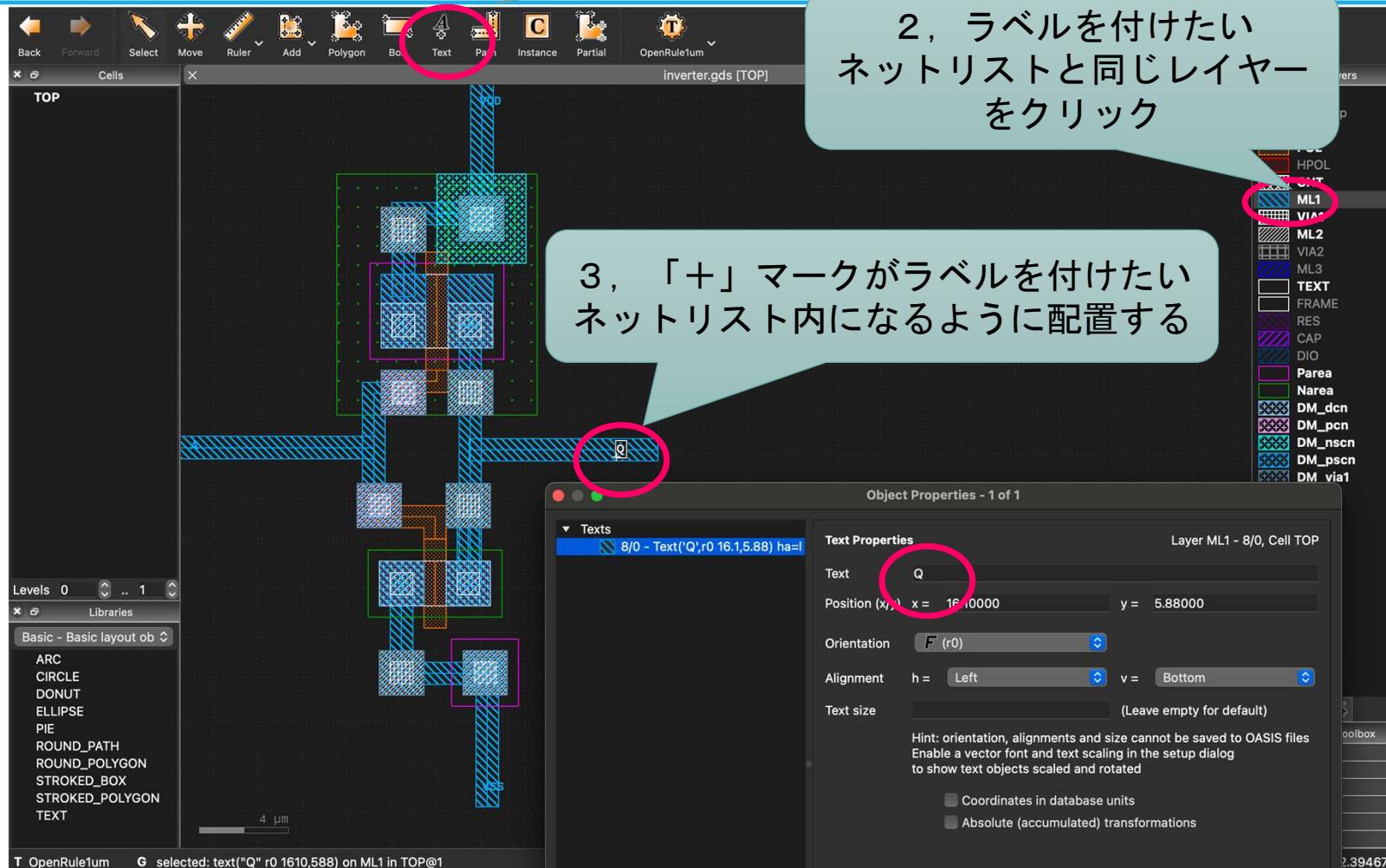


Klayout : ラベルの付け方

1, TEXTをクリック

2, ラベルを付けたい
ネットリストと同じレイヤー
をクリック

3, 「+」マークがラベルを付けたい
ネットリスト内になるように配置する



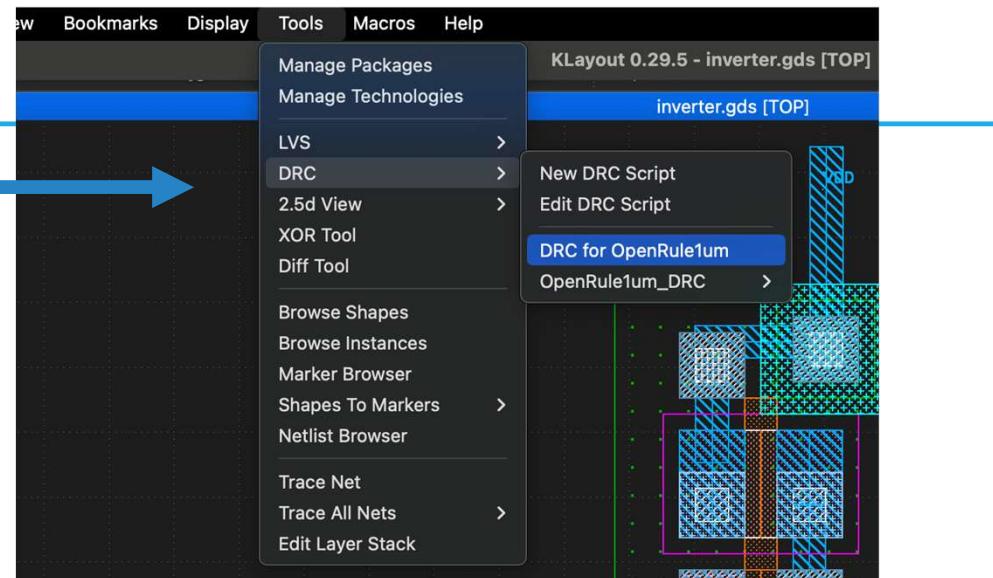
アナログLSIの設計フロー

1. 回路図を描く
2. シミュレーションをする
3. 回路図を基にレイアウトを描く
- 4. レイアウトを検証する**
5. レイアウトを基に寄生成分を考慮したシミュレーションをする
6. (フレームに載せる)

KLayout : レイアウト検証

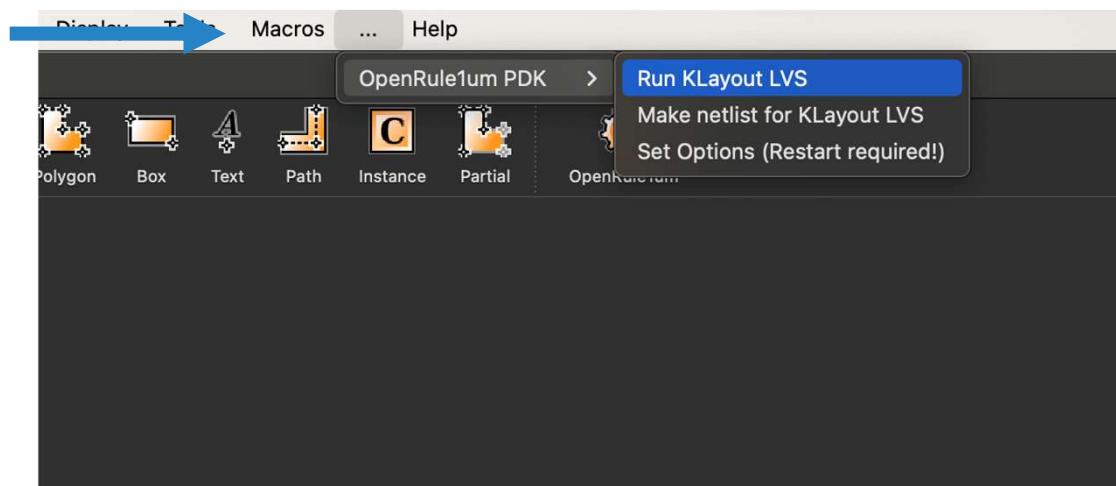
Design Rules Check (DRC)

- 指定されたデザインルールから違反していないか検証

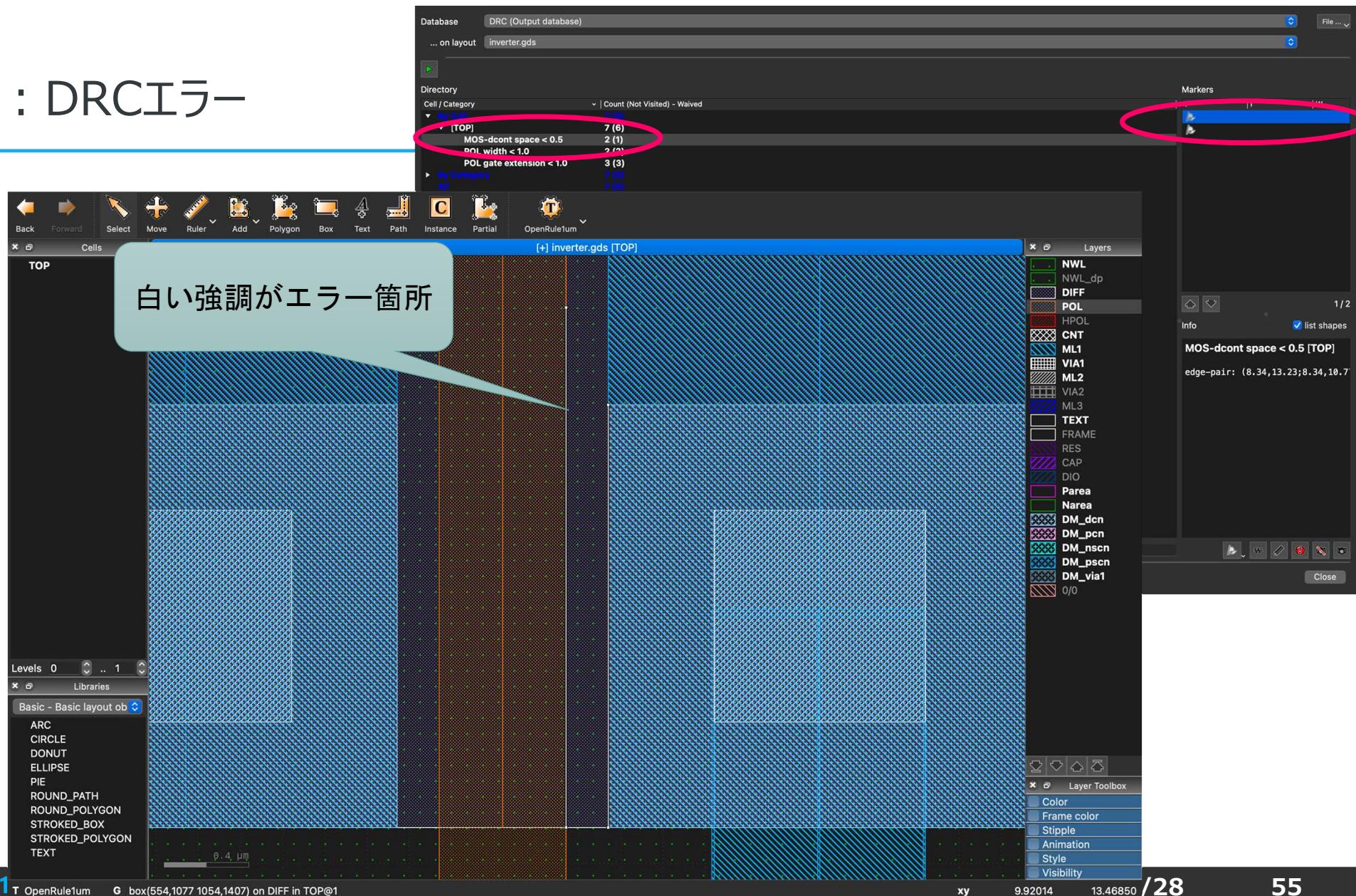


Layout Versus Schematic (LVS)

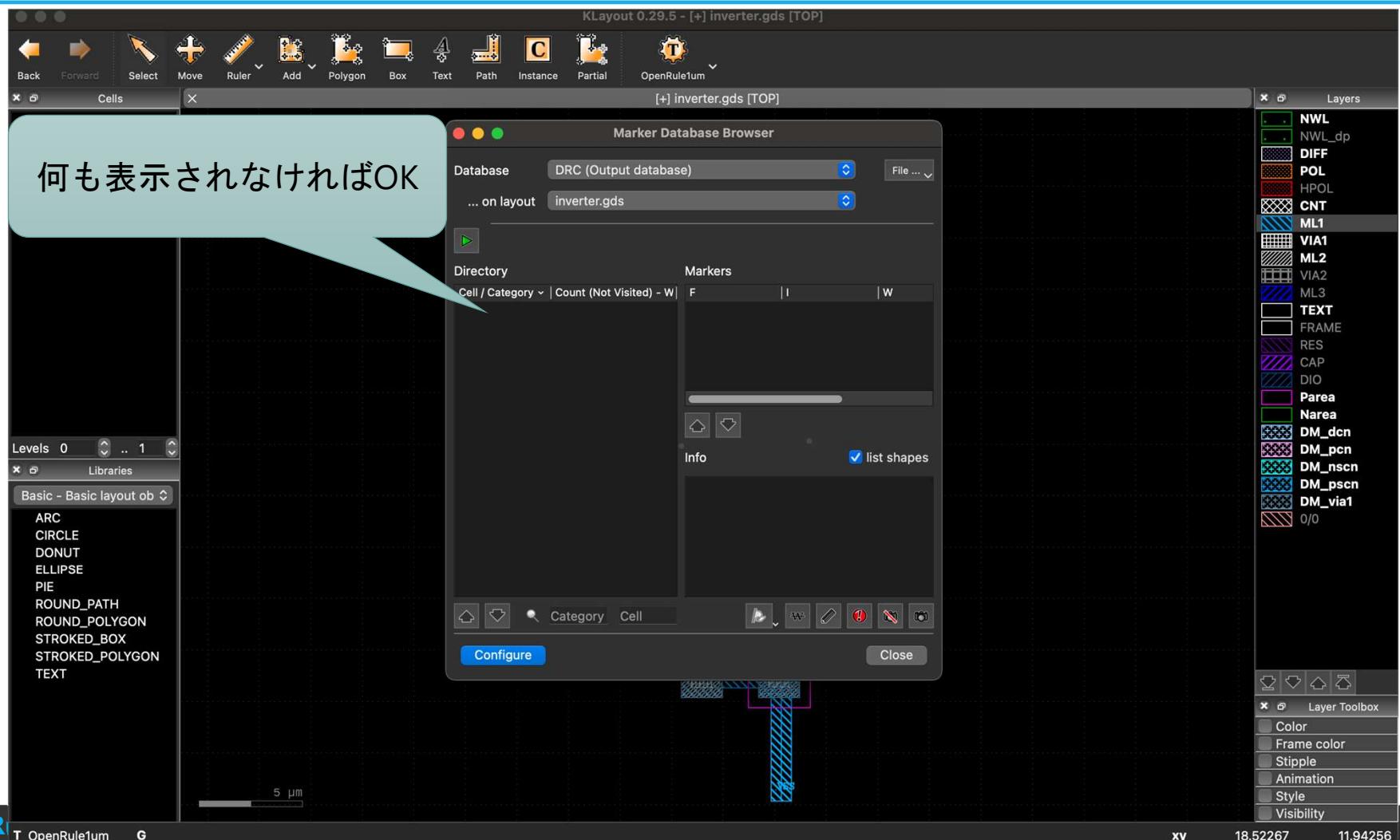
- レイアウトが回路図通りに描けているか検証



Klayout : DRCエラー

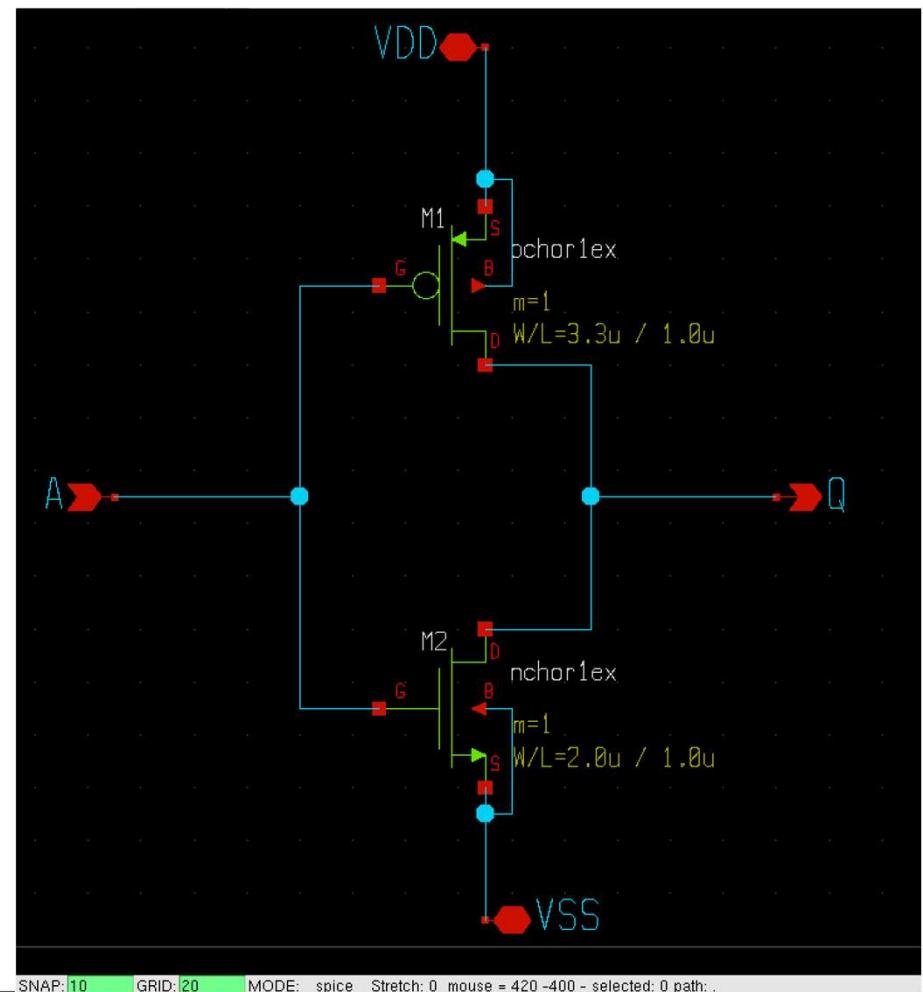


Klayout : DRC OK



Klayout : LVS用ネットリストの出力方法 (1/3)

- xschem上であらかじめLVS用の回路図を作成しておく。
- ピンはipin, iopin, opin のみを使用する。
 - vdd.symやgnd.symを使用するとうまく通らない。
- ファイルを同じ名前にする 例: nand.sch, nand.gds

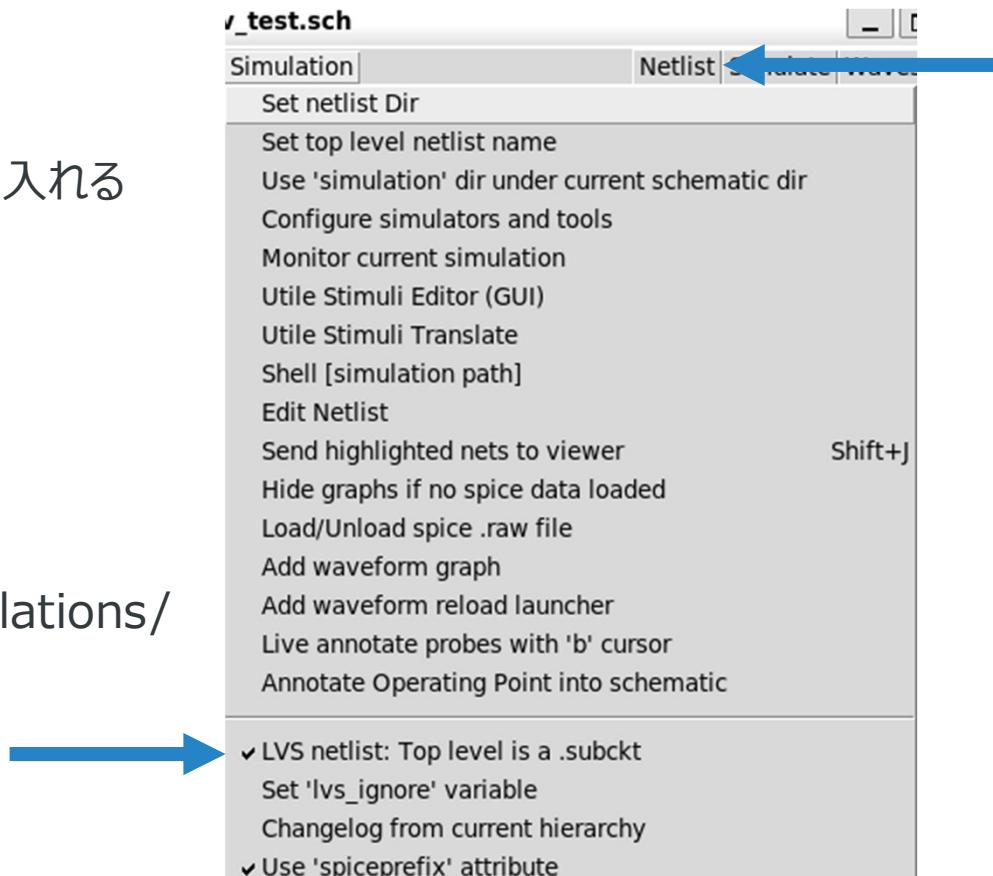


Klayout : LVS用ネットリストの出力方法 (2/3)

- xschem上で**LVS用**のネットリストを出力する。

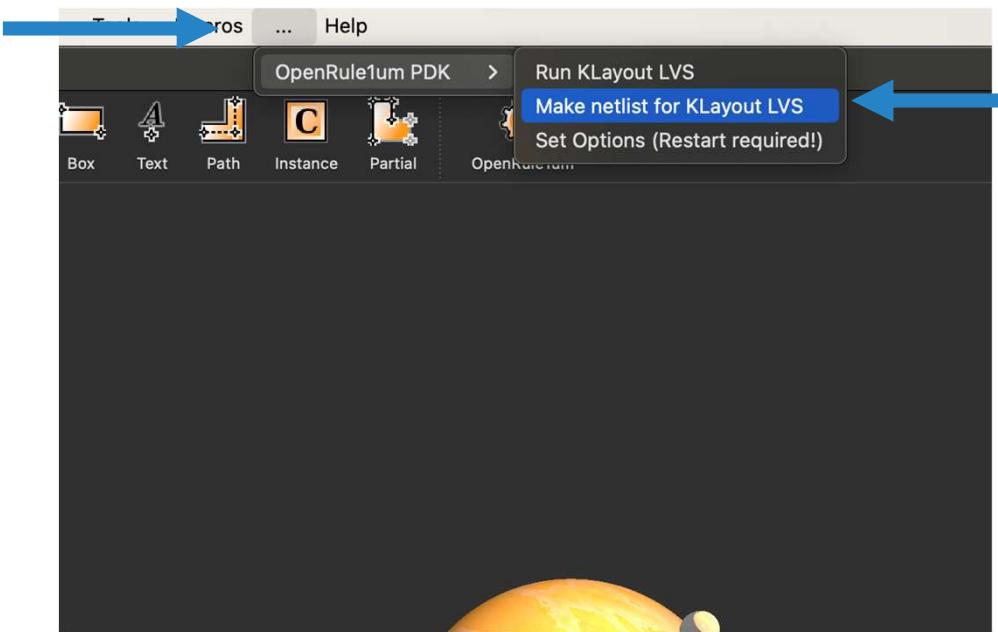
- メニュー Simulation > LVS netlist に✓を入れる
- Netlistを押す

- 終わったら閉じて良い
- デフォルトの生成場所は ~/xschem/simulations/



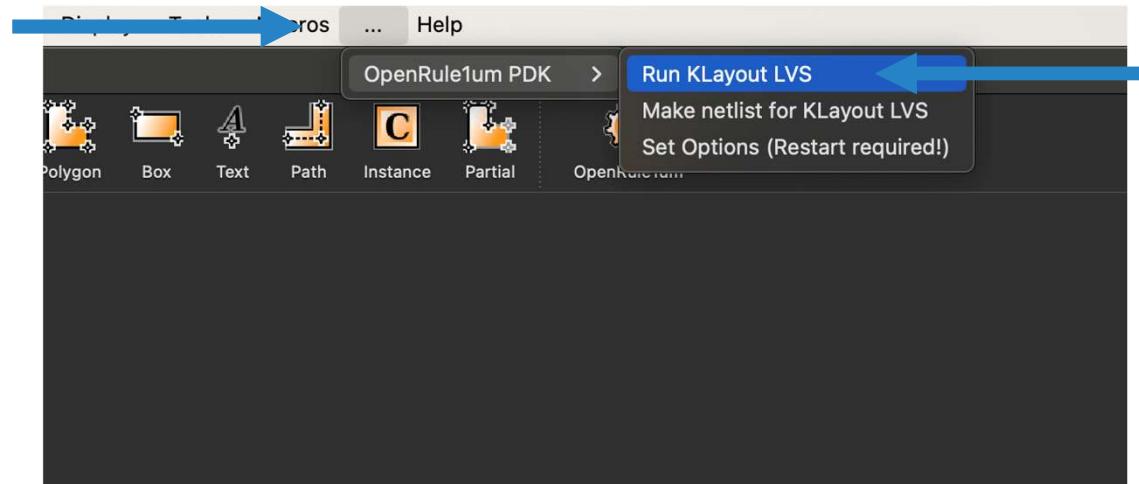
Klayout : LVS用ネットリストの出力方法 (3/3)

- klayout上で**LVS用**のネットリストを出力する。
1. メニュー OpenRule1um PDK > Make netlist for KLayout LVS を実行する

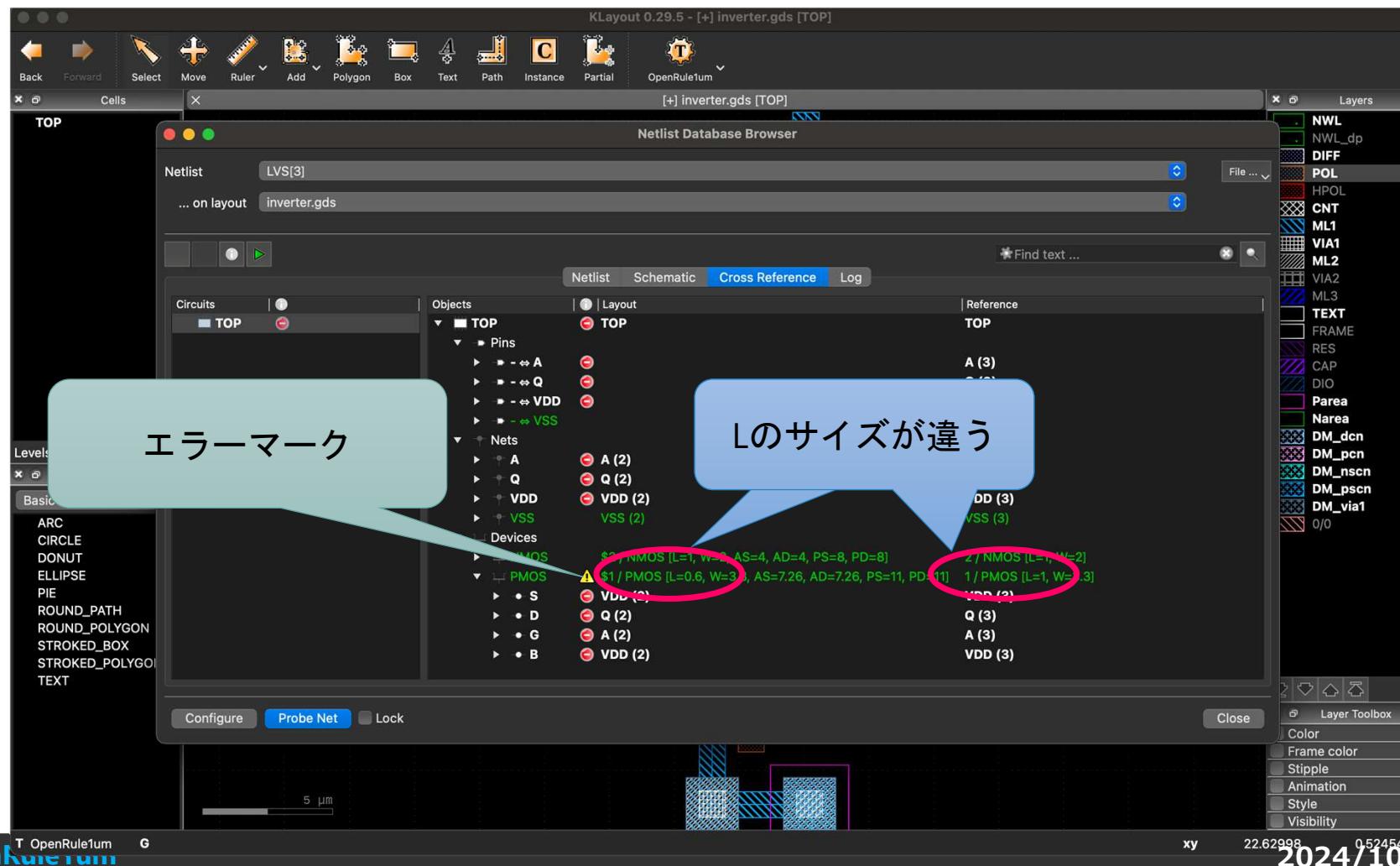


Klayout : LVSの実行

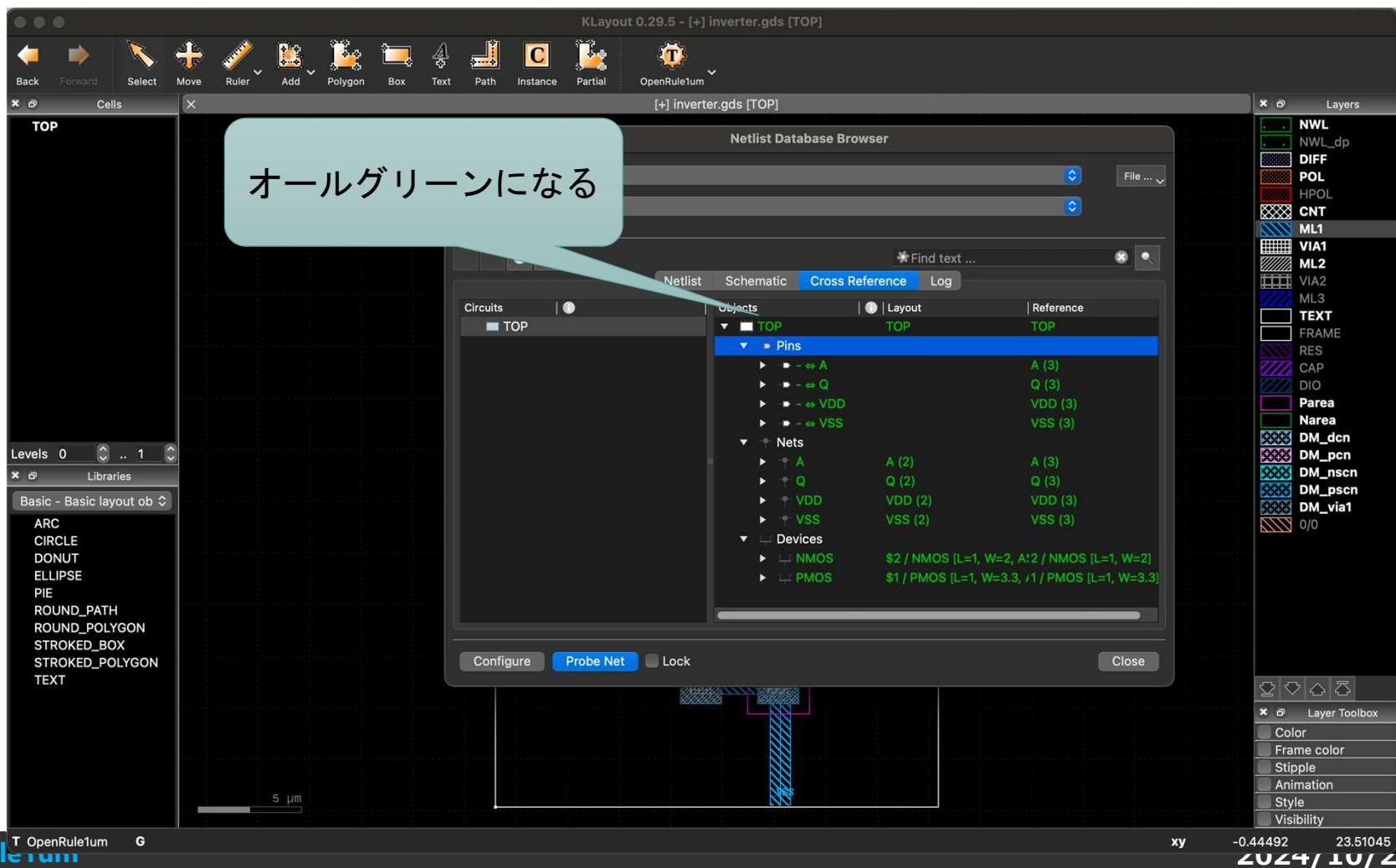
- klayout上でネットリスト比較を実行する。
1. メニュー OpenRule1um PDK > Run KLayout LVS を実行する



Klayout : LVS NG



Klayout : LVS OK



提出

- 本日
 - 提出物：回路図、レイアウト、githubのアカウント、連絡先（チップ送付先）
 - 1, 自分のGithub上にリポジトリを作ってもらって、schファイルとgdsファイルをアップロードする
 - 2, READMEに感想などを記述する
 - 3, DiscordでgithubのURLを告知する
 - 4, ConnpassのIDもお願いします。
- 数日中
 - 提出物：公開用の一言など
- 来年3月～4月
 - イベント：チップが届きますので、お渡し会を兼ねたチップ測定会を開催します