

SKY130で学ぶLSI回路設計

アナログLSI回路設計 基礎編

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

更新日: 2024/12/31

更新履歴

2024/09/21

初版

2024/09/23

ラボデータのファイル名を記載するなど 修正あり

2024/12/31

改訂

Labデータ

- 開発環境セットアップスクリプト (Windows WSL + Ubuntu 22)
 - https://github.com/3zki/wsl_osic
- アナログLSI設計 基礎編
 - <https://github.com/3zki/lab240921>

git clone 等でダウンロードしてください

アジェンダ： アナログLSI設計 基礎編

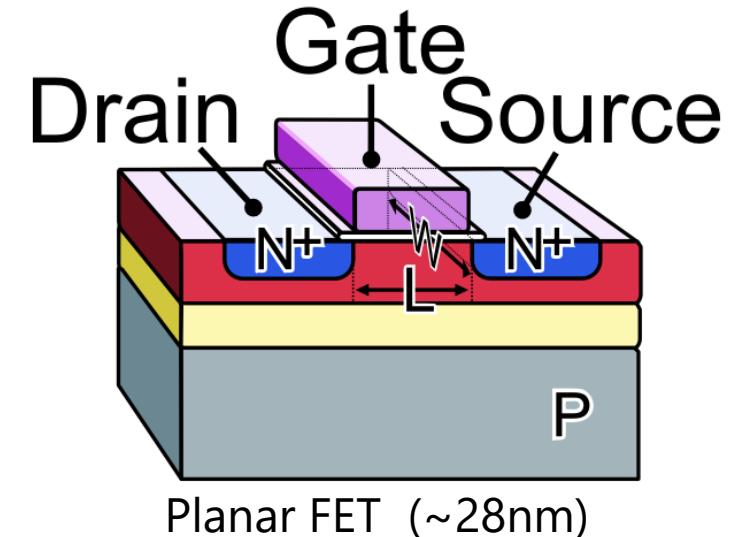
- テーマ：CMOSインバータを作る
 - LSI予備知識
 - 回路設計
 - レイアウト設計
 - ポストレイアウトシミュレーション
- +まとめ、小ネタ集

LSI予備知識

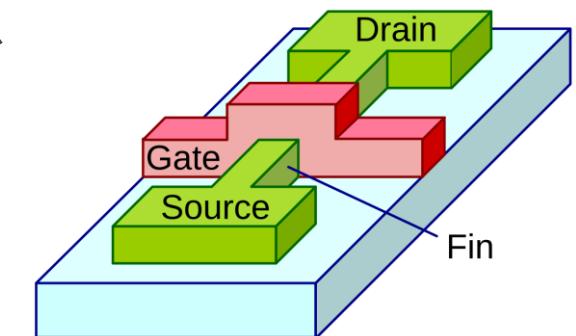
プレーナ型FETプロセス

What is LSI?

- 一言にLSIと言っても、様々な種類があり、設計手法もそれぞれで異なる
- SKY130とGF180MCUは**プレーナ型FETプロセス** ➡
P型MOSFETとN型MOSFETが作成可能
- プレーナ型FETは28nm程度が限界。
CPUのような高密度集積回路ではFinFETやGAA-FETといった
全く別の構造をしたプロセスを使用しており、線幅もより狭い。
- プレーナ型 130nmはロジック回路用としては時代遅れ（2000年前後）だが、
FinFETよりも安く製造でき、そしてアナログLSIとしては現役である。



Planar FET (~28nm)



FinFET (25nm ~ 7nm)

https://commons.wikimedia.org/wiki/File:Multigate_models.png

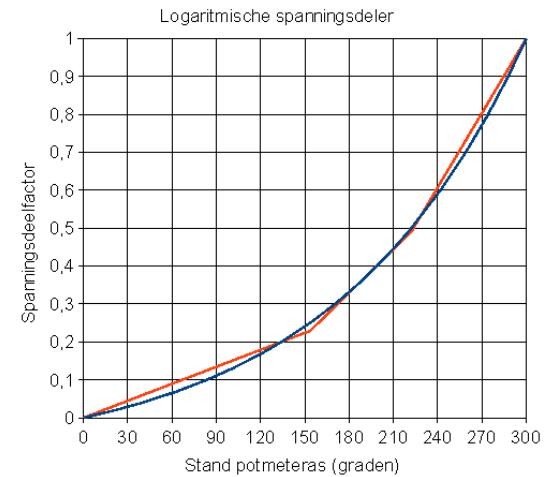
[https://commons.wikimedia.org/wiki/File:MOS-FET_gate_model_\(n-channel\)_E.PNG](https://commons.wikimedia.org/wiki/File:MOS-FET_gate_model_(n-channel)_E.PNG)

プレーナ型FETプロセスで作れる素子

- プレーナMOSFET: **PMOS, NMOS**
 - SKY130では1.8V, 3.3V 5.0V, 10.5V, 16V, 20V の耐圧をサポート
- ダイオード
- 抵抗: **ポリ抵抗**, 拡散層抵抗（アクティブ抵抗）, Nwell抵抗
- キャパシタ: **ポリ容量**, 拡散層容量, **MIMCAP** (プロセスによっては無い), 配線層の寄生容量
- バラクタダイオード: プロセスによっては無い
- ラテラルバイポーラ (PNP, NPN) : プロセスによっては無い

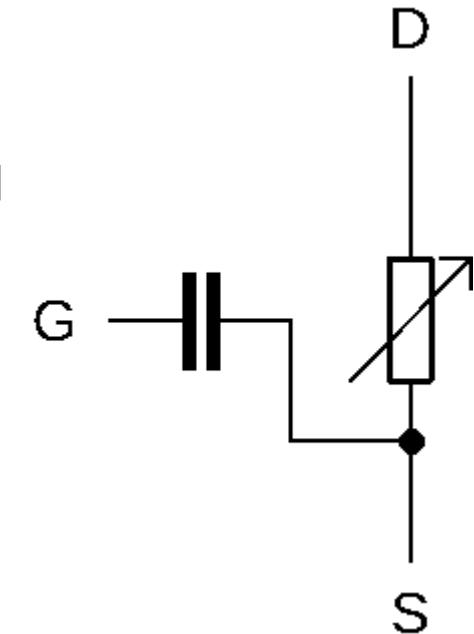
PMOS, NMOSとは

- 可変抵抗（ボリューム）に似ている
 - 0Ωから摺動子(ツマミ)を回すと、抵抗値が増加する
 - スペックでは最大抵抗値とカーブが記載
 - 抵抗値の増え方にバリエーションがある（カーブ）



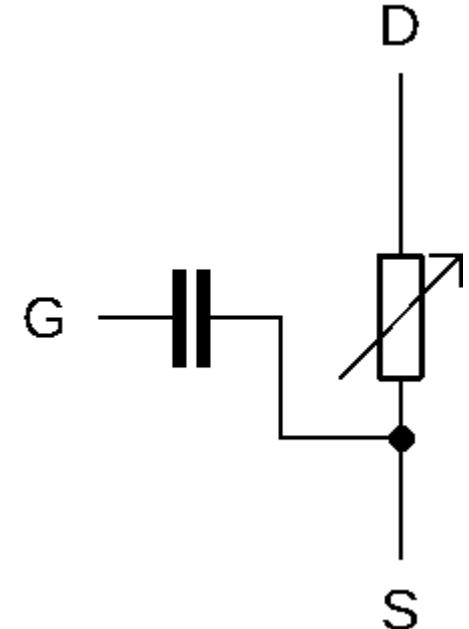
PMOS, NMOSとは

- MOSFETは電流をベースにして考える（抵抗値に置き換えて考える事は基本的ではない）
- MOSFETは可変抵抗の逆
 - 摺動子（ツマミ）ではなくゲート電圧 V_{GS} によってドレインソース間の電流を制御
 - $\infty\Omega$ からゲート電圧を調整すると抵抗値減少
 - スペックでは電流カーブが記載(ドレインソース間電流 I_{DS})
 - 最小抵抗値は電流カーブから計算 (オン抵抗 R_{on})
 - 抵抗値の増え方はプロセス依存 (閾値電圧 V_{th} 、トランスクンダクタンス g_m)

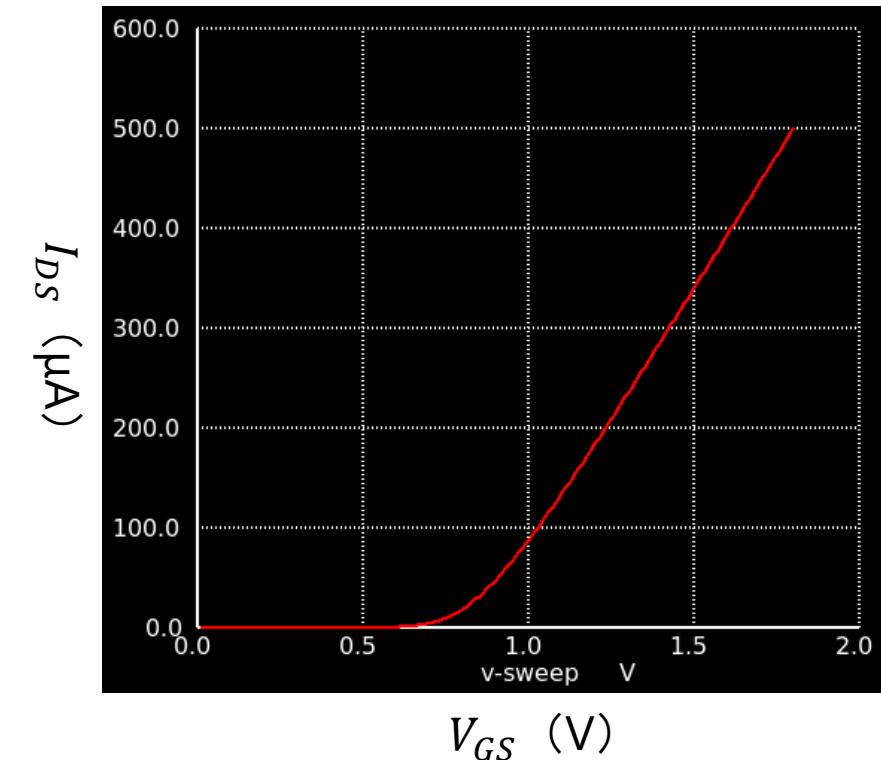
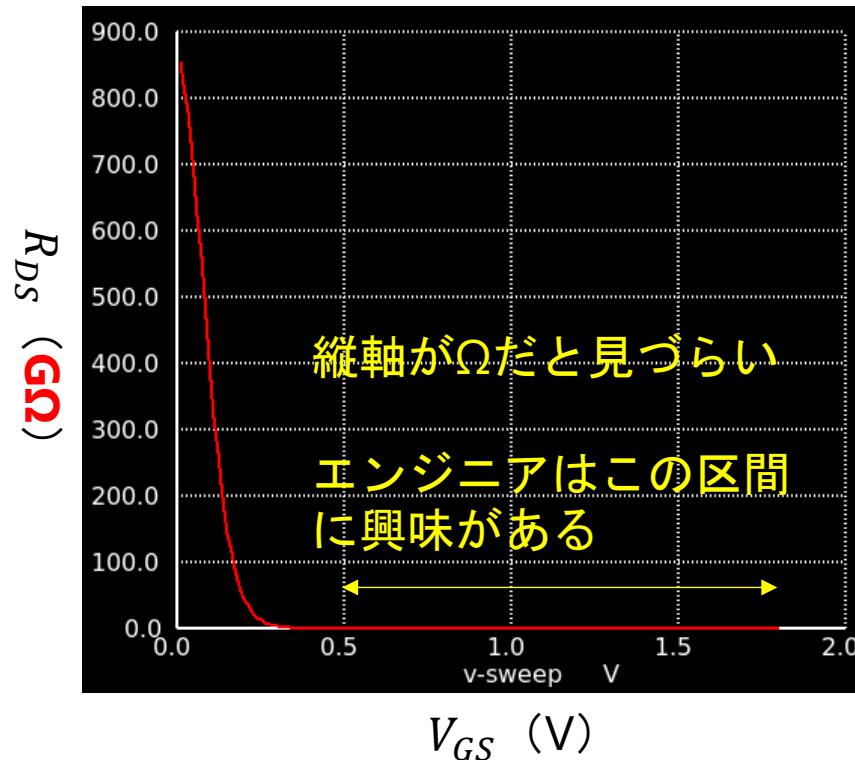
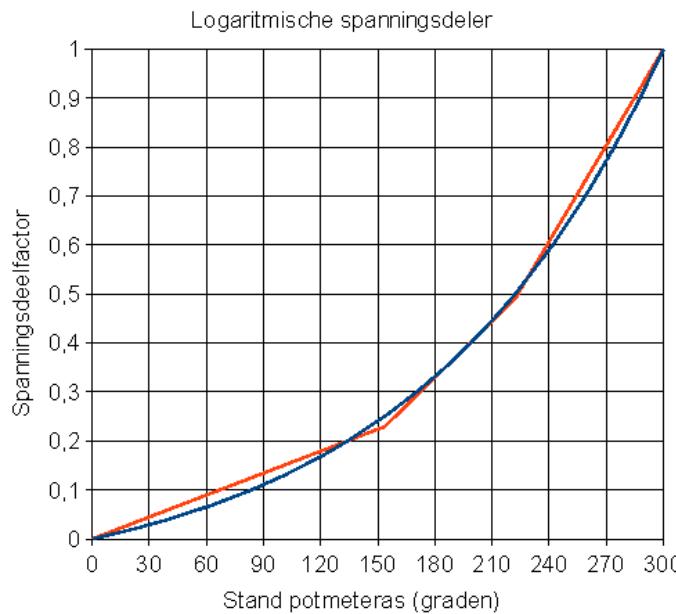


PMOS, NMOSとは

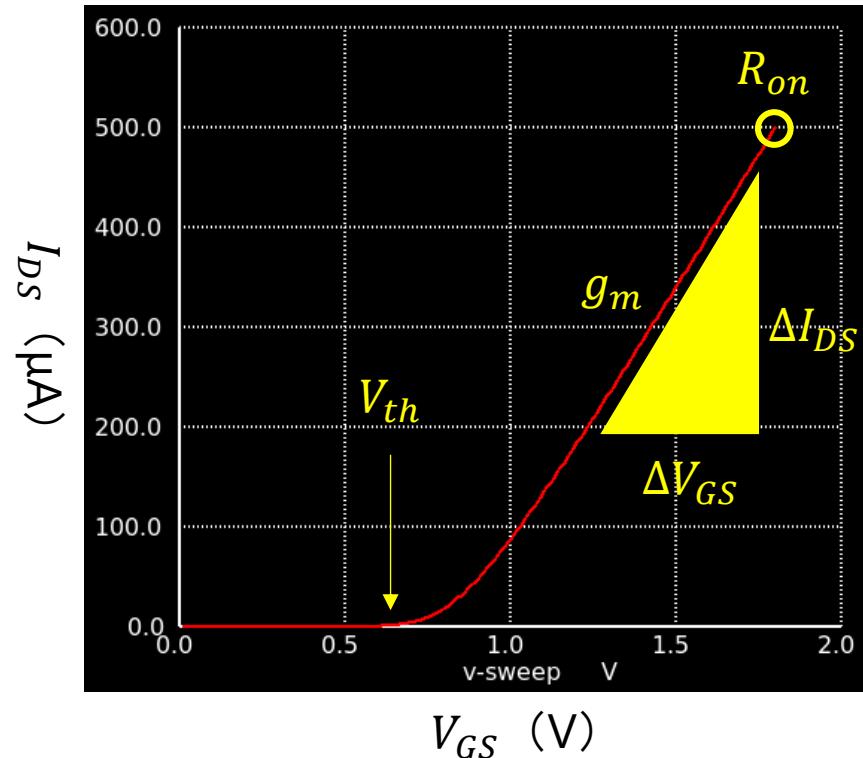
- 右図の等価回路は「MOSFETを敢えてRLCで表すならこうだろう」というもの
 - ドレン・ソース間は「電流源」を置くのが正しいと思われるが、初心者にはあまり馴染みがないので可変抵抗とした
 - MOSFETの重要な特性は **トランスクンダクタンス**とよばれるもの
 $I_{DS} = \frac{1}{2} \times g_m \times (V_{GS} - V_{th})$
 - また特定のドレンソース間電圧領域において、電流源として動作する**飽和(saturation)**とよばれる特性も非常に重要。
 - ゲート容量は**ゲート・ドレン間**と**ゲート・ボディ間**にも存在するが、単純に見映えの問題でゲートソース間のみの記載とした



可変抵抗 vs NMOS



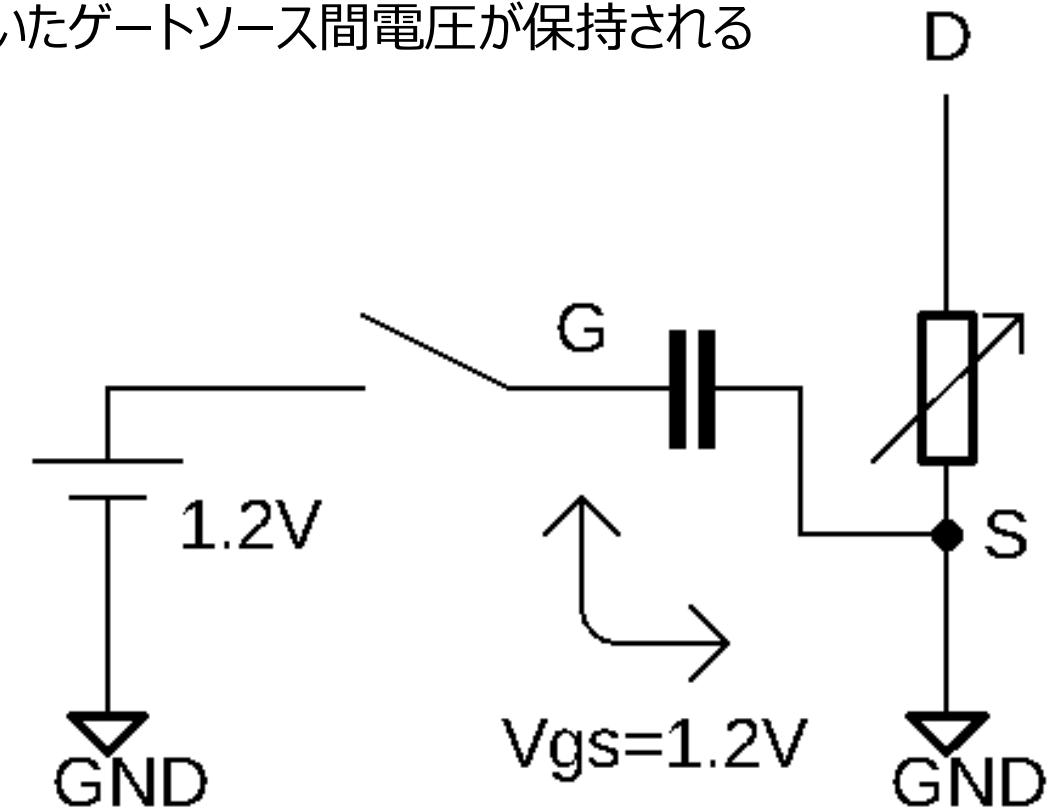
NMOSの特性



- V_{th} 閾値電圧
電流が流れ始めるゲートソース間電圧の閾値
- g_m トランスコンダクタンス
ゲート電圧の変化分 対 ドレインソース間電流の変化分
- R_{on} オン抵抗
オン状態での抵抗値

NMOSの特性：フローティングゲート

- ドレンソース間電流 I_{DS} はゲートソース間の電圧 V_{GS} によって決定するが、**ゲートには寄生容量がある**
- ゲートがフローティング状態になった時は直前に入力されていたゲートソース間電圧が保持される
 - フローティングゲートと呼ばれる状態 SRAMで利用



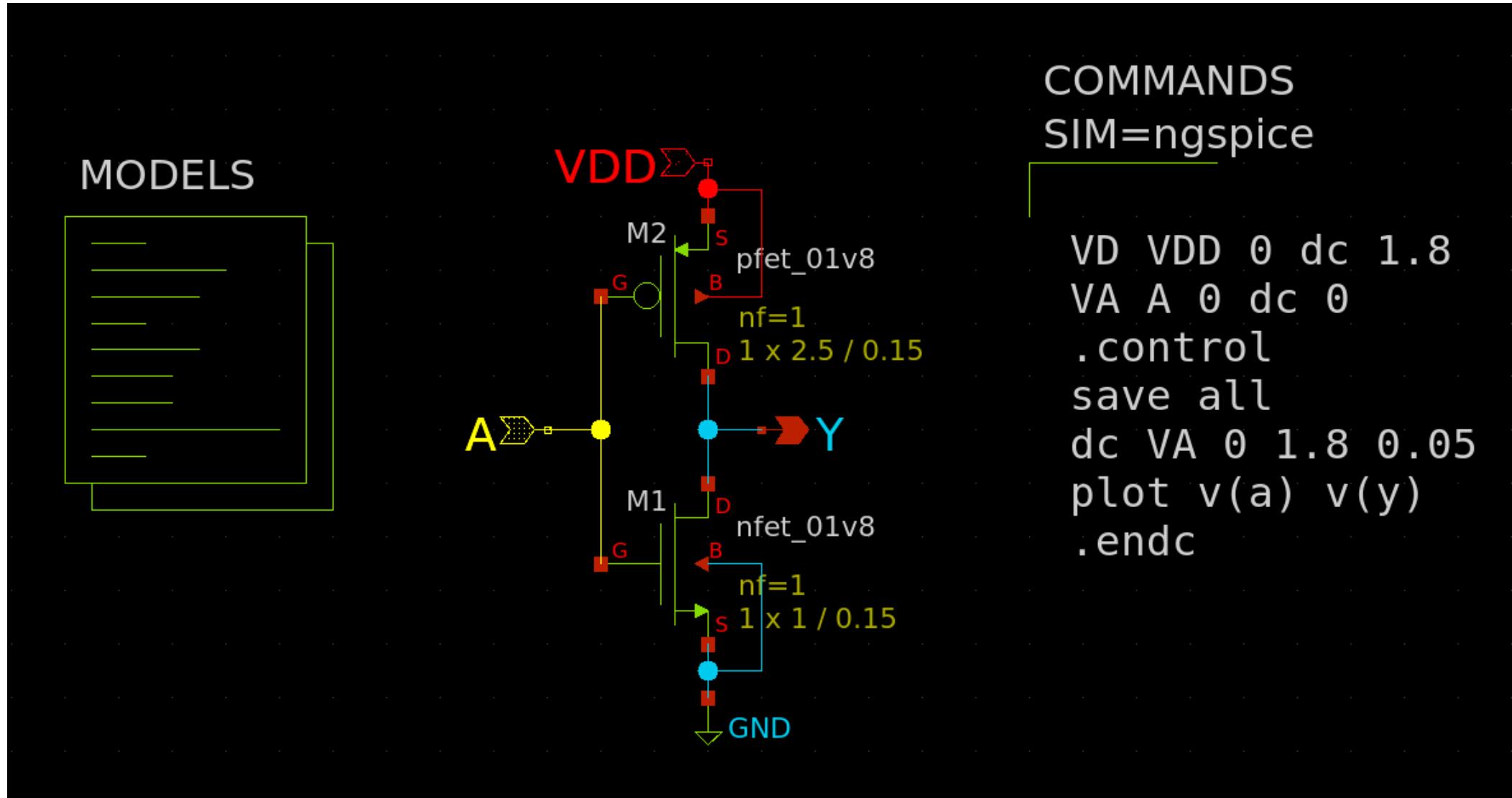
アナログLSIの設計フロー

どのような開発ツールを使用するか、どのような設計をするか

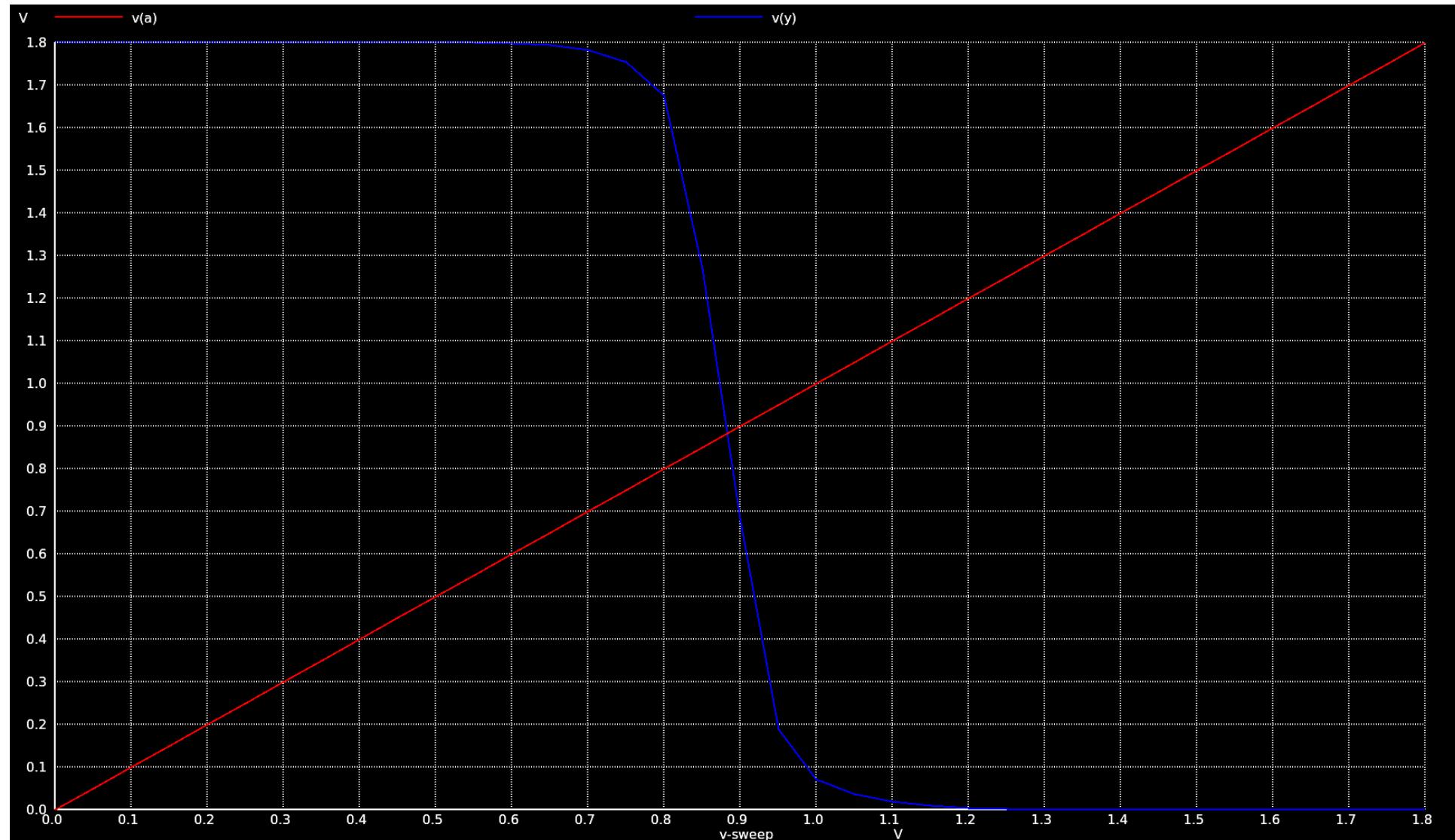
アナログLSIの設計フロー

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

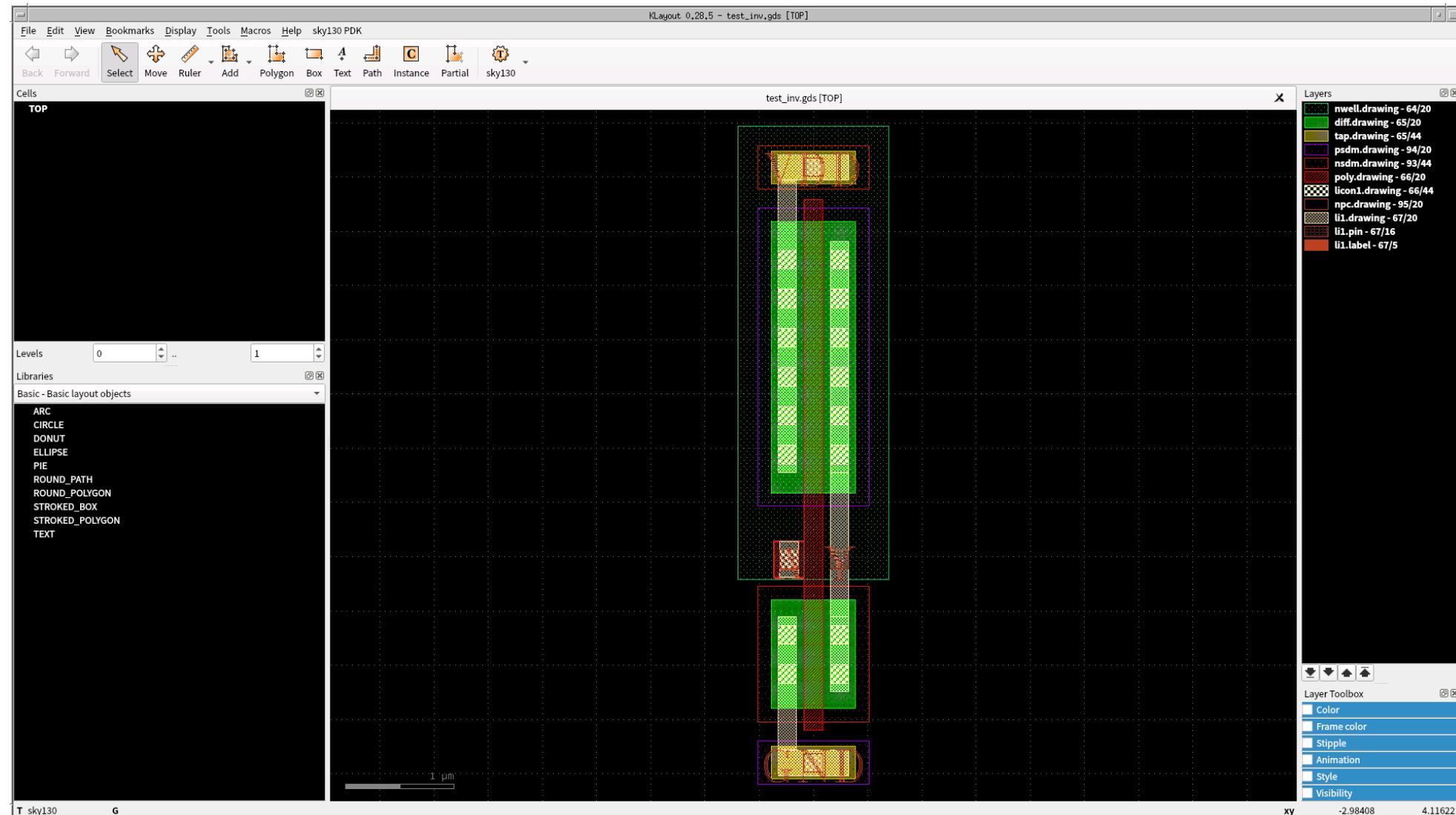
回路図(テストベンチ)を描いて…



論理検証をして…



レイアウトを描いて…



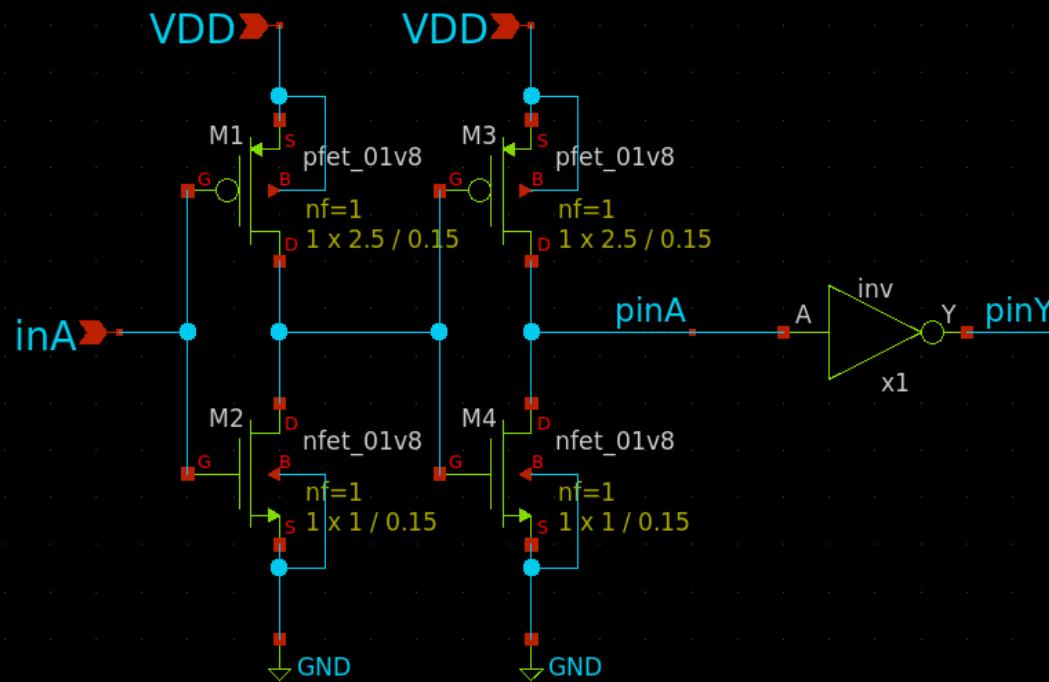
寄生成分を抽出して…

```
 Magic-PEX
File Edit View Search Terminal Help
Loading sky130A Device Generator Menu ...
Loading "/home/user/.klayout/macros/sky130_magic_pex.tcl" from command line.
Warning: Calma reading is not undoable! I hope that's OK.
Library written using GDS-II Release 6.0
Library name: LIB
Reading "TOP".
CIF file read warning: CIF style sky130(): units rescaled by factor of 5 / 1
Extracting TOP into TOP.ext:
exttosim finished.
exttospice finished.
exttospice finished.
* NGSPICE file created from TOP.ext - technology: sky130A

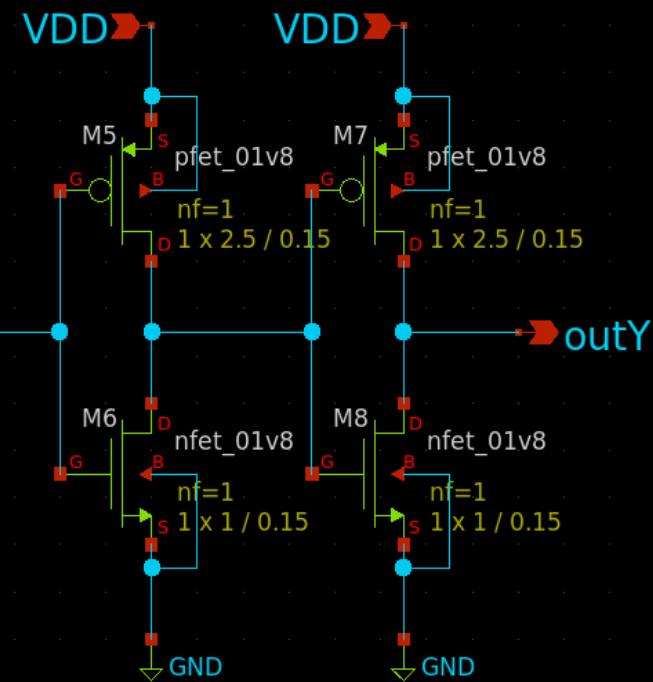
.subckt TOP A Y VDD GND
X0 Y A VDD VDD sky130_fd_pr_pfet_01v8 ad=7.5e+11p pd=5.6e+06u as=7.5e+11p ps=5.
6e+06u w=2.5e+06u l=180000u
X1 Y A GND GND sky130_fd_pr_nfet_01v8 ad=3e+11p pd=2.6e+06u as=3e+11p ps=2.6e+0
6u w=1e+06u l=180000u
C0 A Y 0.05fF
C1 VDD Y 0.17fF
C2 A VDD 0.18fF
.ends
```

テストベンチを作成して…

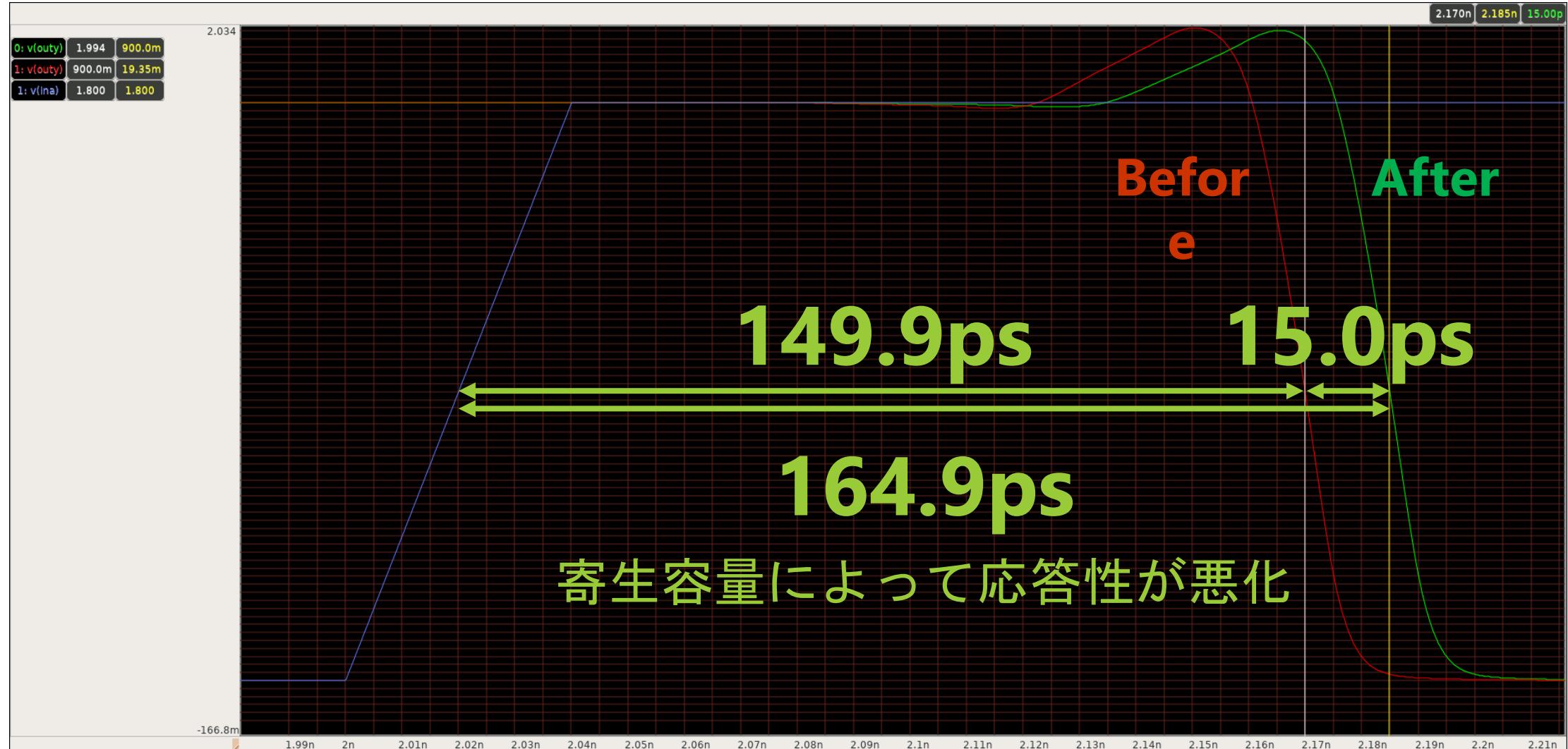
```
ngspice_
VA inA 0 pulse(0 1.8 0 40p 40p 1n 2n) dc 0
VD VDD 0 dc 1.8
.include ~/TOP_pex_extracted.spice
.control
.tran 1p 4n
.wrdata ~/inv_bench.txt v(ina) v(outy)
.write ~/inv_test_pex.raw
.endc
```



MODELS



ポストレイアウトシミュレーションをする



設計する前に…

- CMOSの特性を知りたい！
 - V_{gs} – I_{ds} カーブ
 - 動作点解析 (Operational Point)
 - V_{ds} – I_{ds} カーブ

$V_{gs} - I_{ds}$ カーブ / オン抵抗測定

PMOS/NMOSのオン抵抗を測定する ($L=0.15\mu m$, $W=1.0\mu m$, $V_{ds}=1.8V$)

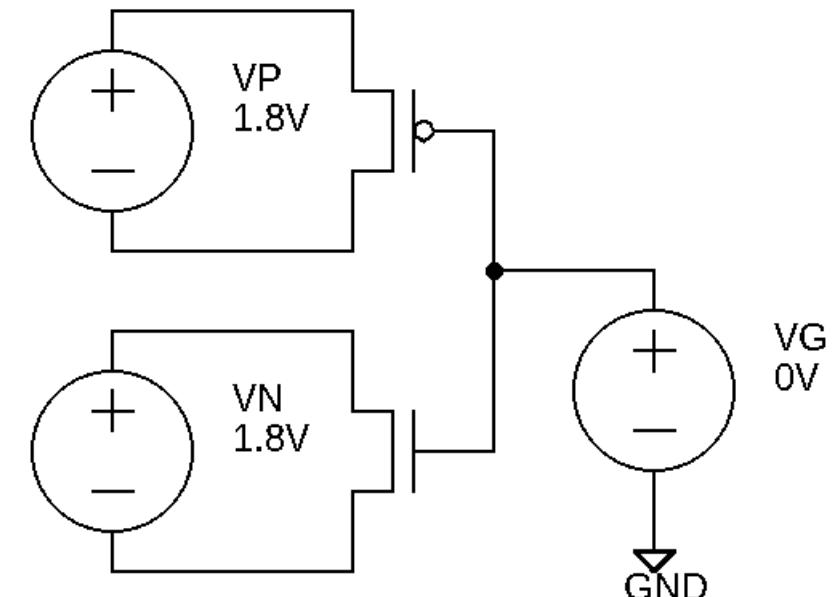
1. ドレイン—ソース間に電圧源 v_p ($V=1.8$), v_n ($V=1.8$)を置く
2. ゲートに電圧源 v_g ($V=0$)を置く
3. DC解析で v_g の電圧を $0V \rightarrow 1.8V$ 間で掃引
4. 電圧源 v_p , v_n の消費電流を見る

DC解析書式

dc [電圧源] [開始電圧] [終了電圧] [ステップ幅]

消費電流

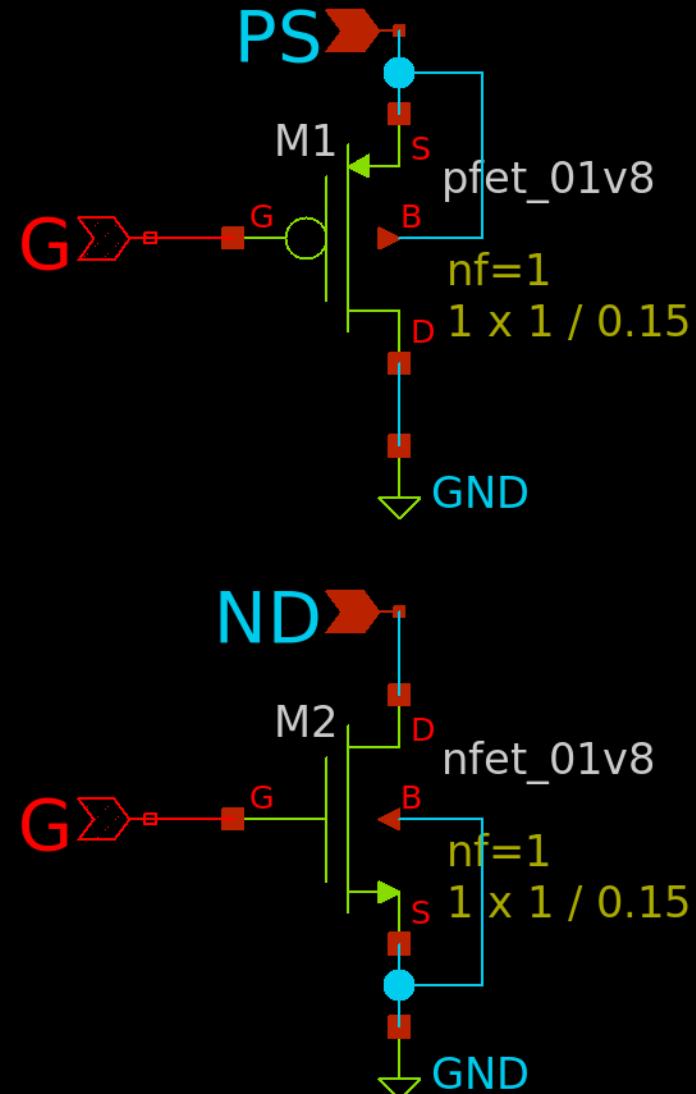
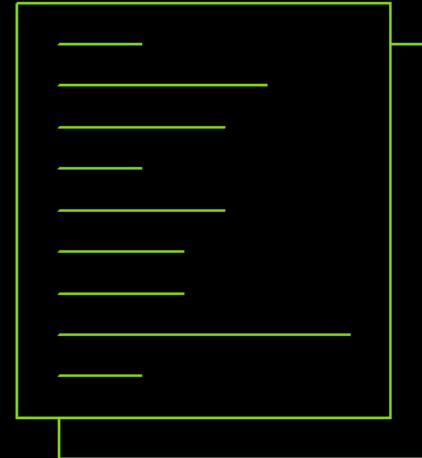
ngspice: $v_p\#branch$, $v_n\#branch$ または gaw: $i(v_p)$, $i(v_n)$



Xschem 構成例 (L=0.15μm)

vgsbench.sch

MODELS

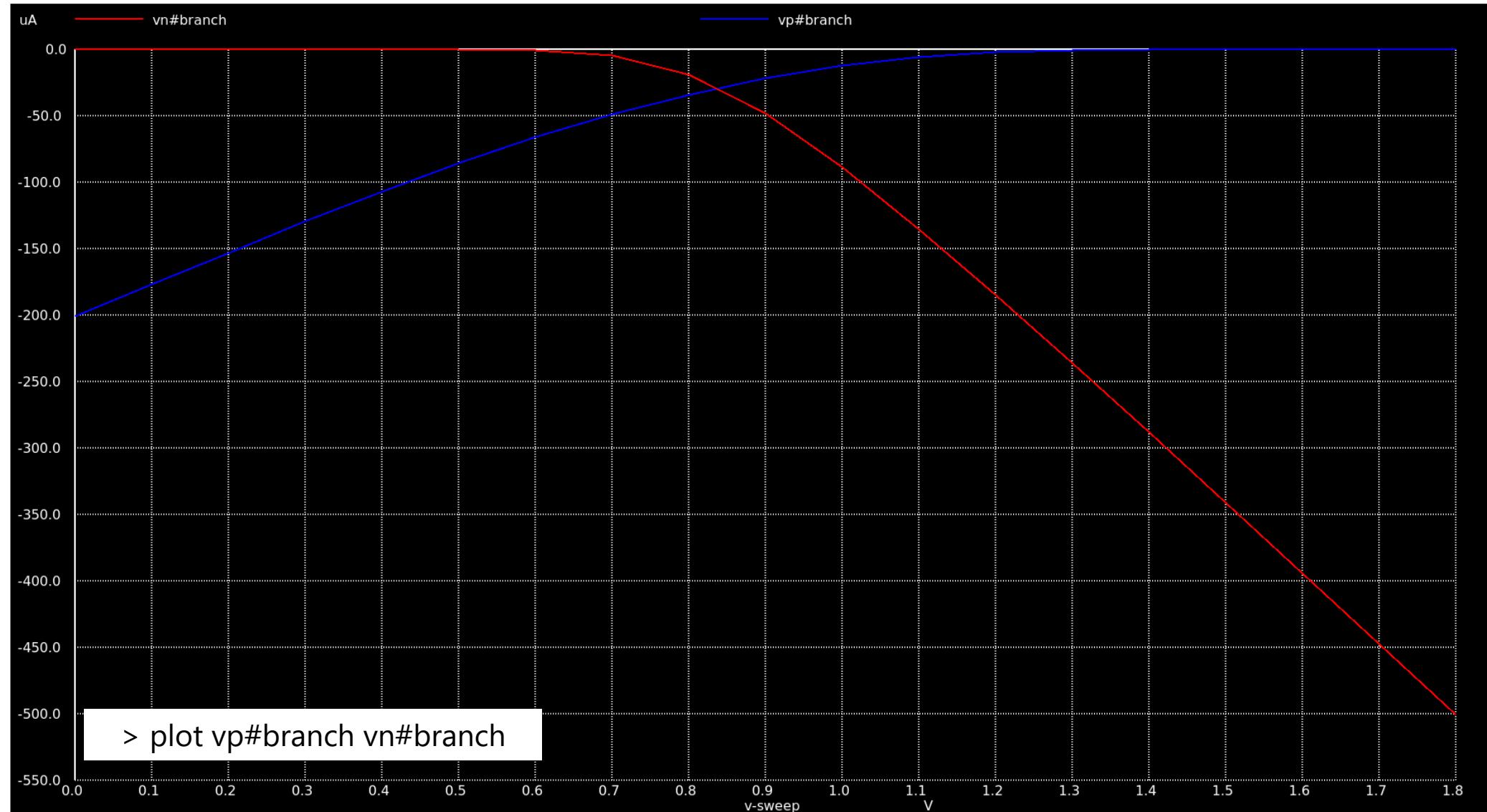


SPICEネットリストで
GNDは数字の0で書く

```
ngspice
VG G 0 dc 0
VP PS 0 dc 1.8
VN ND 0 dc 1.8
.control
save all
dc VG 0 1.8 0.1
write trbench.raw
.endc
```

ピン名は基本任意だが、
基準となるGND or 0を必ず入れる事
“VSS”は0に変換されない！

シミュレーション結果例



ngspiceの生データ出力方法

wrdataで特定のデータを抽出可能
ホームディレクトリ (/home/[ユーザ名]/) にvp.txt, vn.txtが生成される

vp.txt

0.00000000e+00	-2.00726126e-04
1.00000000e-01	-1.76783868e-04
2.00000000e-01	-1.52856174e-04
3.00000000e-01	-1.29370810e-04
4.00000000e-01	-1.06779404e-04
5.00000000e-01	-8.55349063e-05
6.00000000e-01	-6.60656564e-05
7.00000000e-01	-4.87492484e-05
8.00000000e-01	-3.38904626e-05
9.00000000e-01	-2.17083481e-05
1.00000000e+00	-1.23365952e-05
1.10000000e+00	-5.82743752e-06
1.20000000e+00	-2.07958988e-06
1.30000000e+00	-5.36969830e-07
1.40000000e+00	-1.19564554e-07
1.50000000e+00	-2.81003906e-08
1.60000000e+00	-6.92577344e-09
1.70000000e+00	-1.63614705e-09
1.80000000e+00	-3.19909099e-10

vn.txt

0.00000000e+00	-2.03659312e-12
1.00000000e-01	-4.80682161e-12
2.00000000e-01	-3.99689171e-11
3.00000000e-01	-4.84302376e-10
4.00000000e-01	-5.99945693e-09
5.00000000e-01	-7.04327370e-08
6.00000000e-01	-6.97912479e-07
7.00000000e-01	-4.74592920e-06
8.00000000e-01	-1.91451559e-05
9.00000000e-01	-4.83465908e-05
1.00000000e+00	-8.87674453e-05
1.10000000e+00	-1.35141610e-04
1.20000000e+00	-1.84593052e-04
1.30000000e+00	-2.35816015e-04
1.40000000e+00	-2.88115929e-04
1.50000000e+00	-3.41048569e-04
1.60000000e+00	-3.94303222e-04
1.70000000e+00	-4.47655558e-04
1.80000000e+00	-5.00941446e-04

```
ngspice
    VG G 0 dc 0
    VP PS 0 dc 1.8
    VN ND 0 dc 1.8
    .control
        dc VG 0 1.8 0.1
        wrdata ~/vp.txt vp#branch
        wrdata ~/vn.txt vn#branch
    .endc
```

オン抵抗

$L=0.15\mu m$, $W=1.0\mu m$ のときの静特性

PMOS: $0.0000000e+00 -2.00726126e-04$

ドレンソース間は1.8Vなので、 $1.8/2.01e-4 = 9.0k\Omega$

NMOS: $1.8000000e+00 -5.00941446e-04$

ドレンソース間は1.8Vなので、 $1.8/5.01e-4 = 3.6k\Omega$

オン抵抗と W は概ね逆比の関係がある → $P : N = 2.5 : 1$ でオン抵抗が揃う

- トランジスタサイズは通常、 W を変える

動作点解析 (OP)

opbench.sch

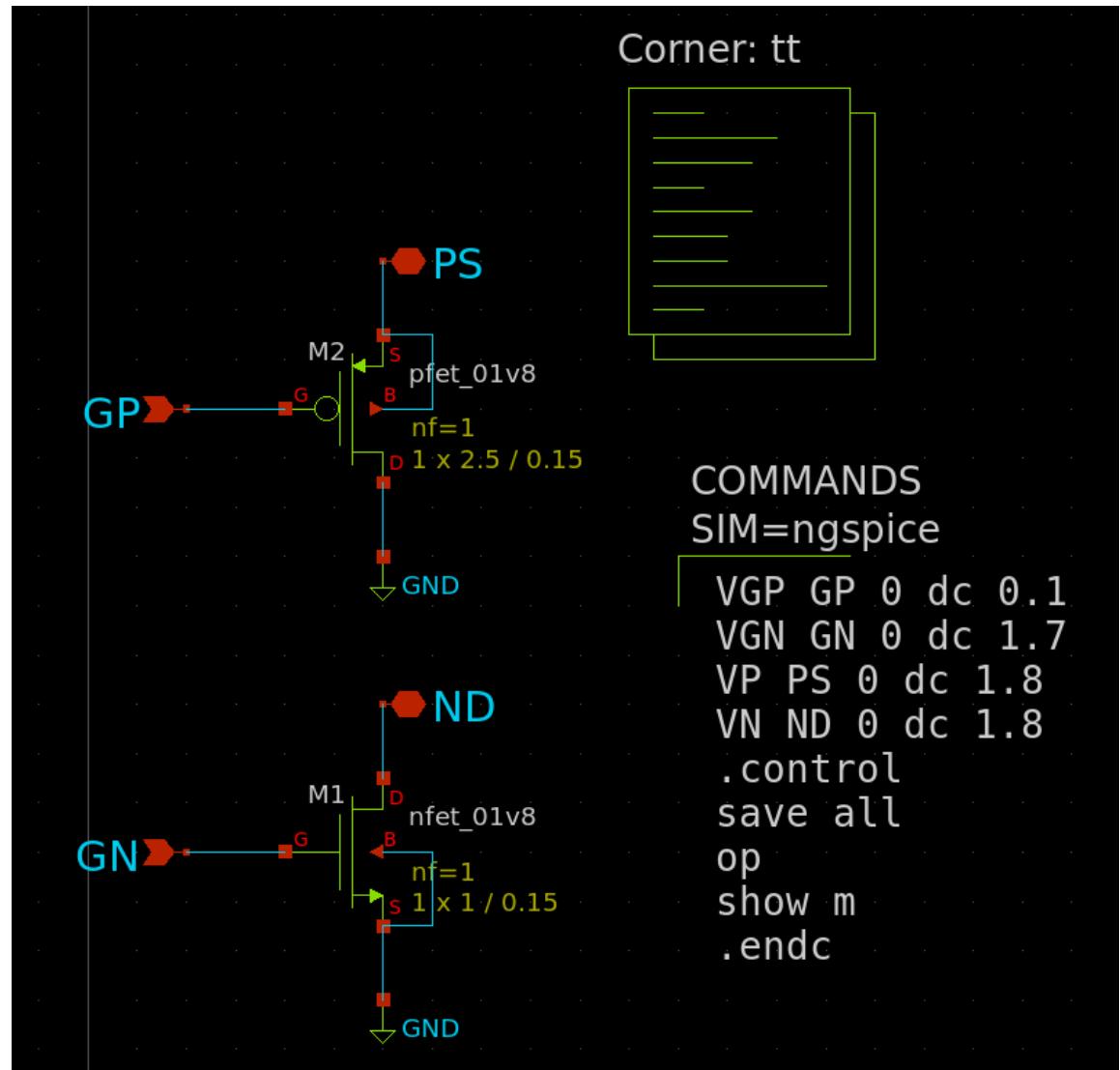
- 動作点解析のコマンドは **op**
- gm等を見る場合は電圧の条件に注意
 - オフ状態とオン状態とでは値が異なる

オフ状態

gm	3.79481e-08	7.63463e-11
gds	4.38991e-09	2.43751e-12
vdsat	0.0447906	0.0430537
vth	0.561294	0.769268

オン状態

gm	0.00060858	0.000533543
gds	9.80226e-05	4.93687e-05
vdsat	0.755181	0.343415
vth	0.561319	0.769291



トランスコンダクタンス gm について

- CMOSについて勉強すると、次のような式を見かけると思います。

$$I_{DS} = \frac{W}{L} \mu_n C_{ox} \left[(V_{GS} - V_{th}) V_{DS} - \frac{1}{2} V_{DS}^2 \right] \quad \text{in 線形領域 } (V_{DS} \leq V_{GS} - V_{th})$$

$$I_{DS} = \frac{W}{L} \mu_n C_{ox} \left[\frac{1}{2} (V_{GS} - V_{th})^2 \right] \quad \text{in 飽和領域 } (V_{DS} \geq V_{GS} - V_{th})$$

- gmは上の式を微分したパラメータなので、つまり、

$$g_m = \frac{W}{L} \mu_n C_{ox} V_{DS} \quad \text{in 線形領域 } (V_{DS} \leq V_{GS} - V_{th})$$

$$g_m = \frac{W}{L} \mu_n C_{ox} (V_{GS} - V_{th}) \quad \text{in 飽和領域 } (V_{DS} \geq V_{GS} - V_{th})$$

$$\rightarrow I_{DS} = \frac{1}{2} \times g_m \times (V_{GS} - V_{th}) \quad \text{in 飽和領域}$$

show m

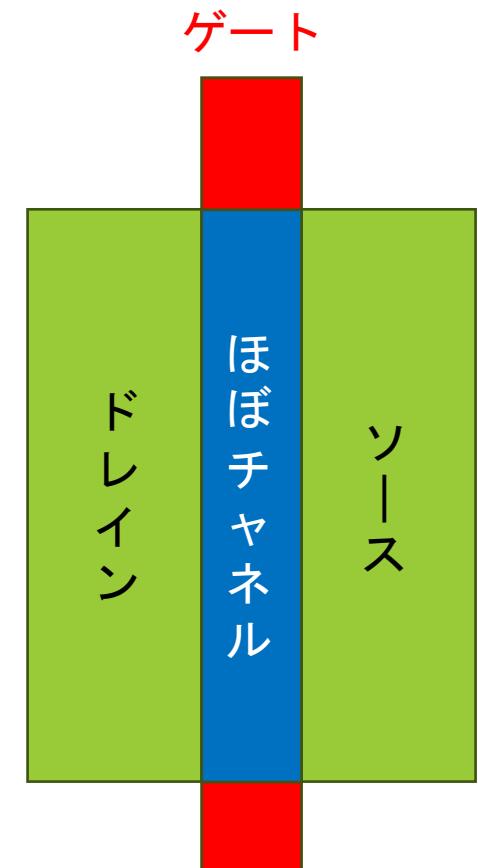
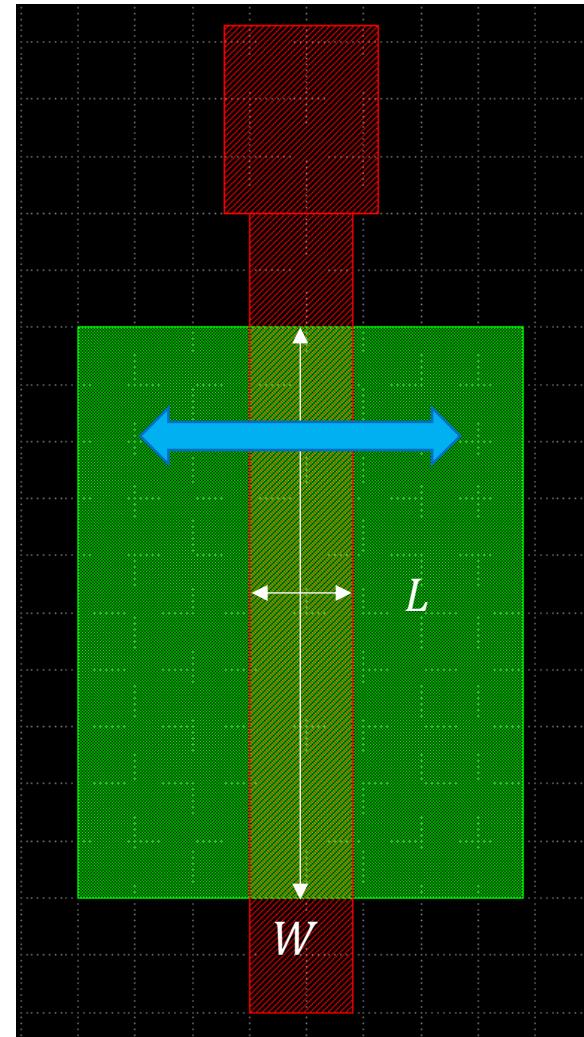
長いのでテキストへの保存をオススメ

- show m > ~/test.txt など

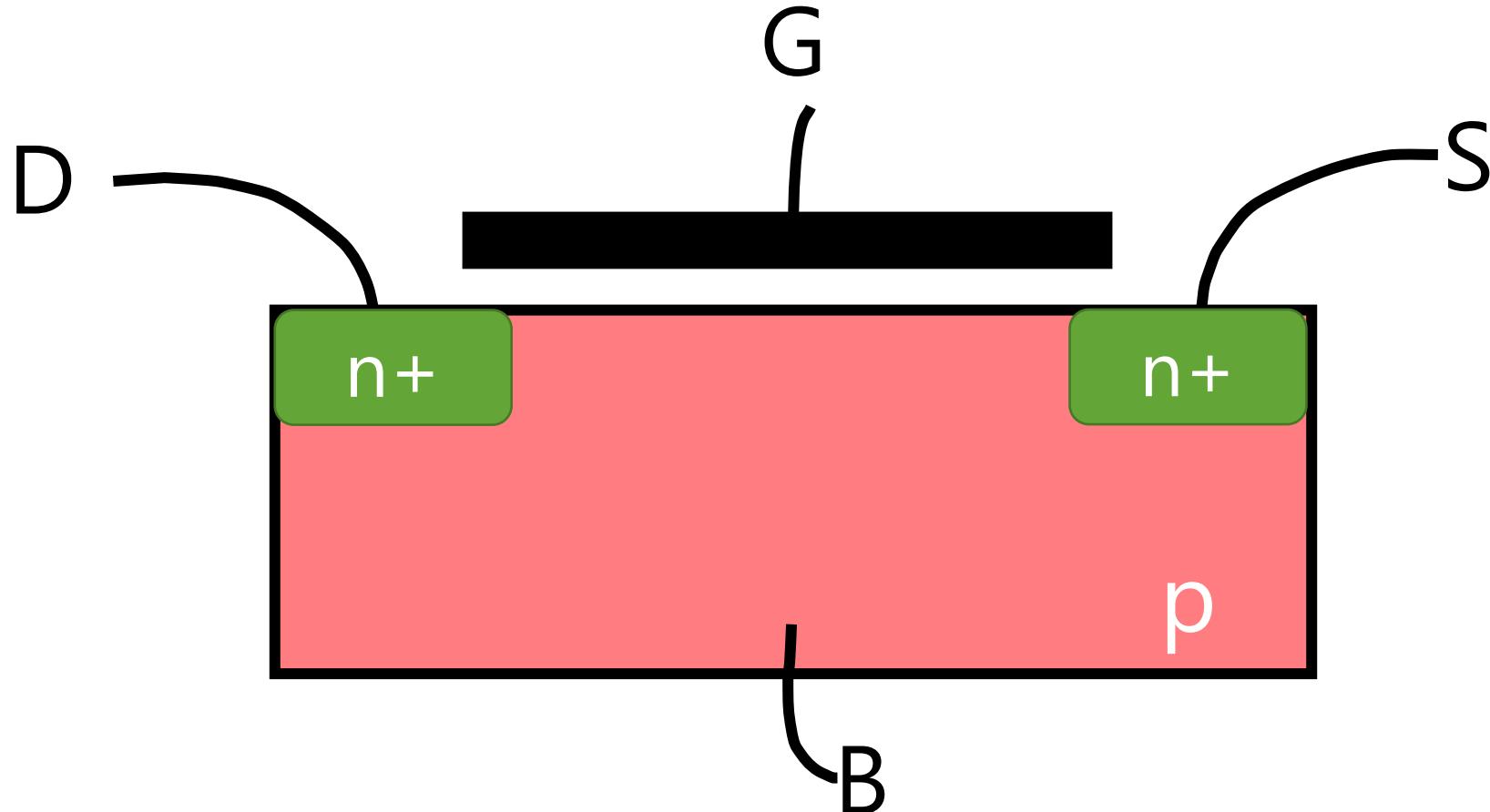
	m.xm2.msky130_fd_pr__	m.xm1.msky130_fd_pr__
device	m.xm2.msky130_fd_pr__	m.xm1.msky130_fd_pr__
model	xm2:sky130_fd_pr_pfe	xm1:sky130_fd_pr_nfe
l	1.5e-07	1.5e-07
w	1e-06	1e-06
m	1	1
nf	1	1
sa	0	0
sb	0	0
sd	0	0
sca	0	0
scb	0	0
scc	0	0
sc	0	0
min	0	0
ad	2.9e-13	2.9e-13
as	2.9e-13	2.9e-13
pd	2.58e-06	2.58e-06
ps	2.58e-06	2.58e-06
nrd	0.29	0.29
nrs	0.29	0.29
off	0	0
rbdb	50	50
rbsb	50	50
rbpb	50	50
rbps	50	50
rbpd	50	50
delvto	0	0
mulu0	1	1
xgw	0	0
ngcon	1	1
trnqsmod	0	0
acnqsmod	0	0
rbodymod	1	1
rgatemod	0	0
geomod	0	0
rgeomod	0	0
gmbs	4.87021e-05	1.57575e-12
am	0.000238192	6.00366e-12

トランジスタのパラメータ

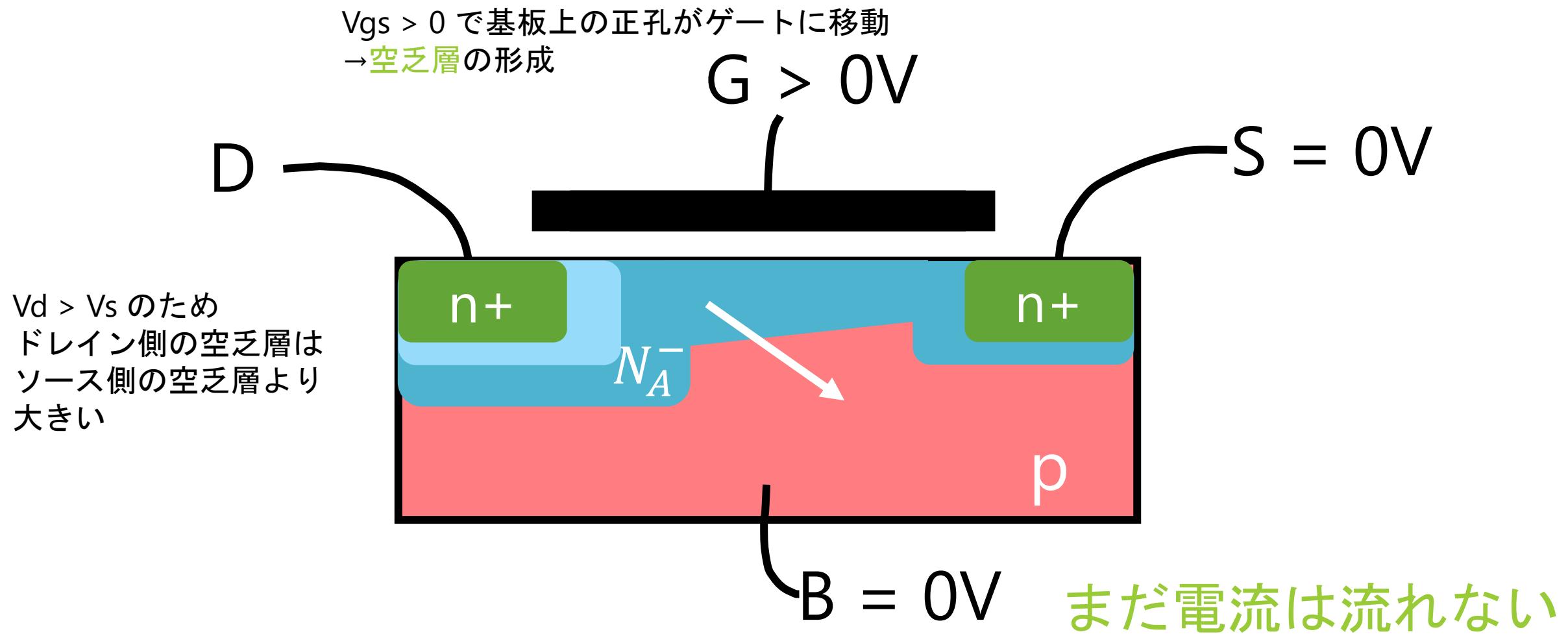
- 基本はWとLを変える
 - W : チャネル幅
 - L : チャネル長（※実際のチャネル長はちょっと短い）
- **ゲートと拡散層の重なった部分がトランジスタ本体**
ゲートに充分な電位を与えると**チャネル**が形成
- m, nf, mf などはゲートの本数 (レイアウト解説にて後述)



nMOS 簡単な模式図



nMOS 簡単な模式図



nMOS 簡単な模式図

$V_{gs} \gg 0$ で伝導電子が多数キャリアとなる

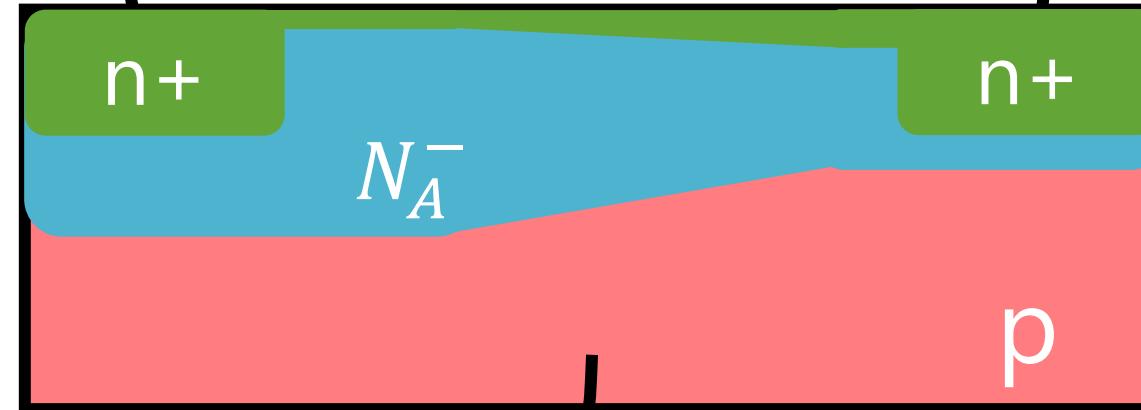
→ 反転層（チャネル）の形成 (V_{th})

$$G \gg 0V$$

D

$$S = 0V$$

反転層は空乏層が
薄いと形成されやすく、
厚いと形成されにくい



$$B = 0V$$

電流が流れる

nMOS 簡単な模式図

$V_{gs} \gg 0$ で伝導電子が多数キャリアとなる

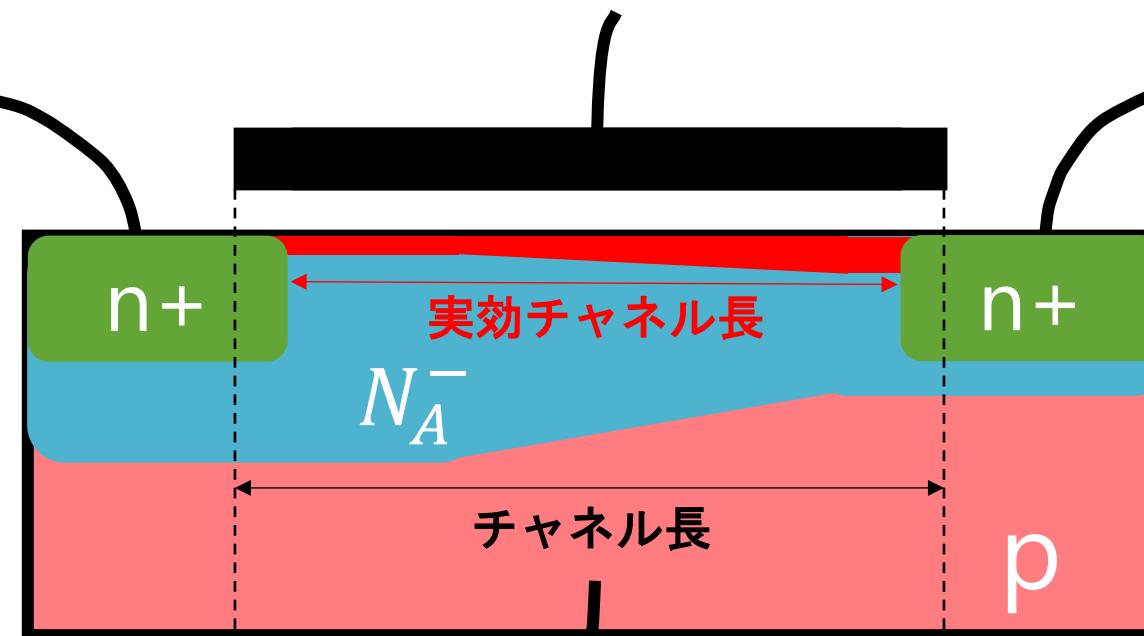
→ 反転層（チャネル）の形成 (V_{th})

$$G \gg 0V$$

D

$$S = 0V$$

反転層は空乏層が
薄いと形成されやすく、
厚いと形成されにくい

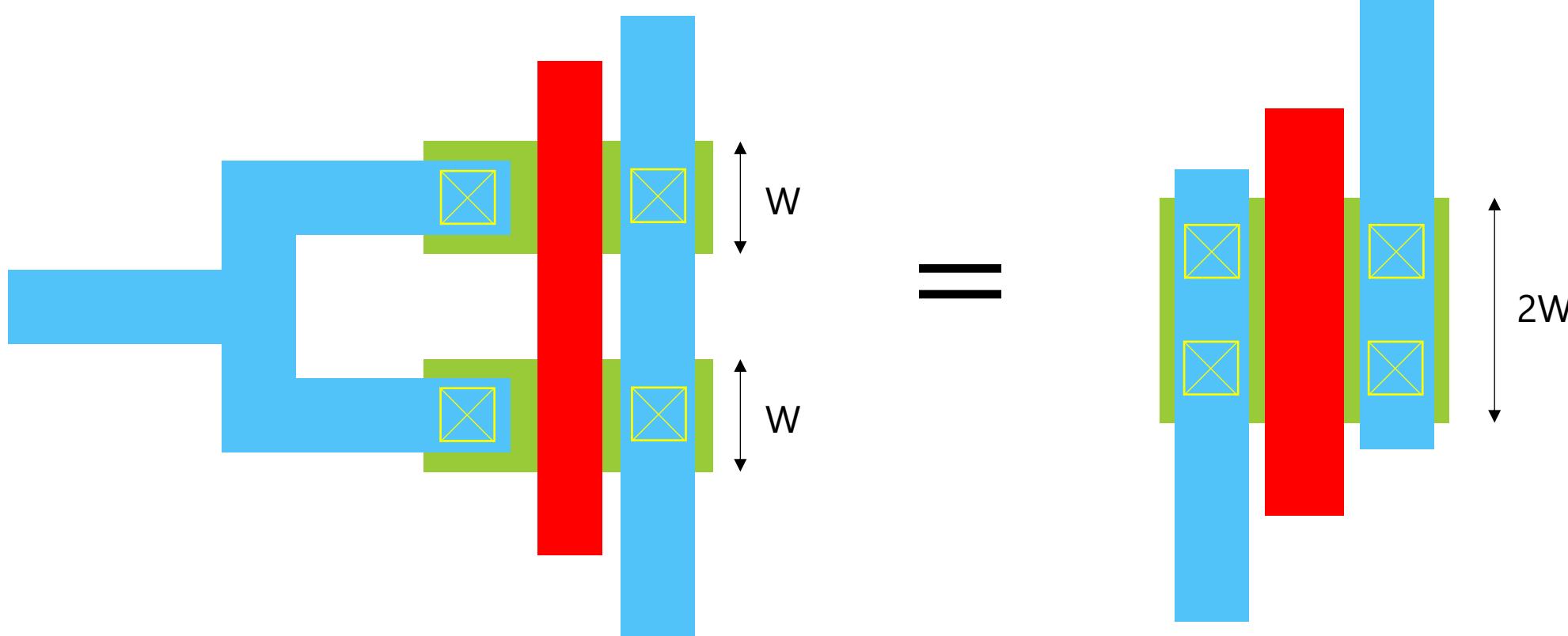


$$B = 0V$$

電流が流れる

チャネル幅 Wを変えるとどうなる？

- チャネル幅を大きくとると電流の通り道が広くなるので、ドレンソース間の電流は増える
- チャネル長が同じトランジスタでは、**チャネル幅とドレンソース間電流は比例の関係**



チャネル長 Lを変えるとどうなる？

- **短チャネル効果**による影響で、特性が全く変化する。
チャネル長Lが短くなると基本は I_{ds} が増加するが、**短チャネル効果**も発生
- DIBL (Drain Induced Barrier Lowering)により、 V_{ds} の増加に対して V_{th} が低下
- Subthreshold swingの増加により、弱反転領域におけるパンチスルーガが増加
 - オフ状態でのリーク電流が増え、 V_{gs} の増加幅に対する I_{ds} の増加幅が鈍くなる →スイッチング特性劣化
- チャネル長変調効果により、 V_{ds} 変動幅に対する I_{ds} 変動幅が増え、**飽和状態にならない**

V_{ds} – I_{ds} カーブ

- NMOSのV_{ds} – I_{ds}カーブを取得し、チャネル長変調効果の影響を見てみる

• 現在製作中！

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

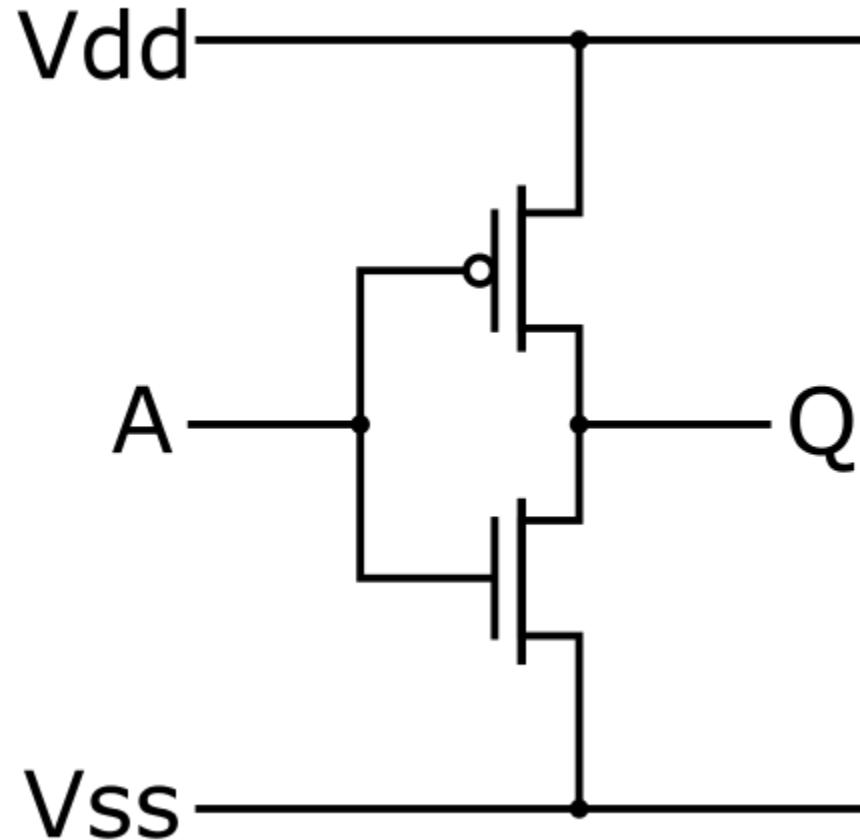
アナログLSIの設計フロー

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
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)

CMOSインバータの設計

- トポロジ検証
- CMOSインバータの性能とは？
- エルモア遅延モデル
 - 入出力条件を調べる
- 遅延時間シミュレーション
- measの使い方
- シミュレーションモデルの変更

CMOSインバータ



入力A	出力Q
L	H
H	L

↓

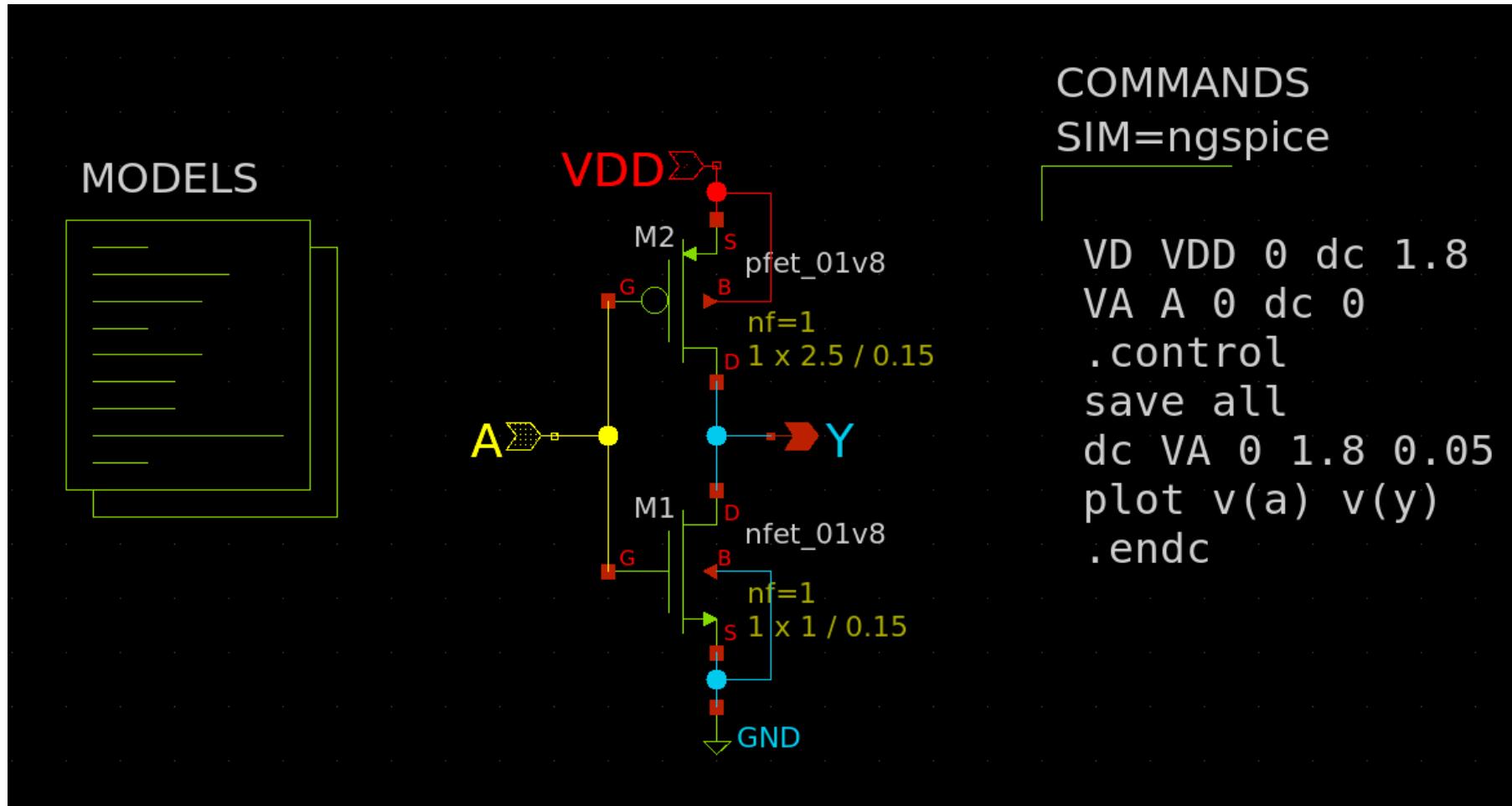
入力A	出力Q
Vss	Vdd
Vdd	Vss

論理検証（トポロジ検証）

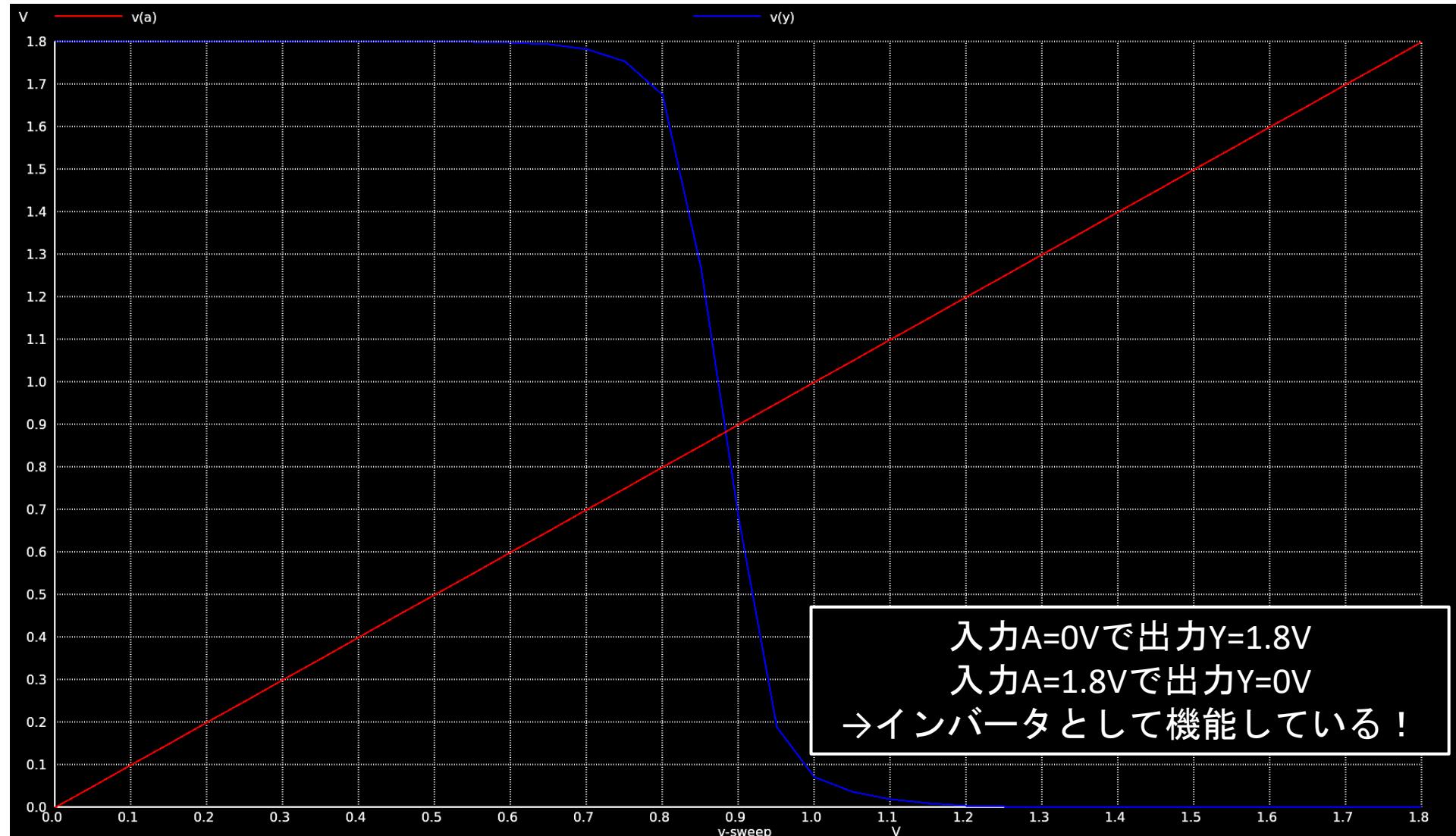
- ・ インバータでは入力に対して反転した出力が確認できれば良い
 - ・ 入力 0 V → 出力 Vdd V, 入力 Vdd V → 出力 0 V
 - ・ DC解析でCMOSインバータの動作を見てみる
- ・ DC解析 電圧を変動させたときの定常応答
- ・ tran解析 時間を変動させたときの過渡応答
- ・ AC解析 周波数を変動させたときの定常応答

Xschem 構成例 (L=0.15μm)

inv_dc.sch



シミュレーション結果



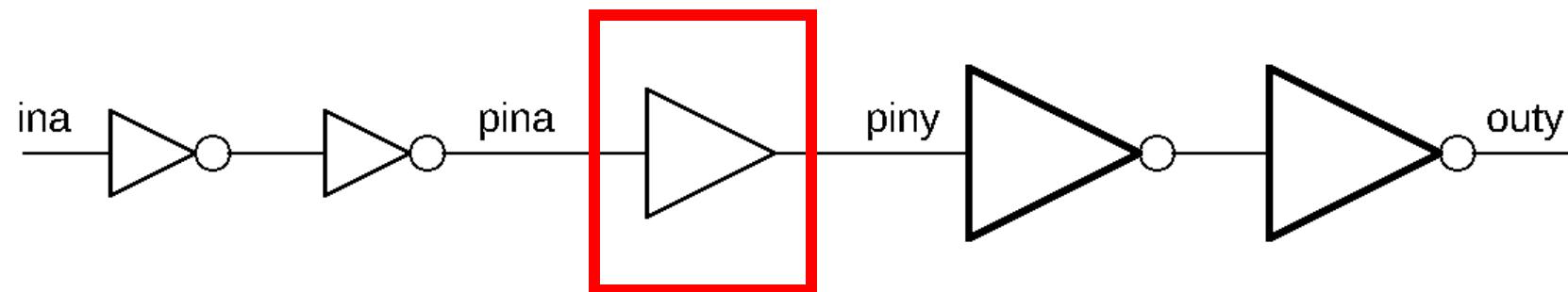
CMOSインバータの「性能」とは？

- トポロジは決定したが、PMOSとNMOSのサイズが決定していない → どうやって決めるの？
- CMOSインバータの性能
 - ドライバビリティ（オン抵抗の大きさ）
 - ファンアウト (F.O.)
 - 遅延時間**（立ち上がり遅延・立ち下がり遅延）

ファンアウト： インバータ／バッファの遅延時間

ファンアウトについて考える

$$P=2.5\mu / N=1\mu$$



$$P=25\mu / N=10\mu$$

ina - outy 間の遅延時間を測定

- 初段インバータサイズは $w_p = 2.5 \mu\text{m}$, $w_n = 1 \mu\text{m}$
- 出力段の負荷容量は入力容量の10倍
- 中間にバッファ（インバータ×2）を置く **このバッファのFOを考える**
- ina から outy までの遅延時間の最小化をめざす

バッファ遅延時間

6段のインバータチェインをシミュレーション

時間対電圧の波形を見たいのでトランジエント解析を行う

トランジエント解析書式

tran [最小ステップ幅] [解析時間]

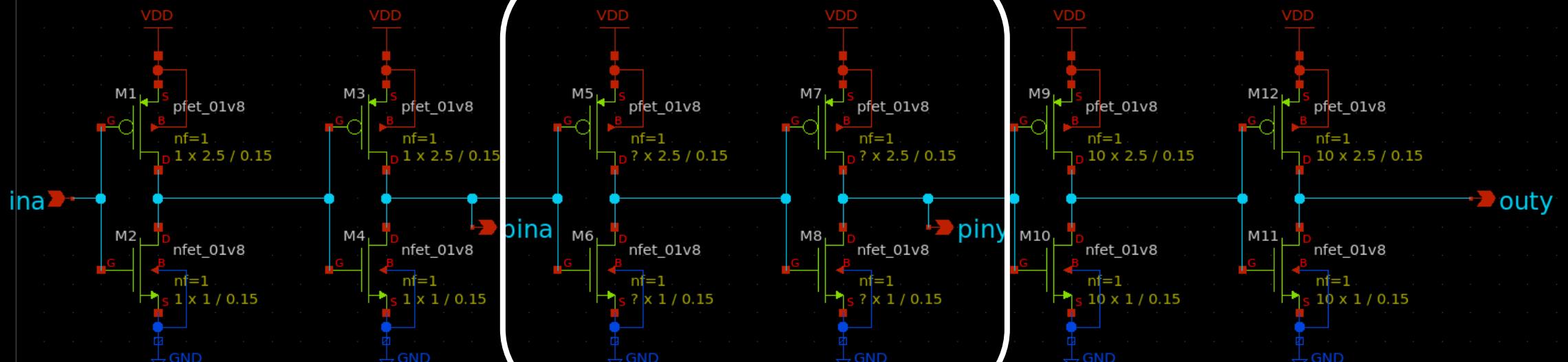
Xschem 構成例 (L=0.15μm)

6inv_test.sch

COMMANDS
SIM=ngspice

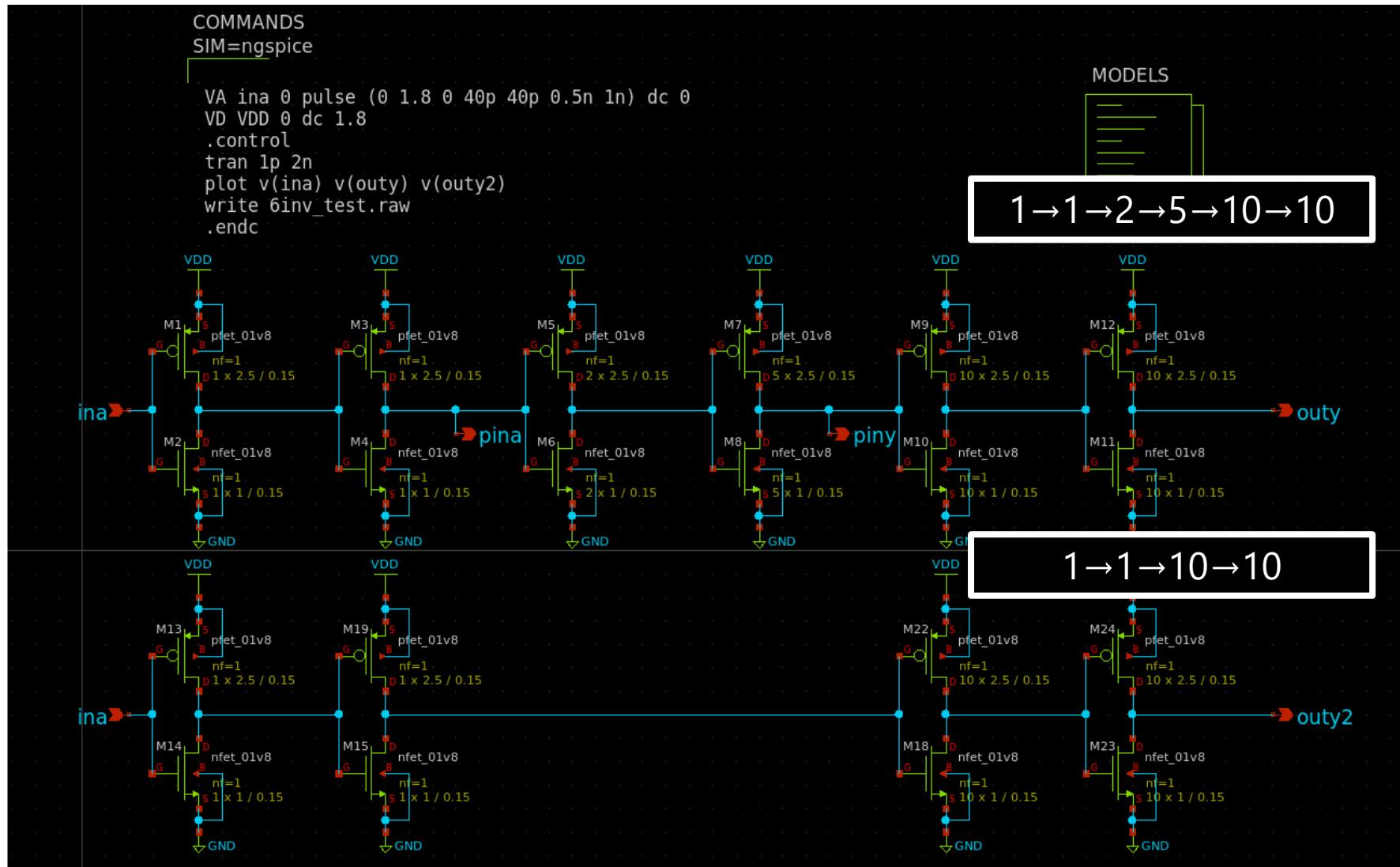
```
VA ina 0 pulse (0 1.8 0 40p 40p 0.5n 1n) dc 0
VD VDD 0 dc 1.8
.control
tran 1p 2n
plot v(ina) v(outy) v(outy)
write 6inv_test.raw
.endc
```

こここのFOをどうする?

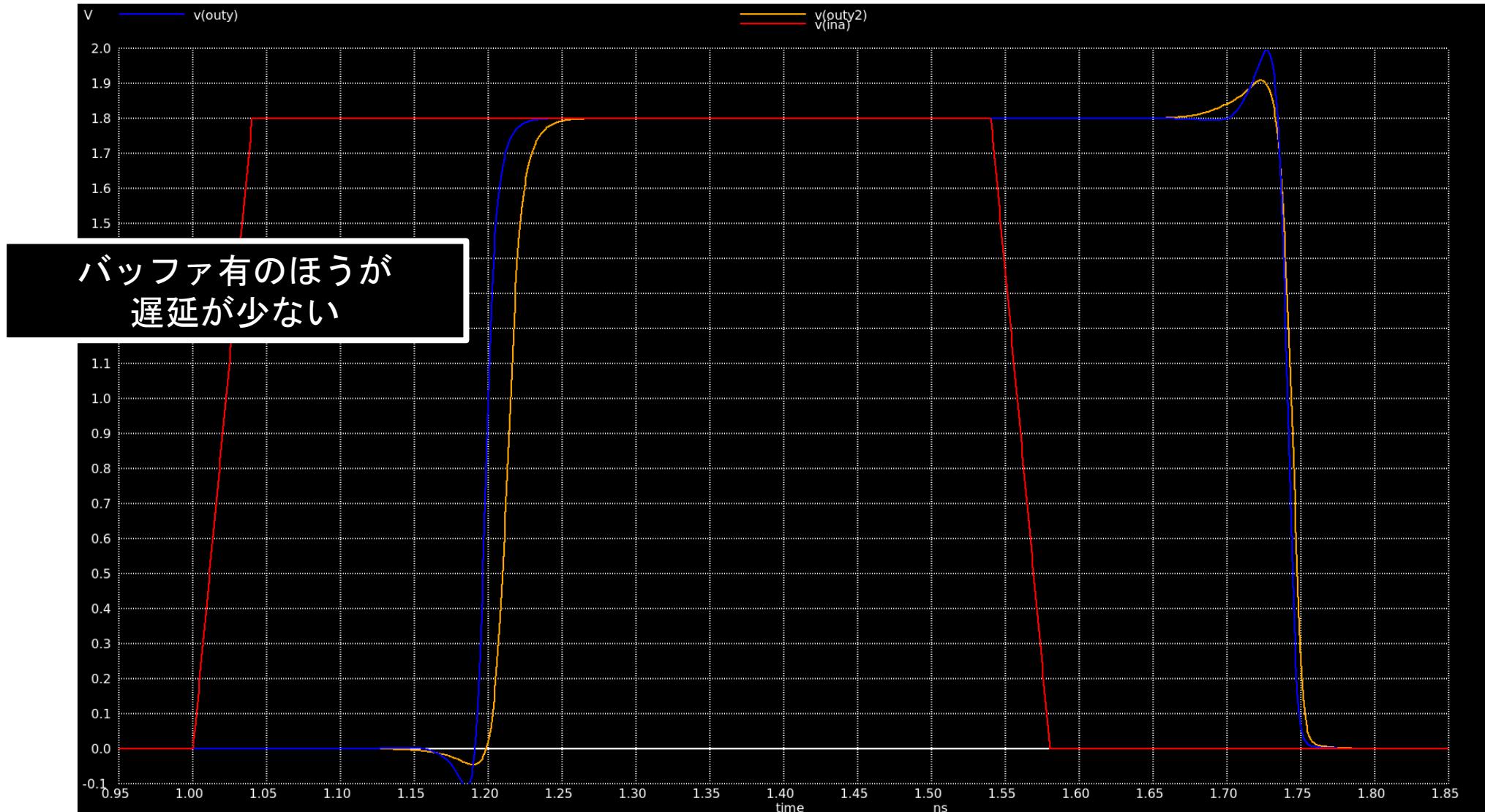


Xschem 構成例 (L=0.15μm)

6inv_test2.sch



シミュレーション結果



参考：スタンダードセルにおけるバッファの構成

buf_2 1→2

buf_4 1→4

buf_8 3→8

clkbuf8 2→8

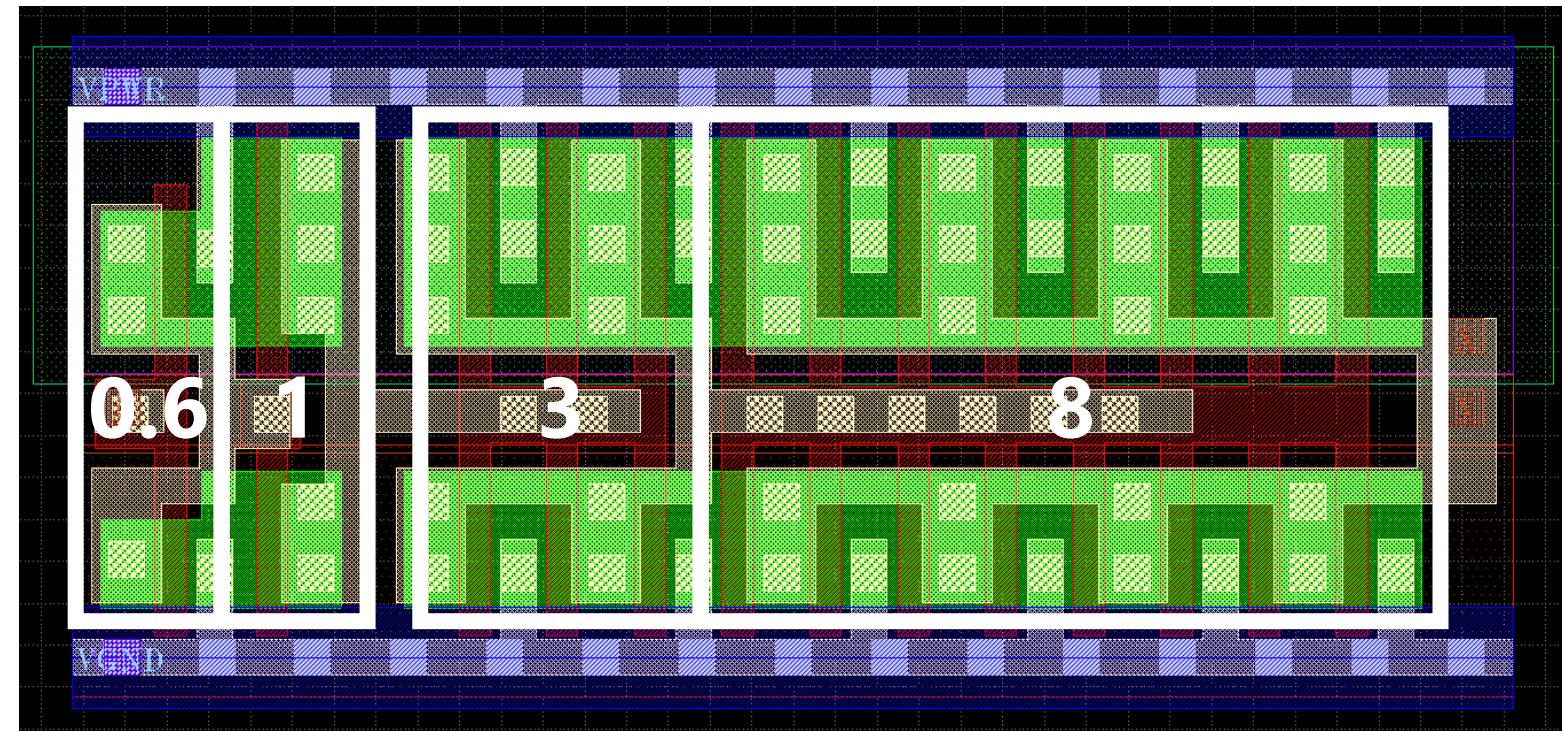
bufbuf8 0.6→1→3→8

buf_12 4→12

buf_16 6→16

clkbuf16 4→16

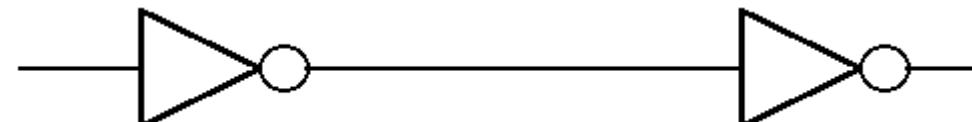
bufbuf16 1→3→6→16



遅延時間とは？

- 遅延時間 = **寄生容量の充放電時間** (入力ゲート容量+出力ドレイン容量 & 出力電流)
- エルモア遅延モデルでは、以下のような近似式で遅延時間を計算
- $t_{PHL} = 0.75 \times (C_{dd.n} + C_{dd.p} + C_{gg.n} + C_{gg.p})R_n$
入力立ち上がり・出力立ち下がり遅延
- $t_{PLH} = 0.75 \times (C_{dd.n} + C_{dd.p} + C_{gg.n} + C_{gg.p})R_p$
入力立ち下がり・出力立ち上がり遅延
- C_{dd} = 出力側のドレイン容量、 C_{gg} = 入力側のゲート容量 R =出力側のオン抵抗

$R_n = R_p$ なら立ち上がり遅延と立ち下がり遅延は一致する？



Cdd, Cgg はop解析でシミュレーション可能

ネットでMOSFETの入力容量(C_{iss})を検索すると下記の式がてくる

$$C_{iss} = C_{gs} + C_{gd} = \text{ゲートソース間容量} + \text{ゲートドレイン間容量}$$

LSIのCMOSでは赤ゲートボディ間容量も考慮する必要がある

$$C_{gg} = |C_{gs} + C_{gd} + C_{gb}|$$

- ゲート容量は C_{gg} というパラメータで表される
- 出力容量=ドレイン容量 $C_{dd} = |C_{dg} + C_{ds} + C_{db}|$
 - C_{gd}, C_{dg} はどちらもドレイン・ゲート間の寄生容量を表しているが、
 C_{dg} はドレイン電圧が変化した時に影響する寄生容量である一方、
 C_{gd} は ゲート電圧が変化した時に影響する寄生容量。 普通は $C_{dg} \neq C_{gd}$

入力容量 / 出力容量を調べる時は
OP解析で C_{gg} / C_{dd} を見る

過渡応答、遅延時間

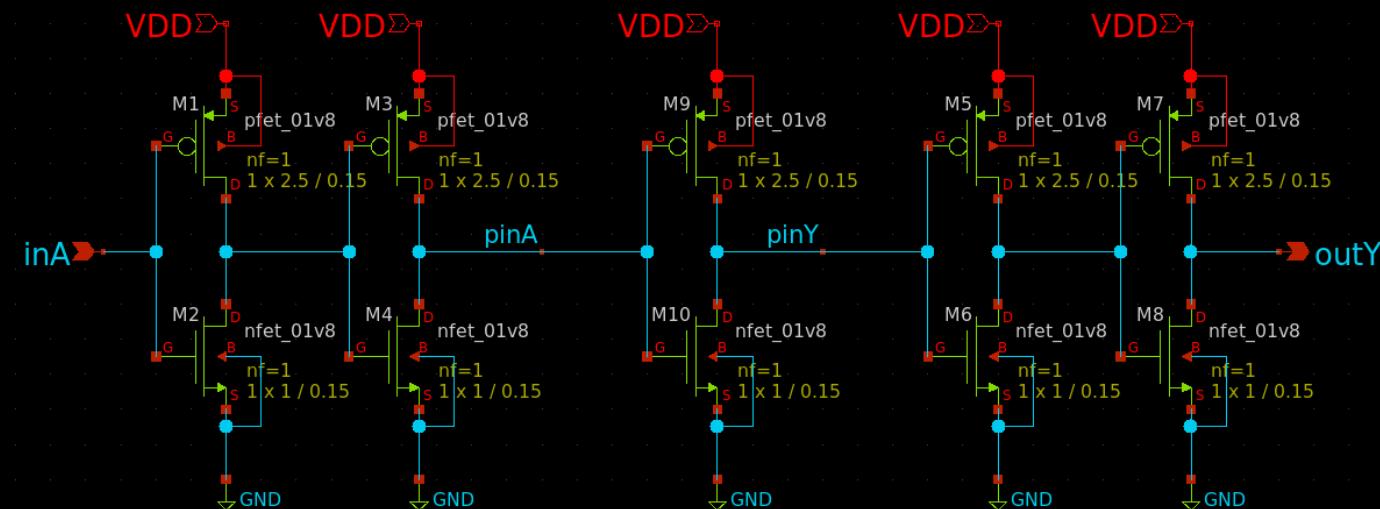
- 遅延時間を見たい場合はtran解析を用いる
- **ロジックでは入力・出力段に負荷を設定しないと正しい過渡応答（遅延）が見れない**
今回はFO=1のインバータを設計するので、同じサイズのインバータを接続

テストベンチ例

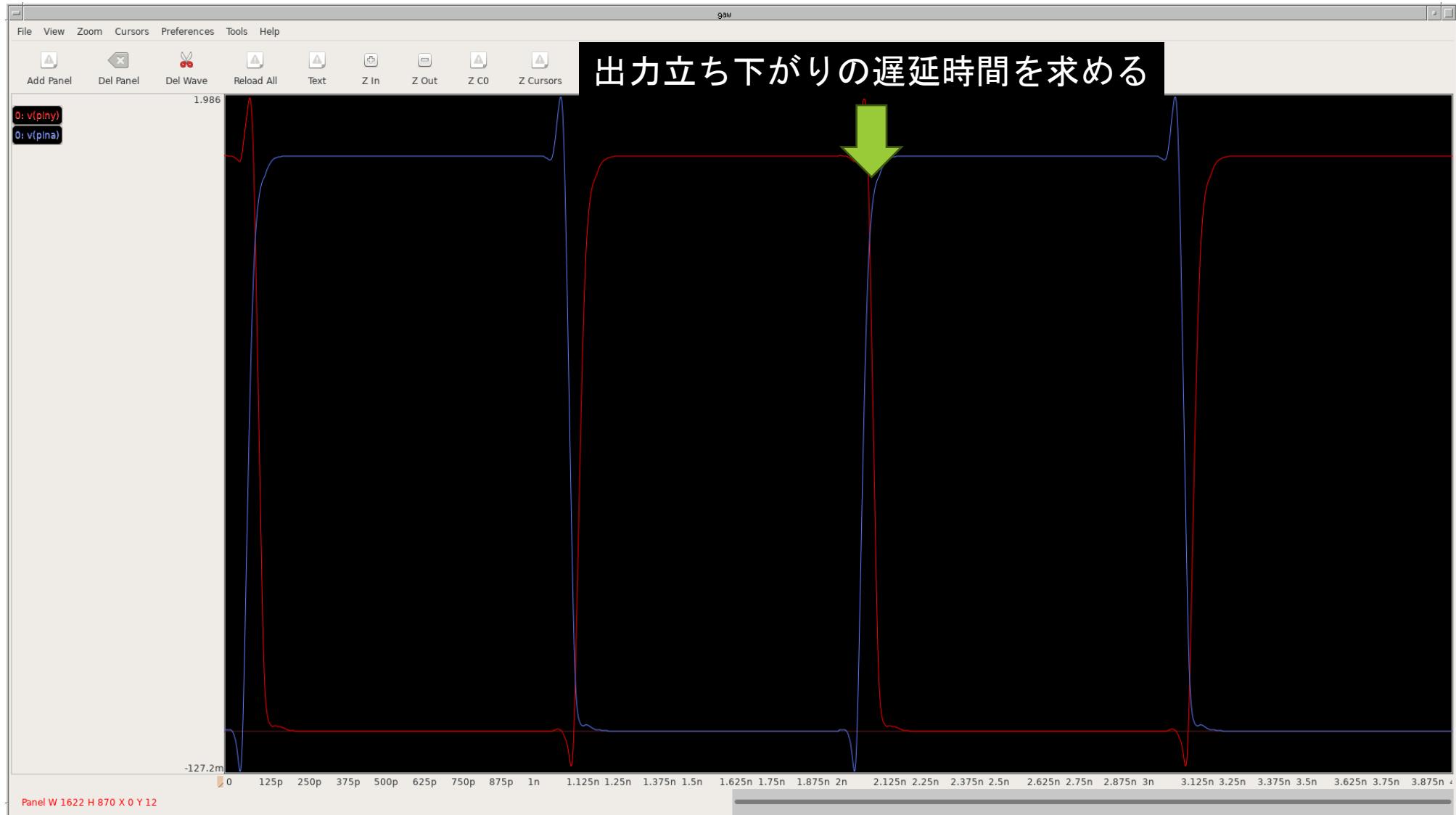
inv_meas.sch

```
ngspice
VA inA 0 pulse(0 1.8 0 40p 40p 1n 2n) dc 0
VD VDD 0 dc 1.8
.control
tran 1p 4n
meas tran delayA1 find time WHEN v(pinA)=0.9 RISE=2
meas tran delayY1 find time WHEN v(pinY)=0.9 FALL=2
let delay1 = delayY1 - delayA1
echo $&delay1
.endc
```

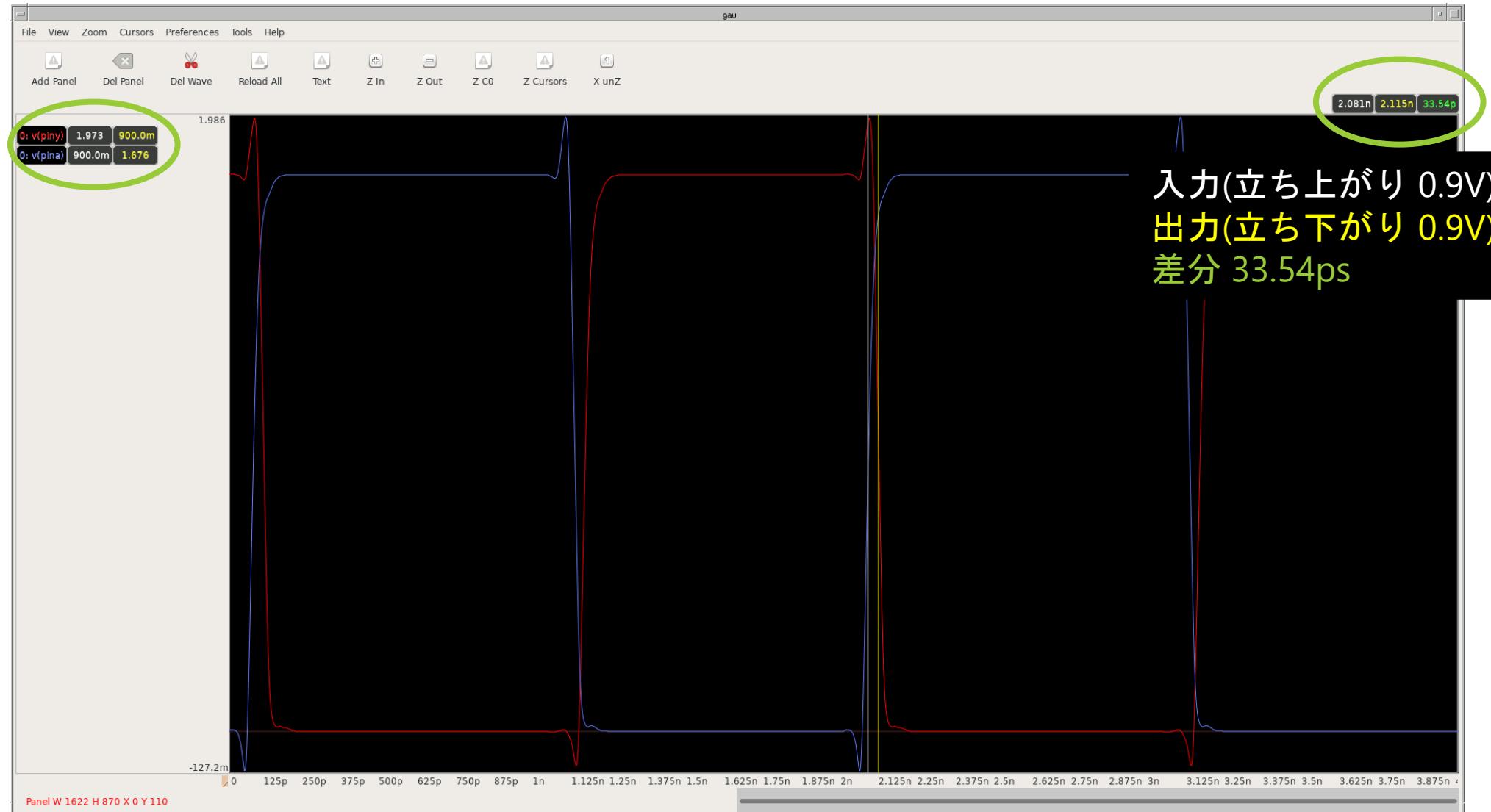
MODELS



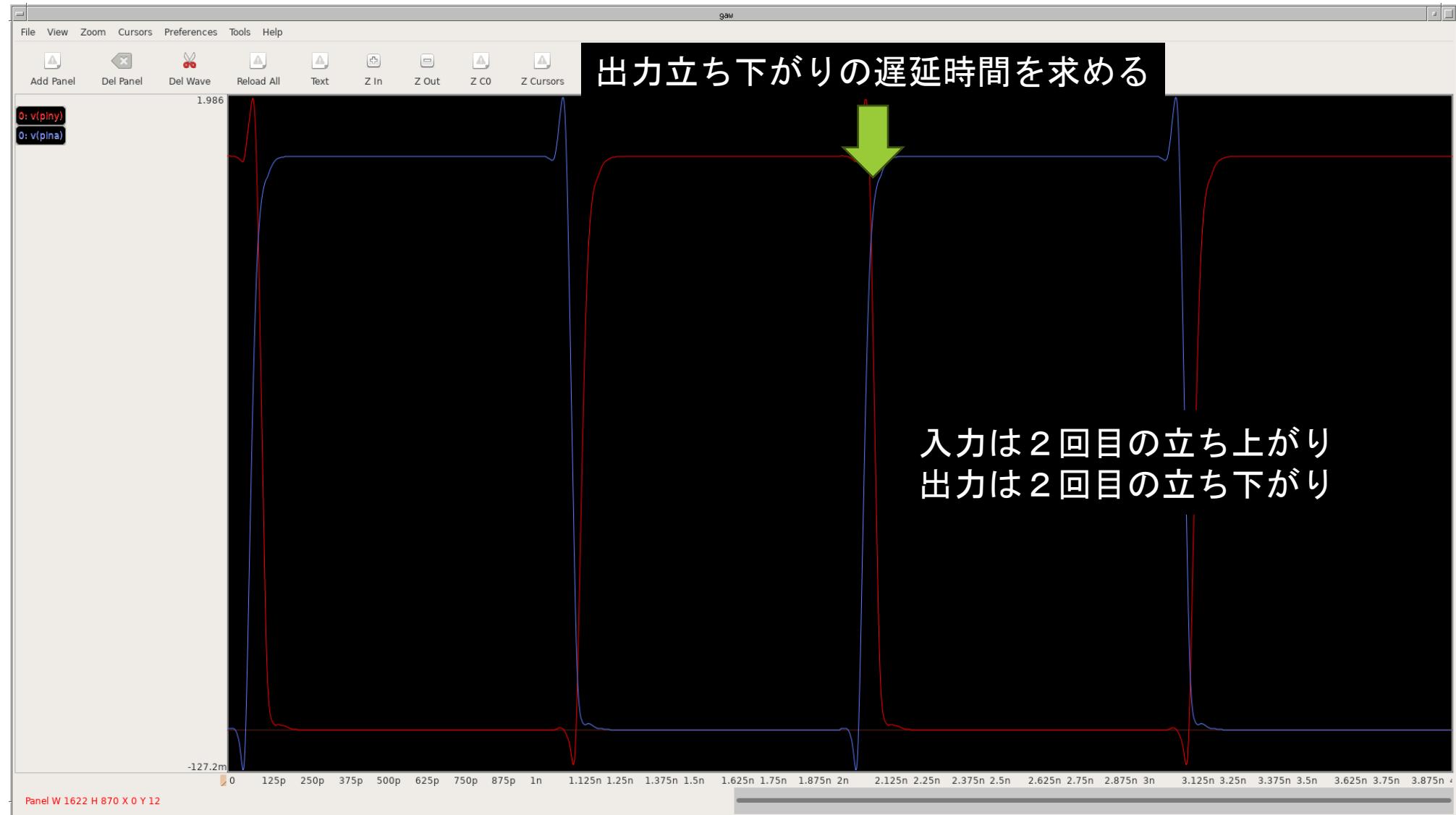
波形ビューワから遅延時間を測る



波形ビューワで遅延時間を見ると…



measで遅延時間を測る



meas を使う

波形やグラフから遅延時間を都度求めるのは面倒 → meas

- 多機能コマンドだが、今回は遅延時間算出に絞って紹介

meas 書式

meas tran [変数] FIND time WHEN [条件] [コマンド]

- [変数] - 結果を代入する変数
- [条件] - 今回は “ $v(\text{ノード}) = Vdd / 2$ ”
- コマンド : RISE / FALL / CROSS のいずれかだが、CROSSは恐らく使わない。

meas を使う

波形やグラフから遅延時間を都度求めるのは面倒 → meas

- 多機能コマンドだが、今回は遅延時間算出に絞って紹介

meas 書式例

meas tran delay1 FIND time WHEN v(vin)=0.9 RISE=2
v(vin)が2回目の立ち上がりで0.9Vの時の時間をdelay1に代入

meas tran delay2 FIND time WHEN v(vout)=0.9 FALL=2
v(vout)が2回目の立ち下がりで0.9Vの時の時間をdelay2に代入

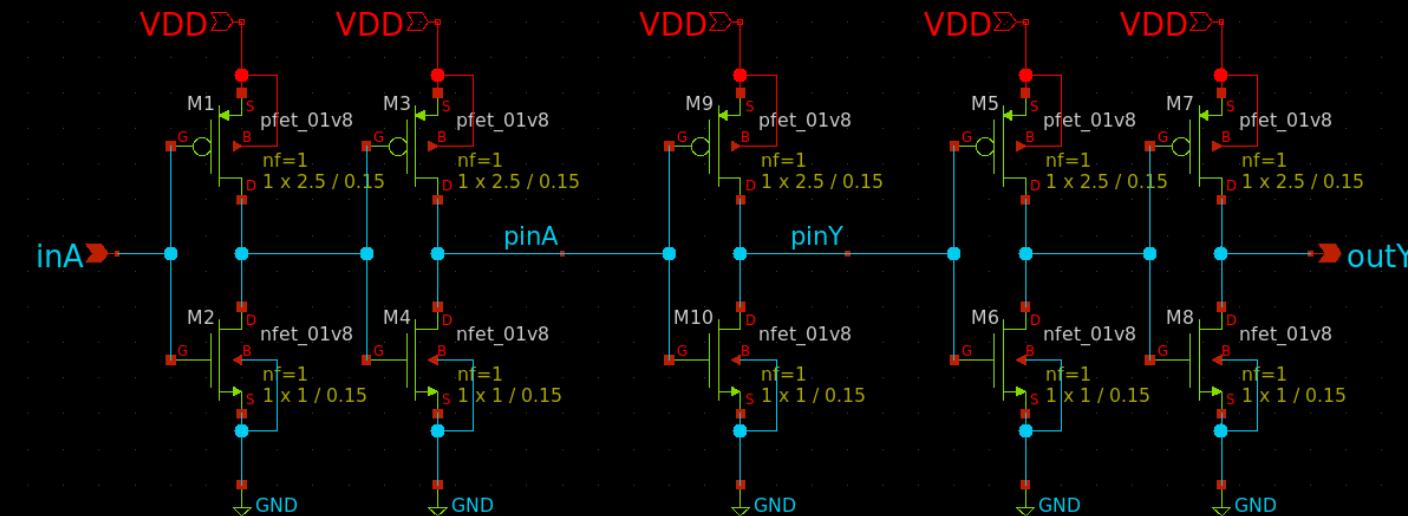
measで遅延時間を見ると…

inv_meas.sch

```
ngspice
VA inA 0 pulse(0 1.8 0 40p 40p 1n 2n) dc 0
VD VDD 0 dc 1.8
.control
tran 1p 4n
meas tran delayA1 find time WHEN v(pinA)=0.9 RISE=2
meas tran delayY1 find time WHEN v(pinY)=0.9 FALL=2
let delay1 = delayY1 - delayA1
echo $&delay1
.endc
```

MODELS

入力は2回目の立ち上がり
出力は2回目の立ち下がり



measで遅延時間を見ると…

inv_meas.sch

```
ngspice
VA inA 0 pulse(0 1.8 0 40p 40p 1n 2n) dc 0
VD VDD 0 dc 1.8
.control
tran 1p 4n
meas tran delayA1 find time WHEN v(pinA)=0.9 RISE=2
meas tran delayY1 find time WHEN v(pinY)=0.9 FALL=2
let delay1 = delayY1 - delayA1
echo $&delay1
.endc
```

MODELS

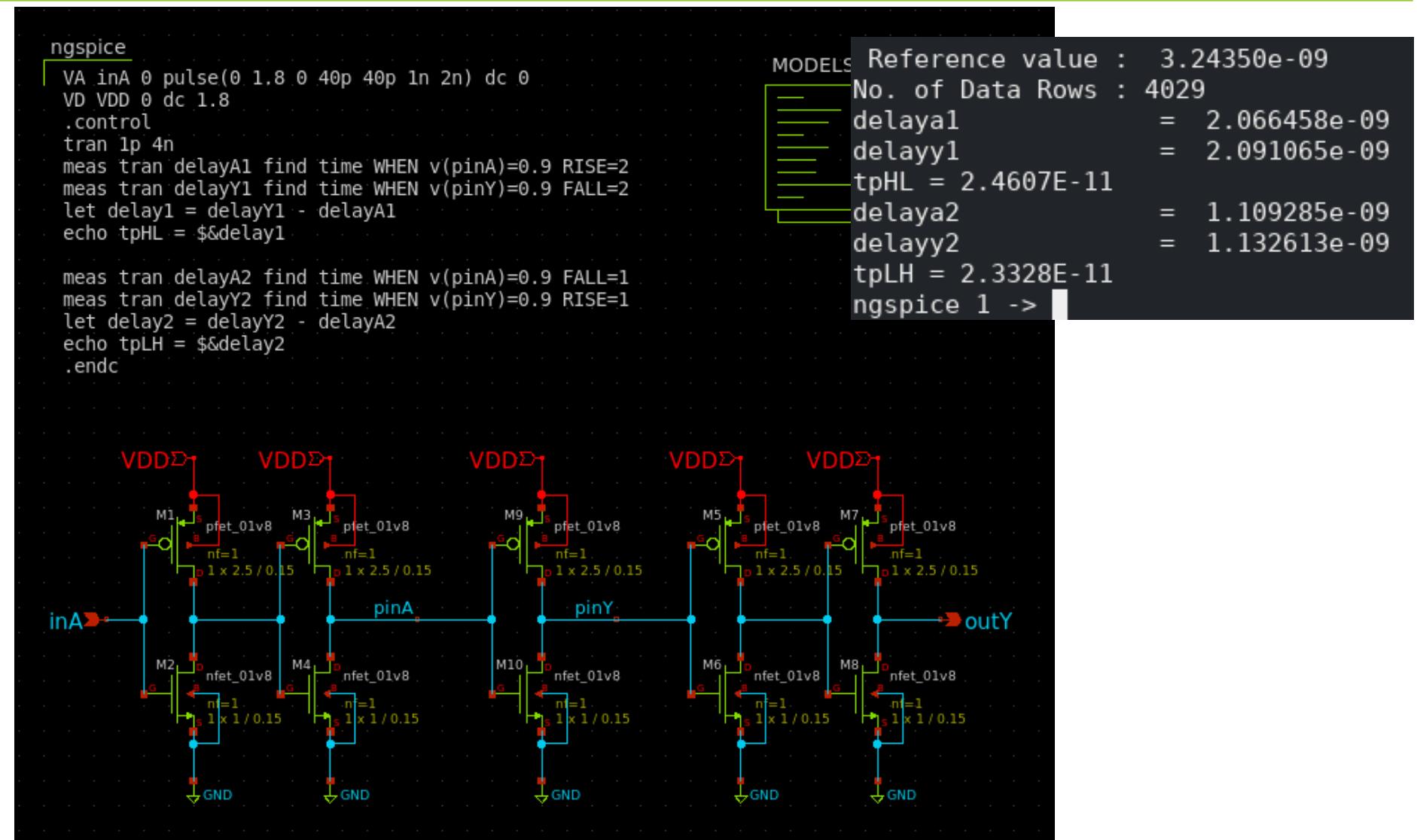
入力は2回目の立ち上がり
出力は2回目の立ち下がり

```
Reference value : 2.18050e-09
No. of Data Rows : 4029
delayA1           = 2.066458e-09
delayY1           = 2.091065e-09
2.4607E-11 ←遅延時間が自動で求まる
ngspice 1 ->
```



同様にして出力立ち上がり遅延時間も見る

inv_meas.sch



立ち上がり時間と立ち下がり時間

N=1u 固定、PMOSのWを変更、入力段・出力段のバッファも変更した

	2.0u	2.1u	2.2u	2.3u
t _{PHL}	23.238ps	23.496ps	23.764ps	24.035ps
t _{PLH}	23.889ps	23.725ps	23.591ps	23.482ps

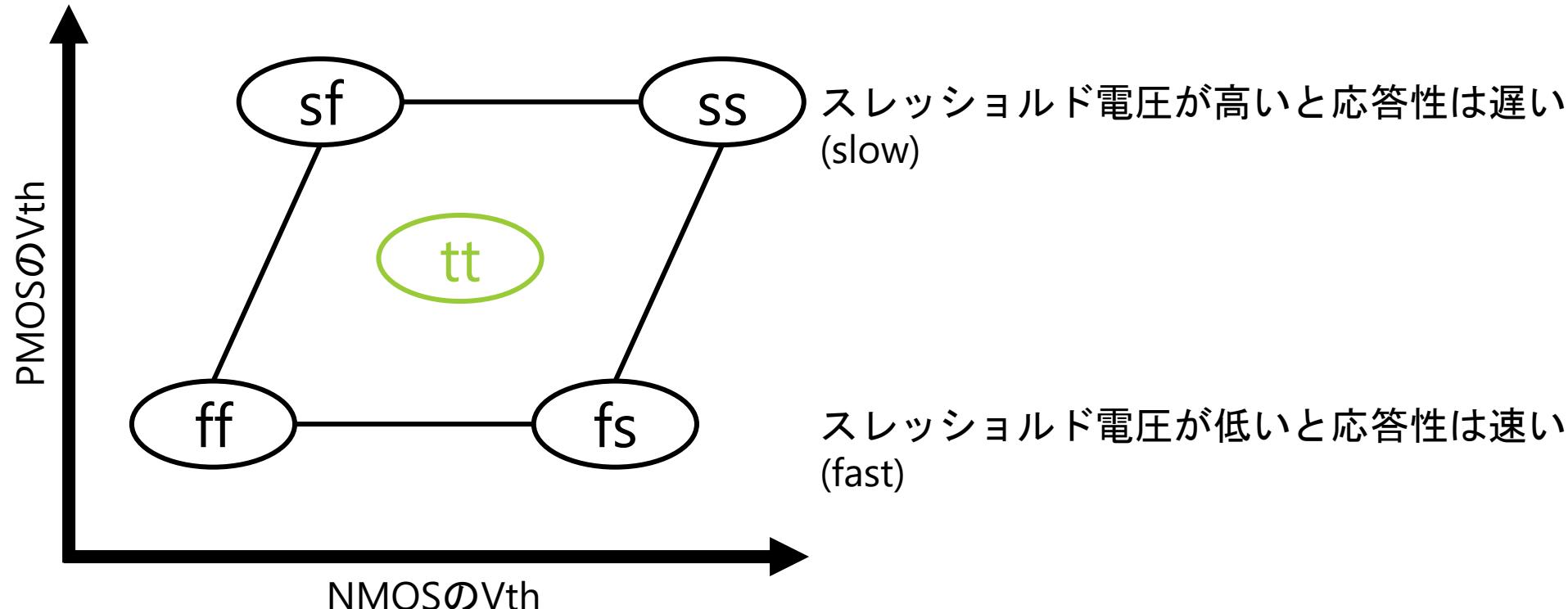
- N=1uならP=2.1u or 2.2u がよさそう

$R_n = R_p$ で立ち上がり遅延と立ち下がり遅延は一致しない

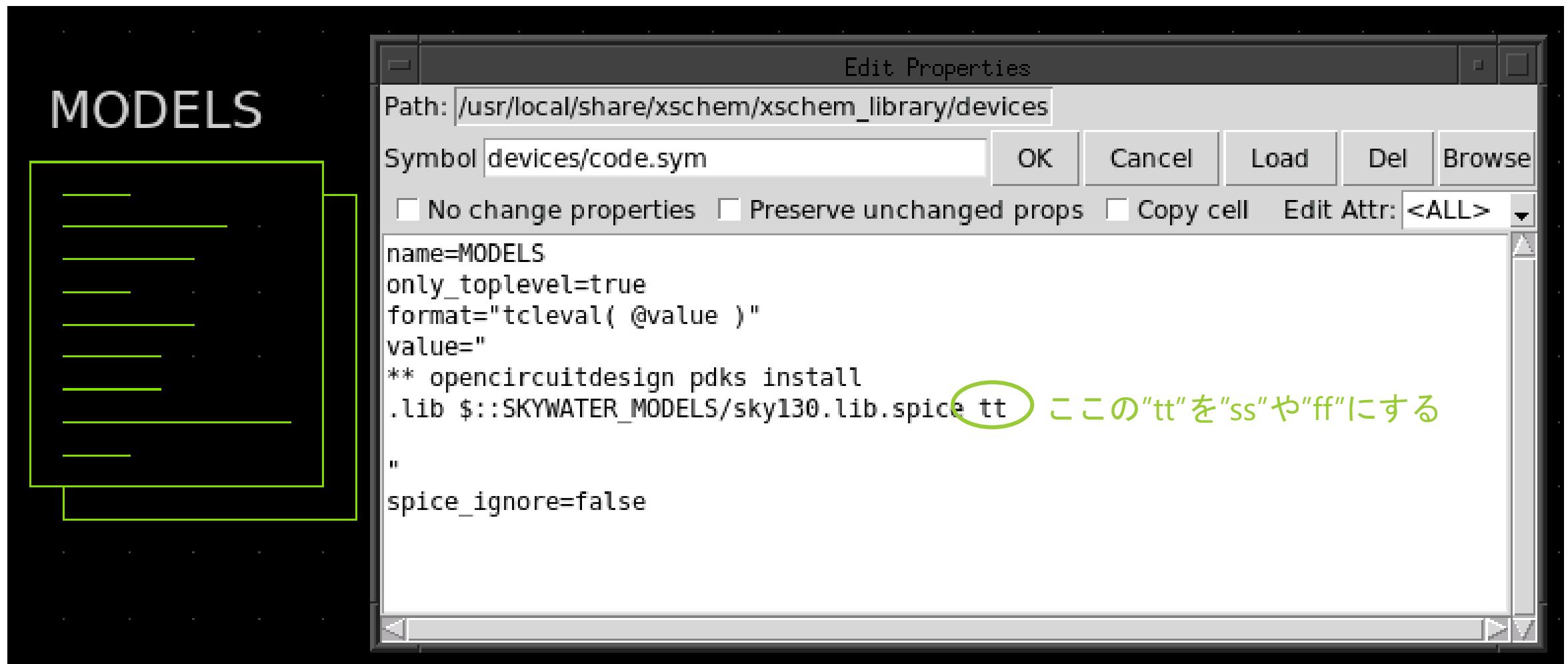
- PMOSとNMOSのV_{th}, gmが一致していない

シミュレーションモデルについて

- SSモデル – 応答性が最悪(slow-slow)の時のモデル
- ttモデル – 応答性がtypical-typicalなモデル



モデル変更



立ち上がり時間と立ち下がり時間

- モデルssで実行すると結果が変わる

	2.4u	2.5u	2.6u	2.7u
t _{PHL}	33.242ps	33.538ps	33.856ps	34.191ps
t _{PLH}	33.845ps	33.754ps	33.724ps	33.750ps

- N=1uならP=2.**5**u or 2.**6**u がよさそう

$R_n = R_p$ で立ち上がり遅延と立ち下がり遅延は一致しない

- ssではP:N = 3:1 だった

LSI予備知識

プレーナ型FETプロセス レイアウト編

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
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)

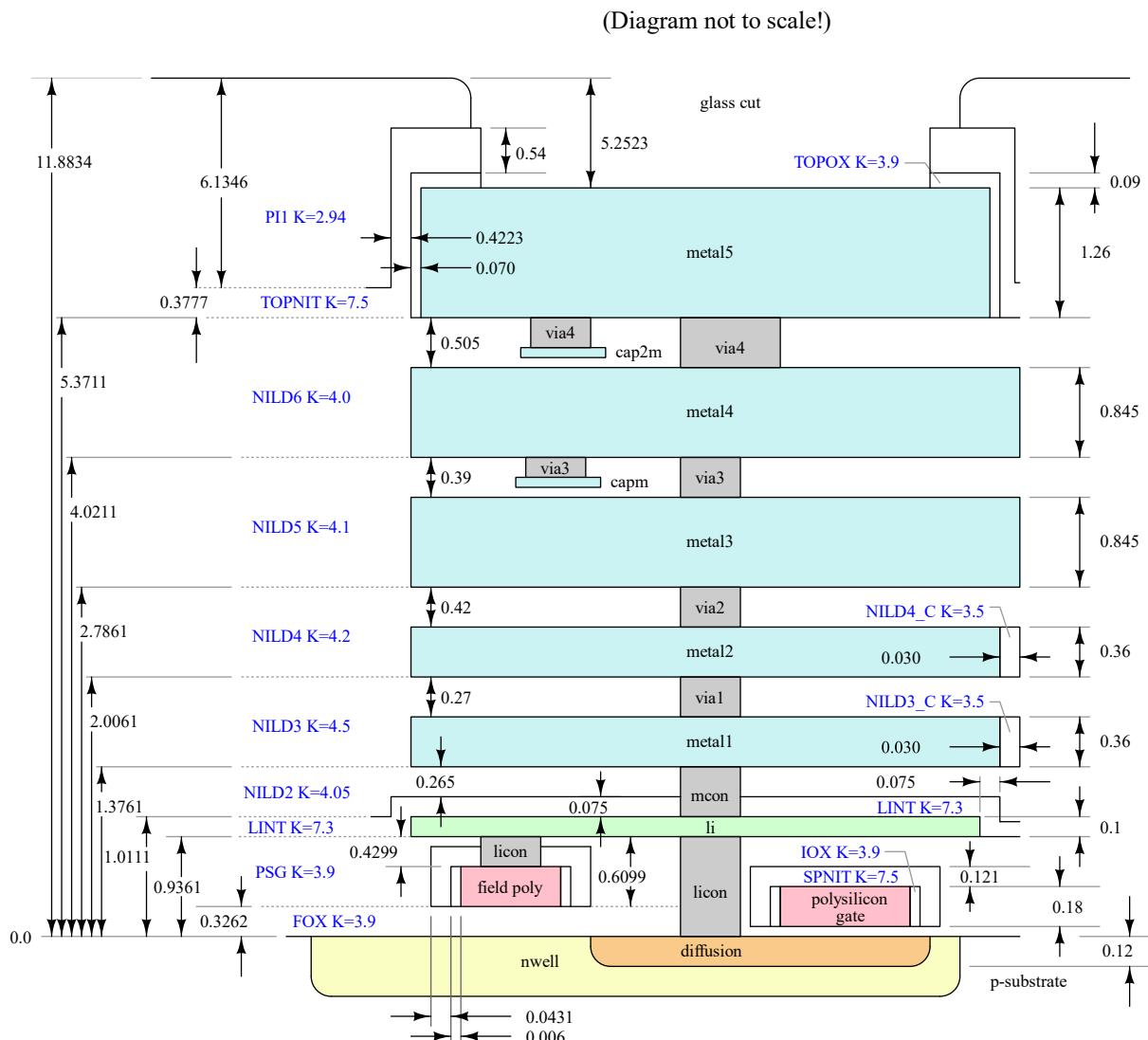
プレーナ（平面）型MOSトランジスタ

substrate（基板）を最下層として積層していく。

基板上にMOSトランジスタ(well, diffusion, poly)

トランジスタより上は配線層(metal*, via*)

metal間に絶縁体(insulator)を挟んで
MIMキャパシタが作成可能(capm, cap2m)

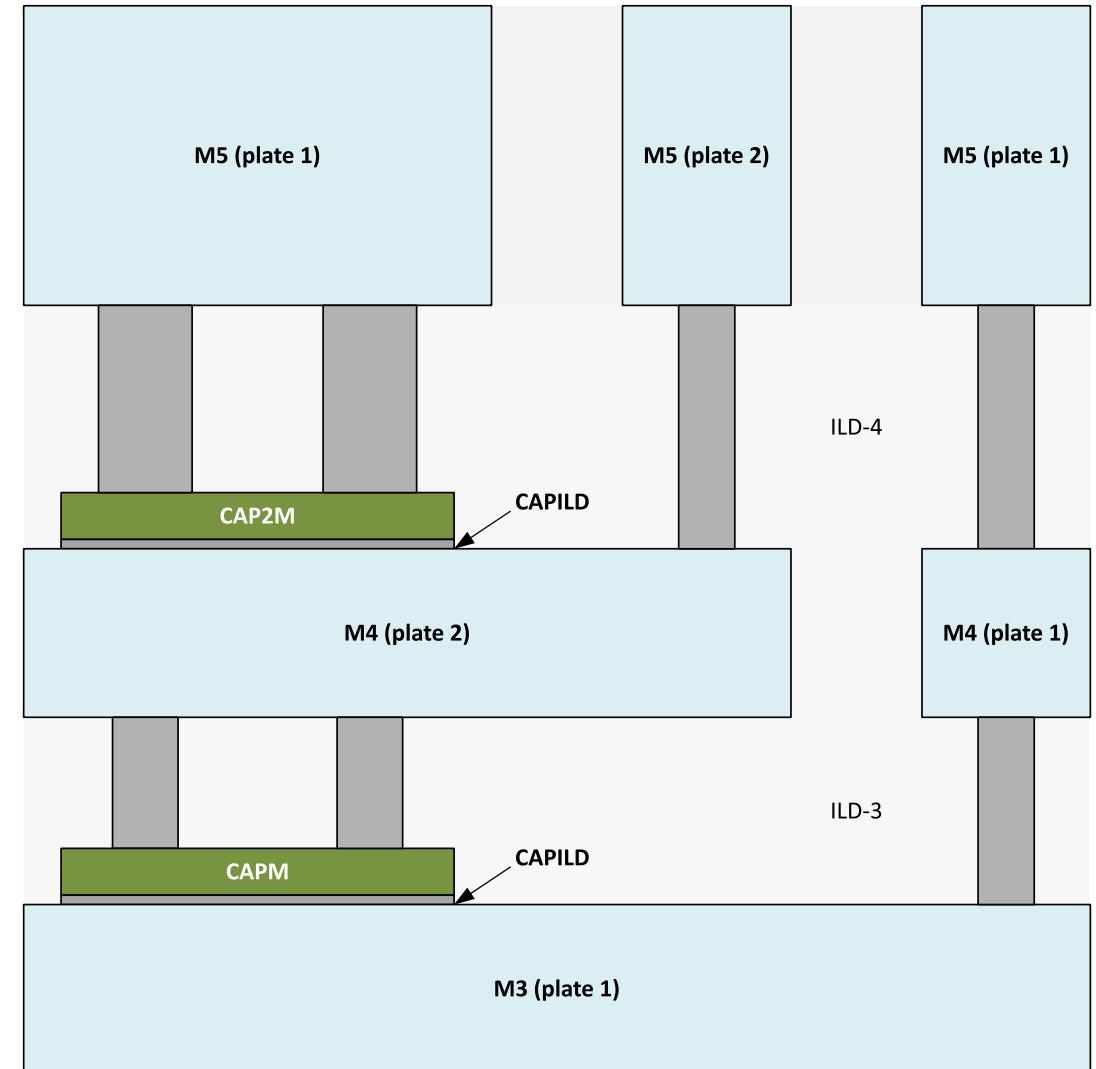
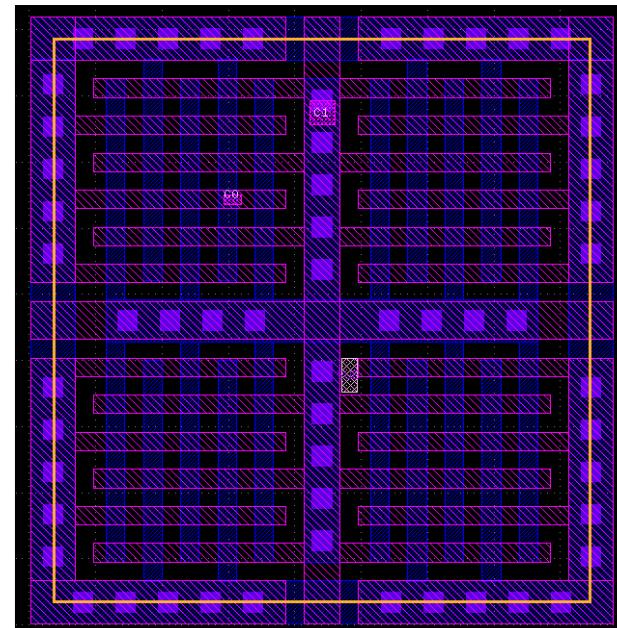


MIMCAP

M5-M4間とM4-M3間で2つ存在

単に配線層を交差させても小容量だがキャパシタになる
→寄生容量の原因

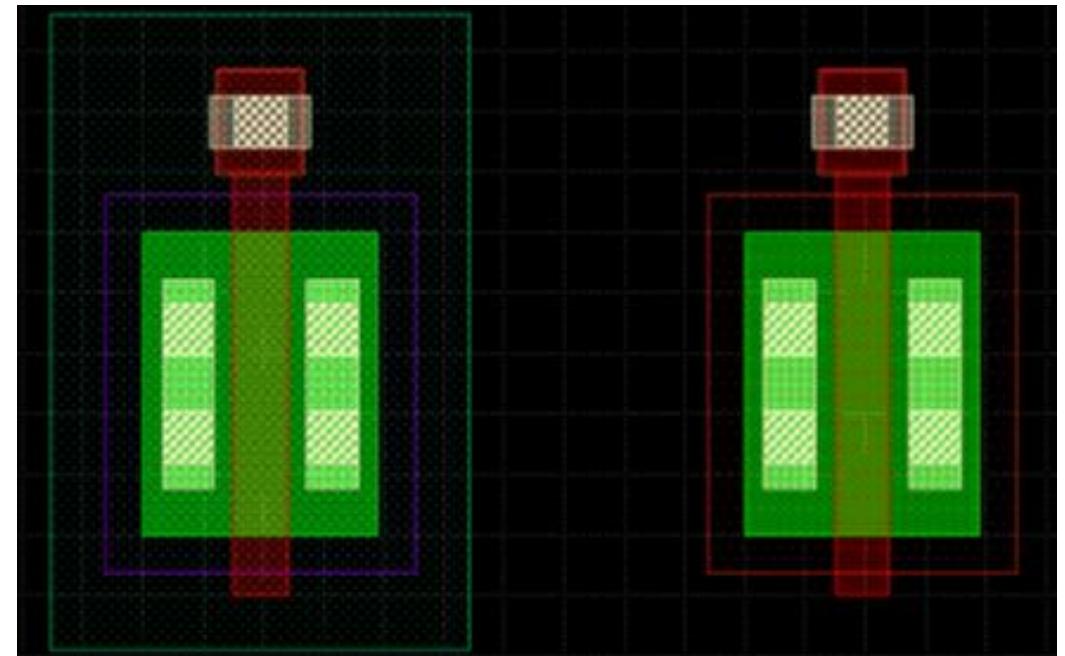
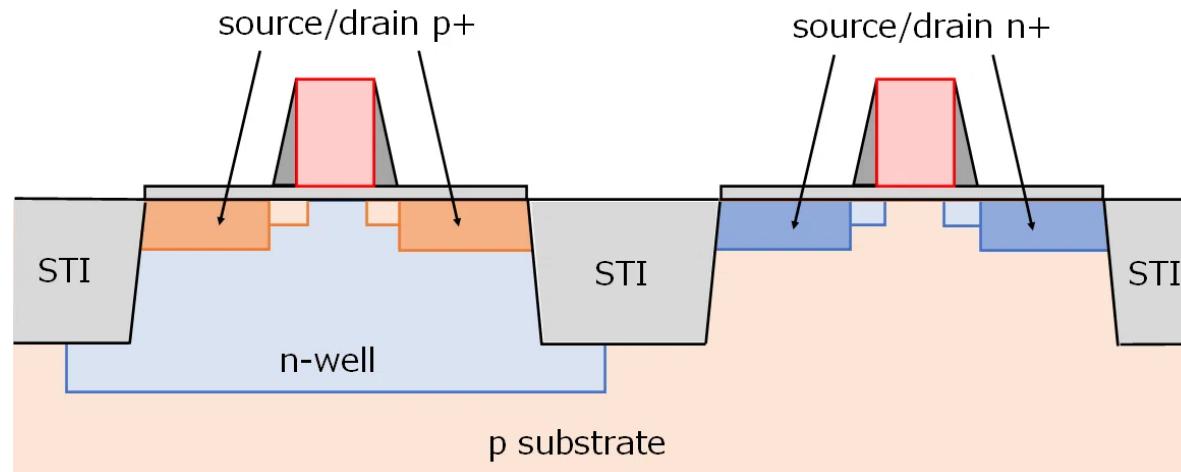
METAL1とMETAL2のみで構成されたキャパシタ→



ダブルウェル, デュアルウェル, ツインウェル

P-サブストレートとN-wellにより構成されるCMOSトランジスタ

PCELLにより自動生成されるレイアウトはダブルウェル

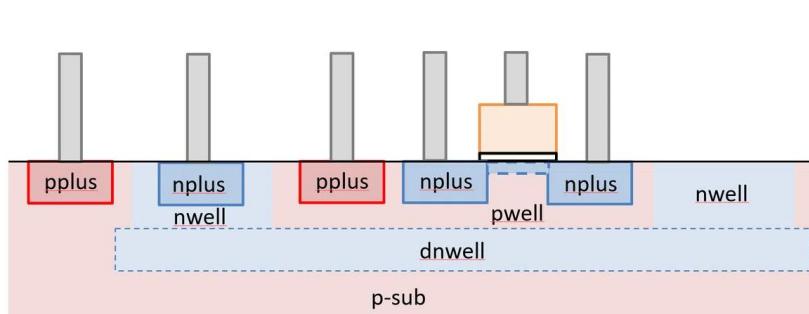
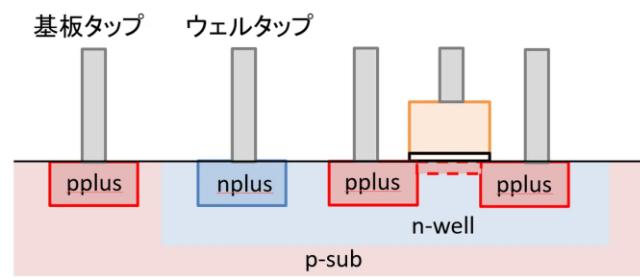


- PMOSは個別にwellを持つのでボディの電圧は可変
- NMOSのボディはサブストレートであるため、ダブルウェルNMOSのボディは全て同じバイアス電圧(回路内で一番低い電圧, GND, VSS)

https://note.com/akira_tsuchiya/n/n416b7f74b701

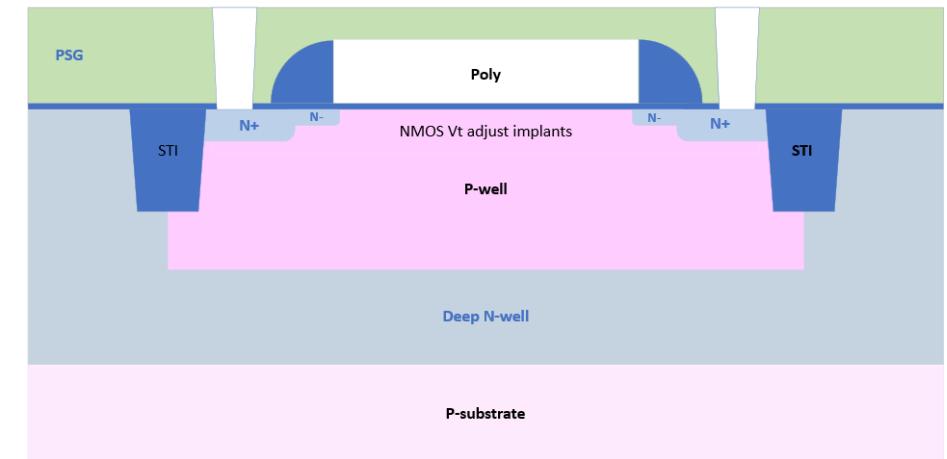
トリプルウェル

Deep N-well, P-well, N-wellにより構成されるCMOSトランジスタ
PDKで説明されている断面図はこれ



PMOS

NMOS



https://note.com/akira_tsuchiya/n/nd4444304f013

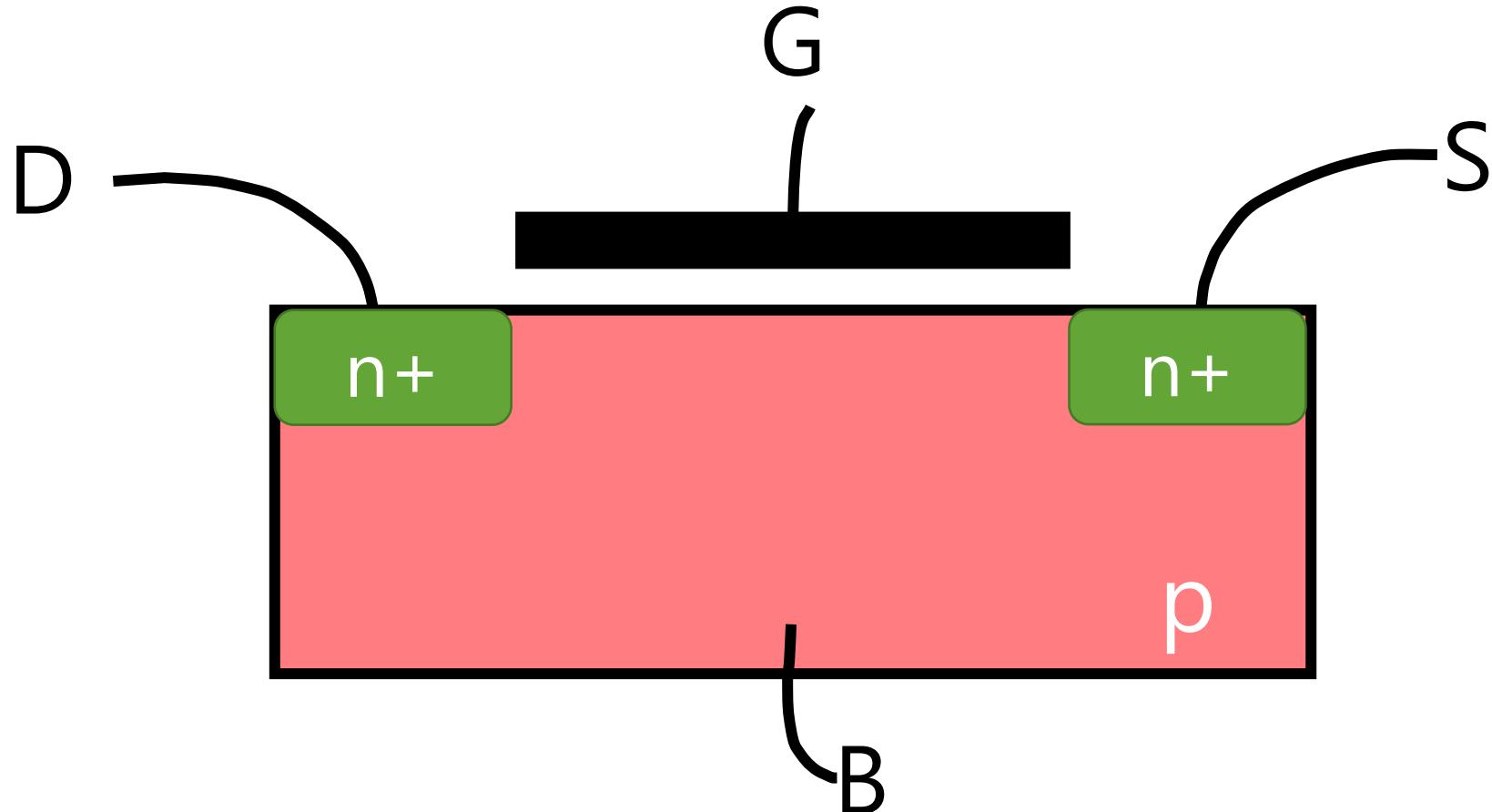
- ・ NMOSが個別にwellを持つのでボディの電圧は可変必要となるアプリケーション例
 - ・ スレッショルド電圧を調整したい (body effect)
 - ・ NMOSのボディをソースに接続してカスコード接続
 - ・ サブストレートからのノイズ遮断が必要など

基板バイアス効果 (body effect)

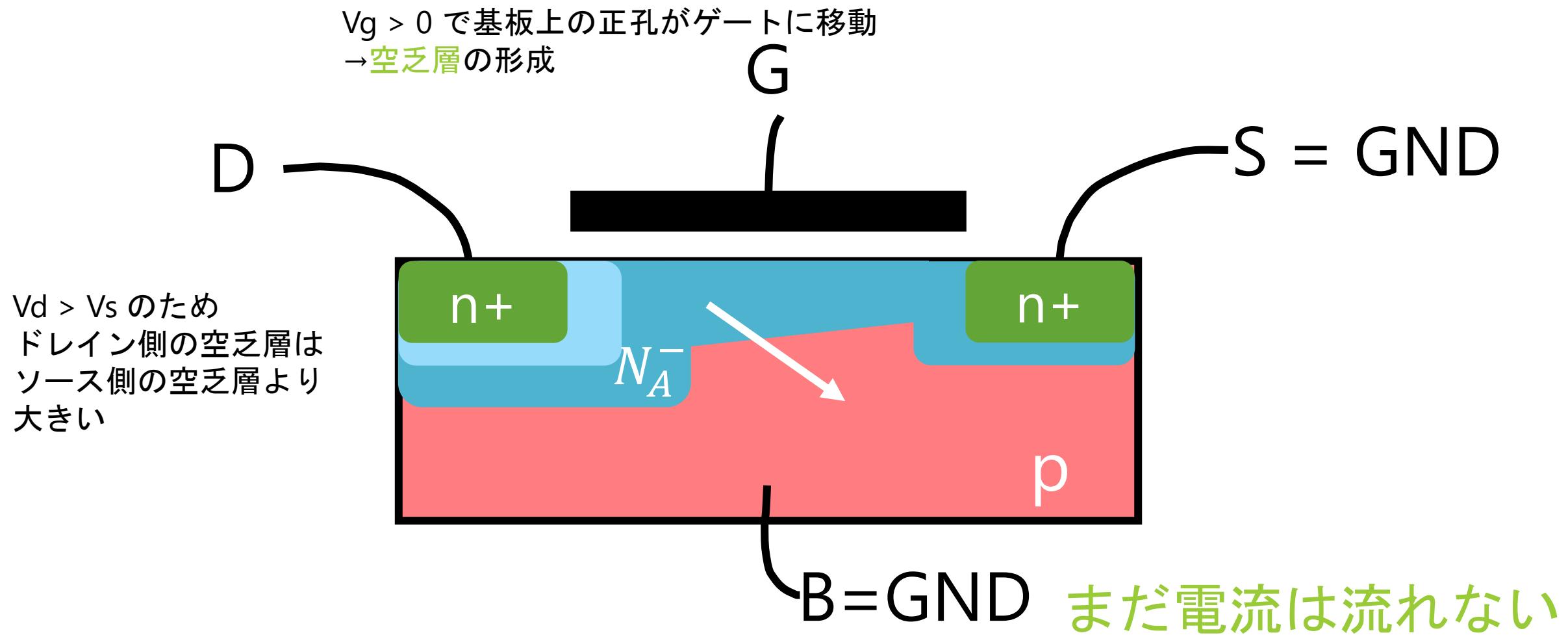
ボディにかかるバイアス電圧によってトランジスタの動作が変化する

- スレッショルド電圧 (V_{th}) が変化
- ゲート電圧を固定して比較した時、ドレイン・ソース間電流(I_{ds})が変化

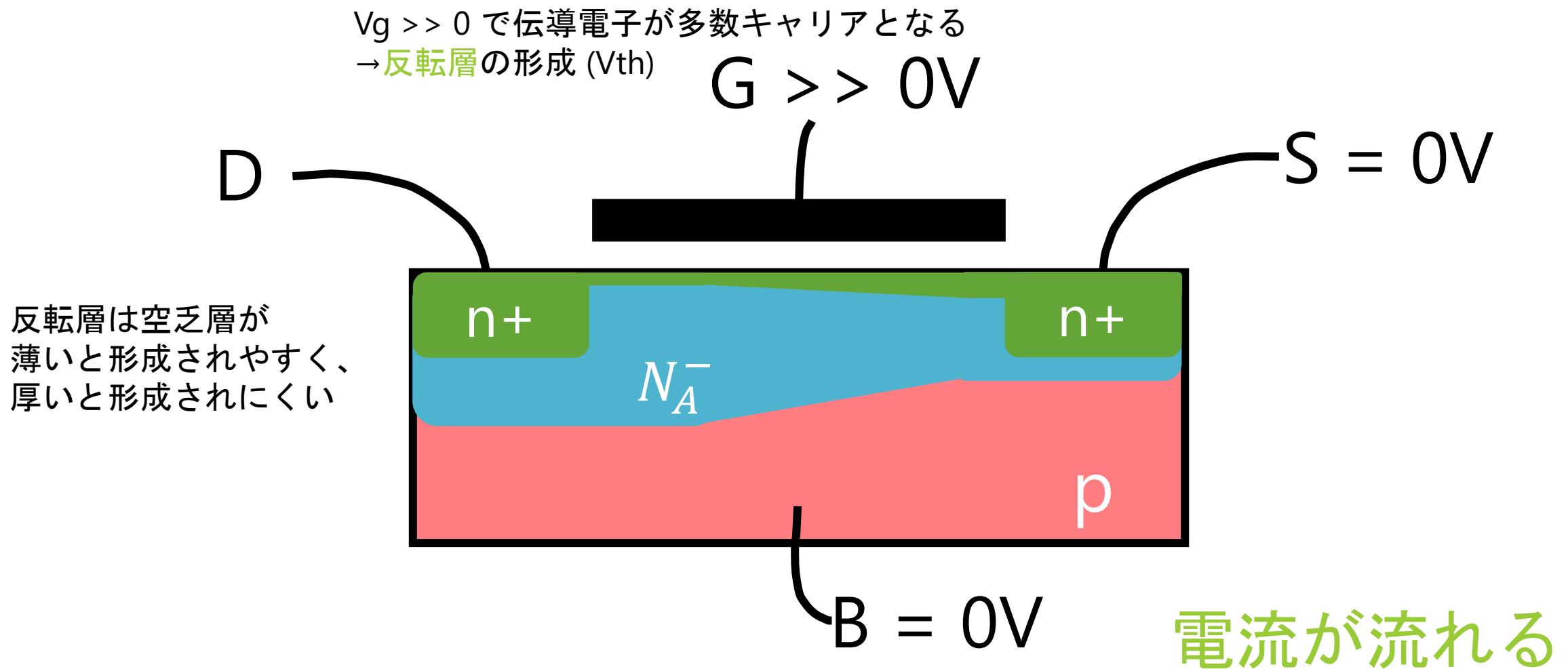
nMOS 簡単な模式図



nMOS 簡単な模式図



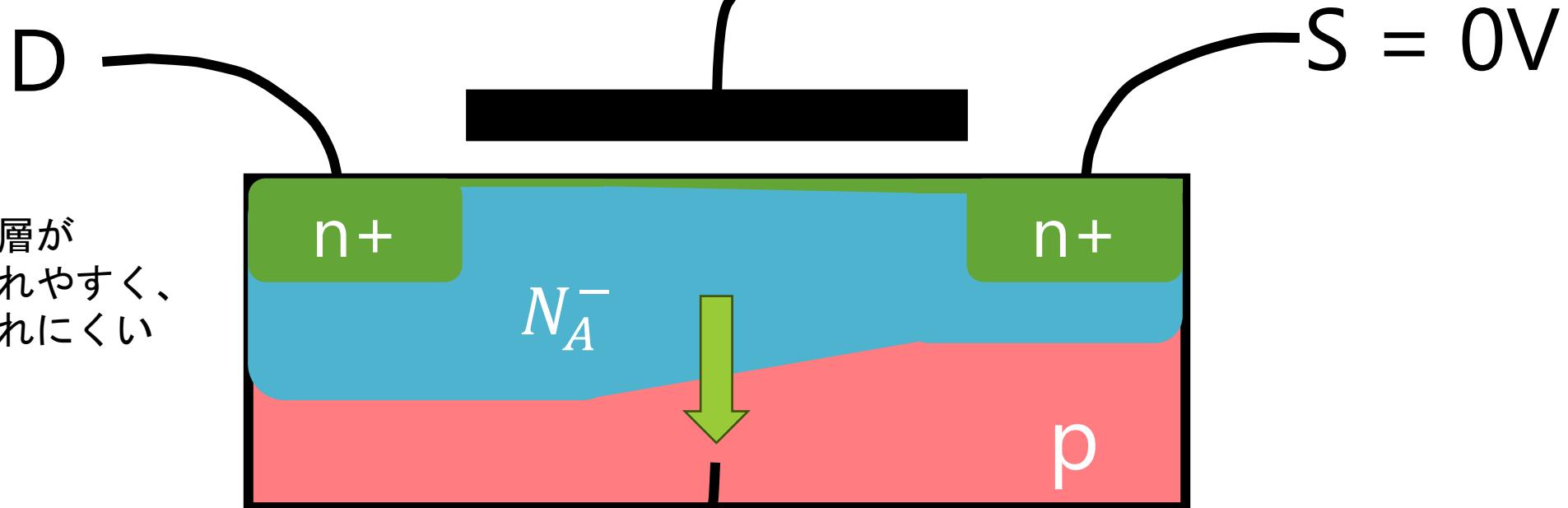
nMOS 簡単な模式図



nMOS 簡単な模式図

$V_g >> 0$ で伝導電子が多数キャリアとなる
→ 反転層の形成 (V_{th})

$$G >> 0V$$



$V_b < 0$ となると空乏層が広がる

→ 反転層が形成されにくくなり(V_{th} 上昇)、薄くなる (オン抵抗増加)

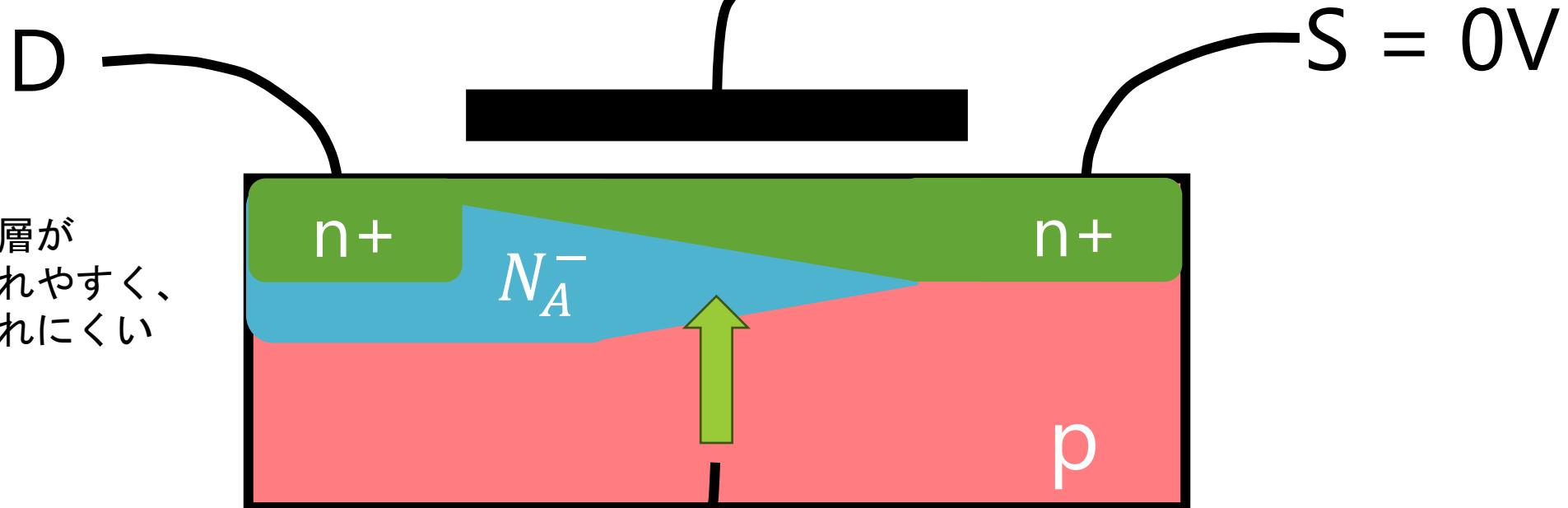
$$B < 0V$$

電流が流れにくい

nMOS 簡単な模式図

$V_g >> 0$ で伝導電子が多数キャリアとなる
→ 反転層の形成 (V_{th})

$$G >> 0V$$



反転層は空乏層が
薄いと形成されやすく、
厚いと形成されにくい

$V_b > 0$ となると空乏層が狭まる

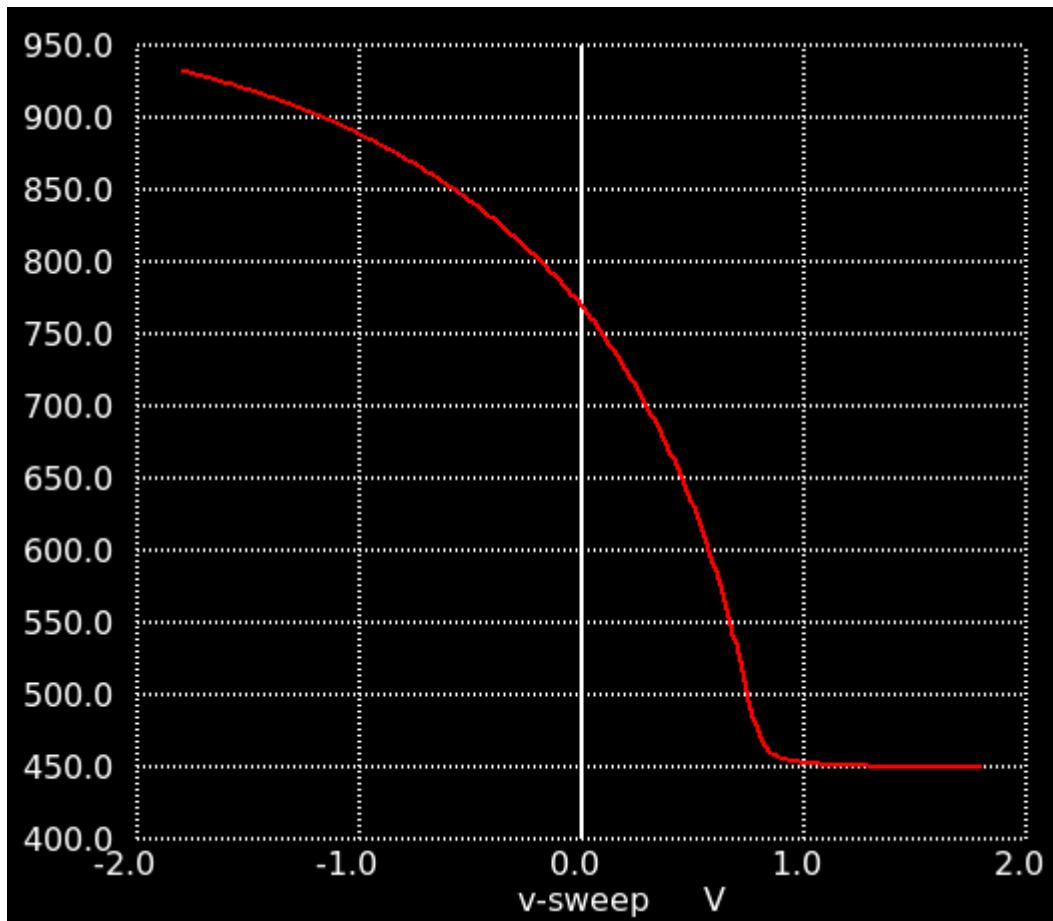
→ 反転層が形成されやすくなり(V_{th} 減少)、厚くなる (オン抵抗減少)

$$B > 0V$$

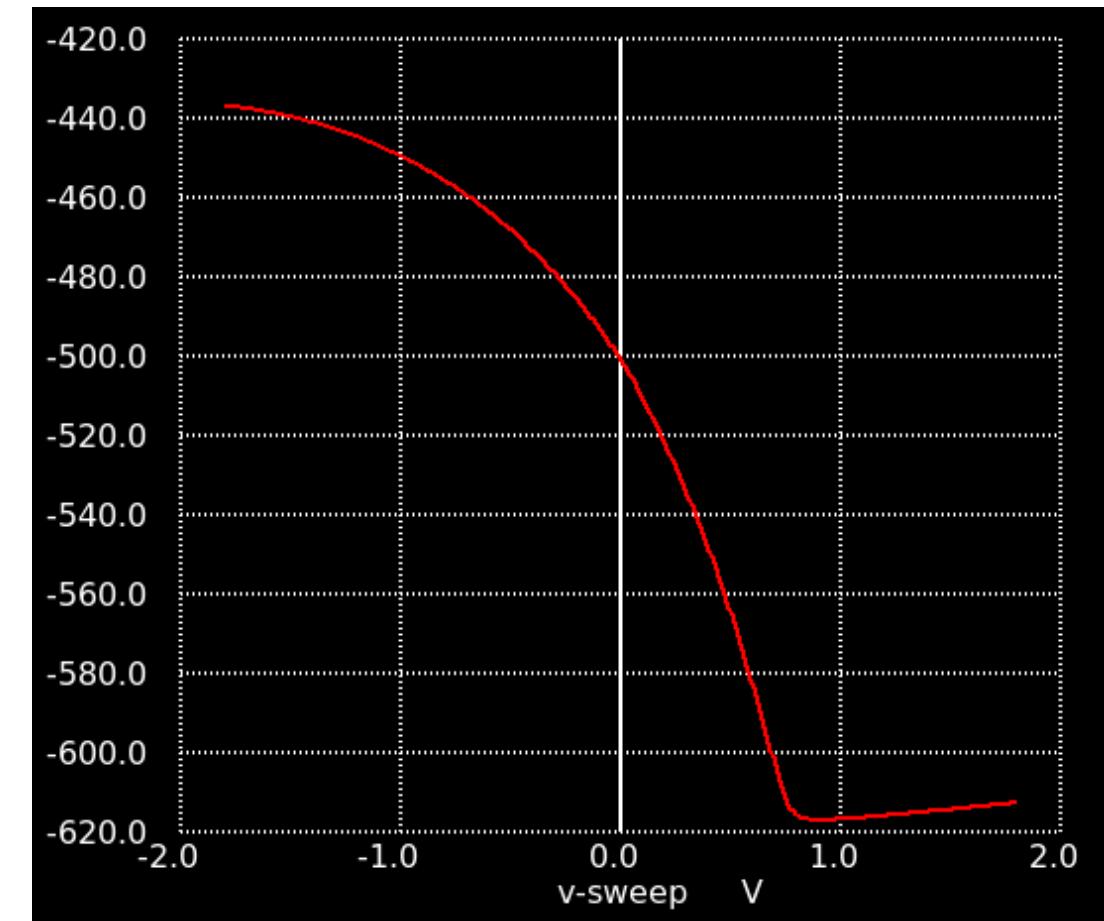
電流が流れやすい

nMOSの基板バイアス効果

trbench_body.sch

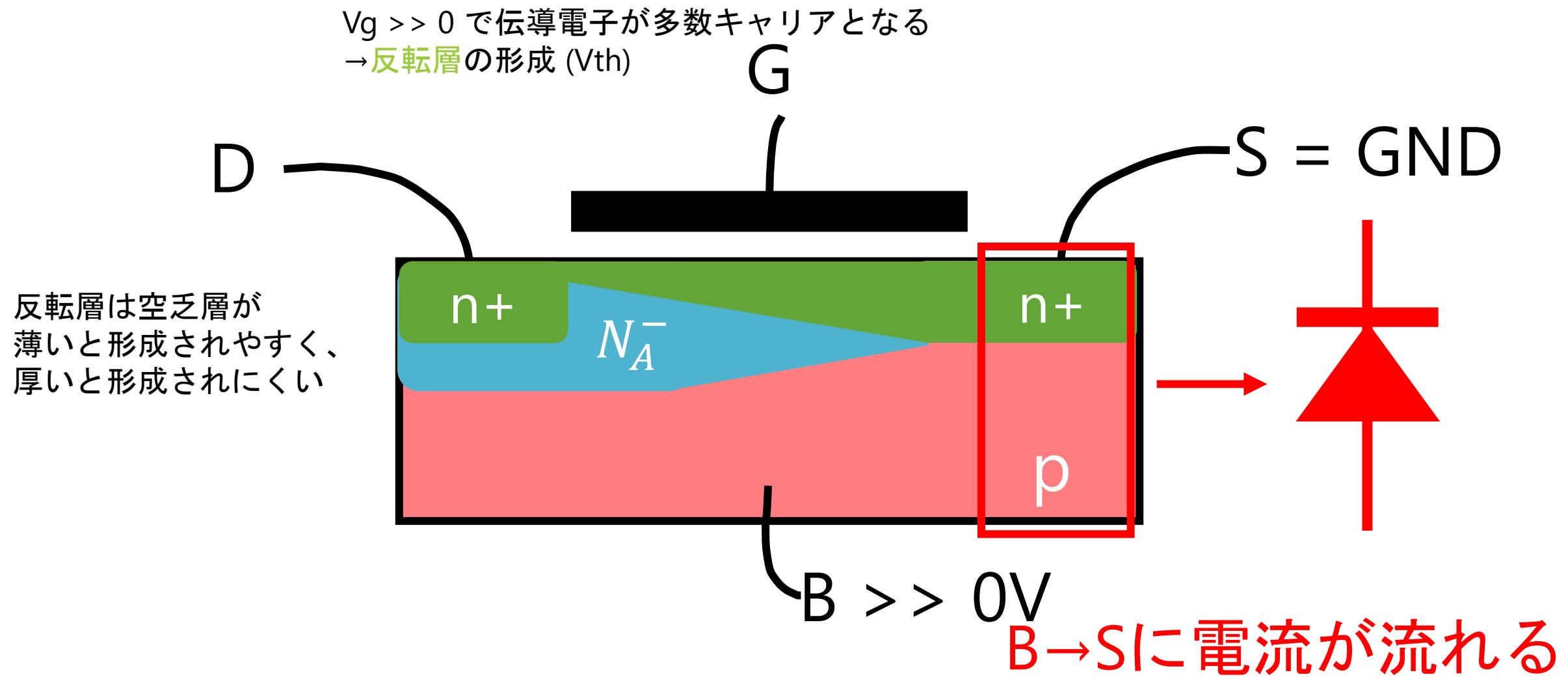


$$V_{BS} - V_{TH}$$

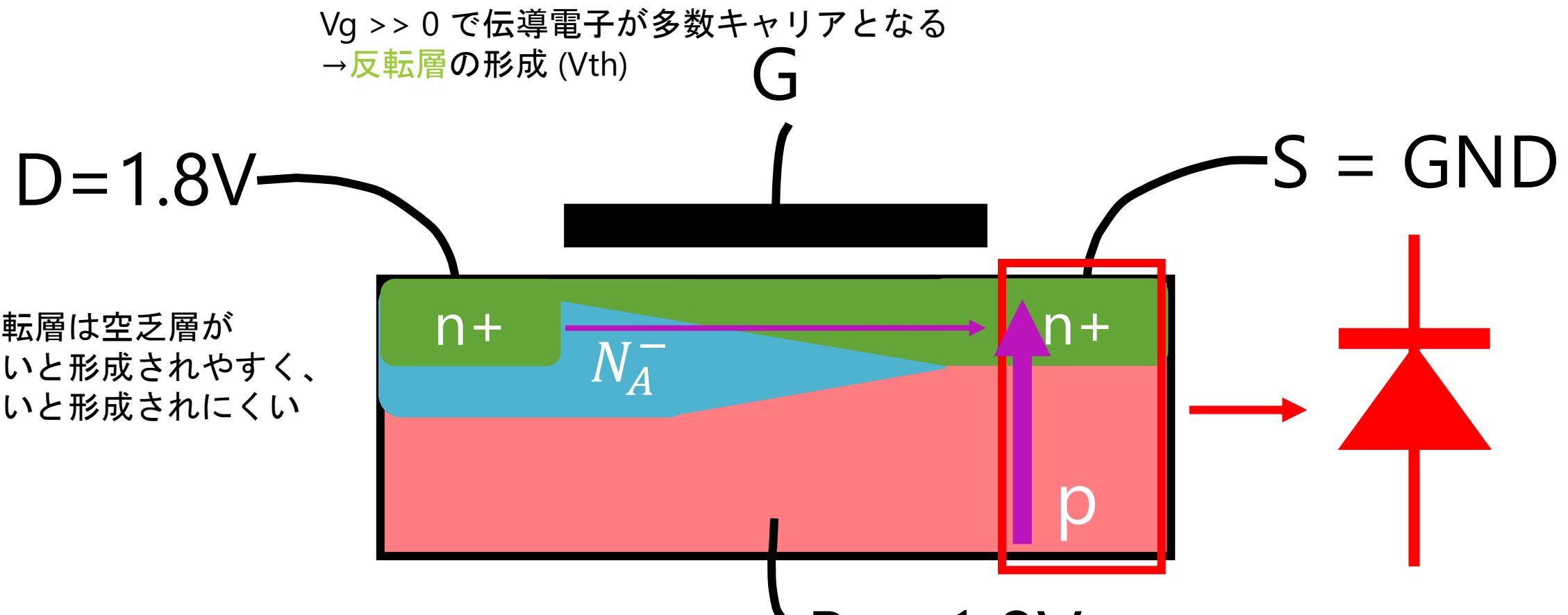


$$V_{BS} - I_{DS}$$

nMOS 簡単な模式図



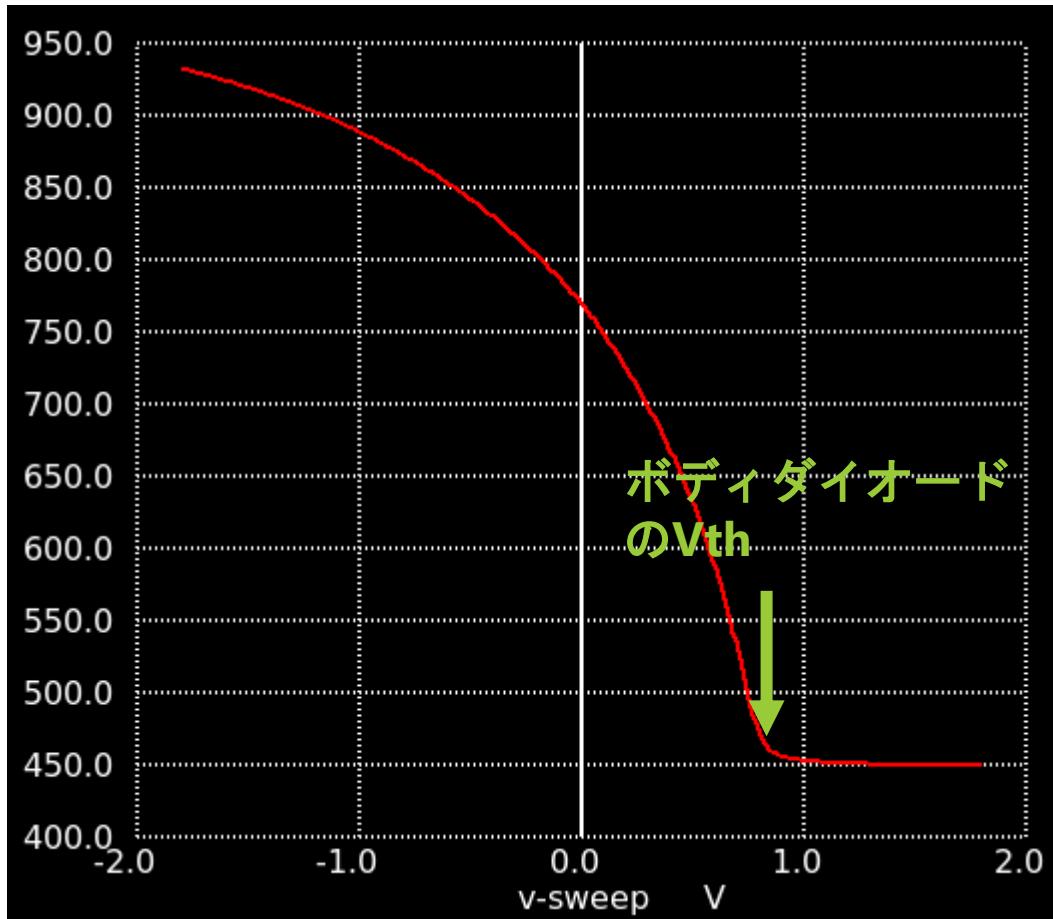
nMOS 簡単な模式図



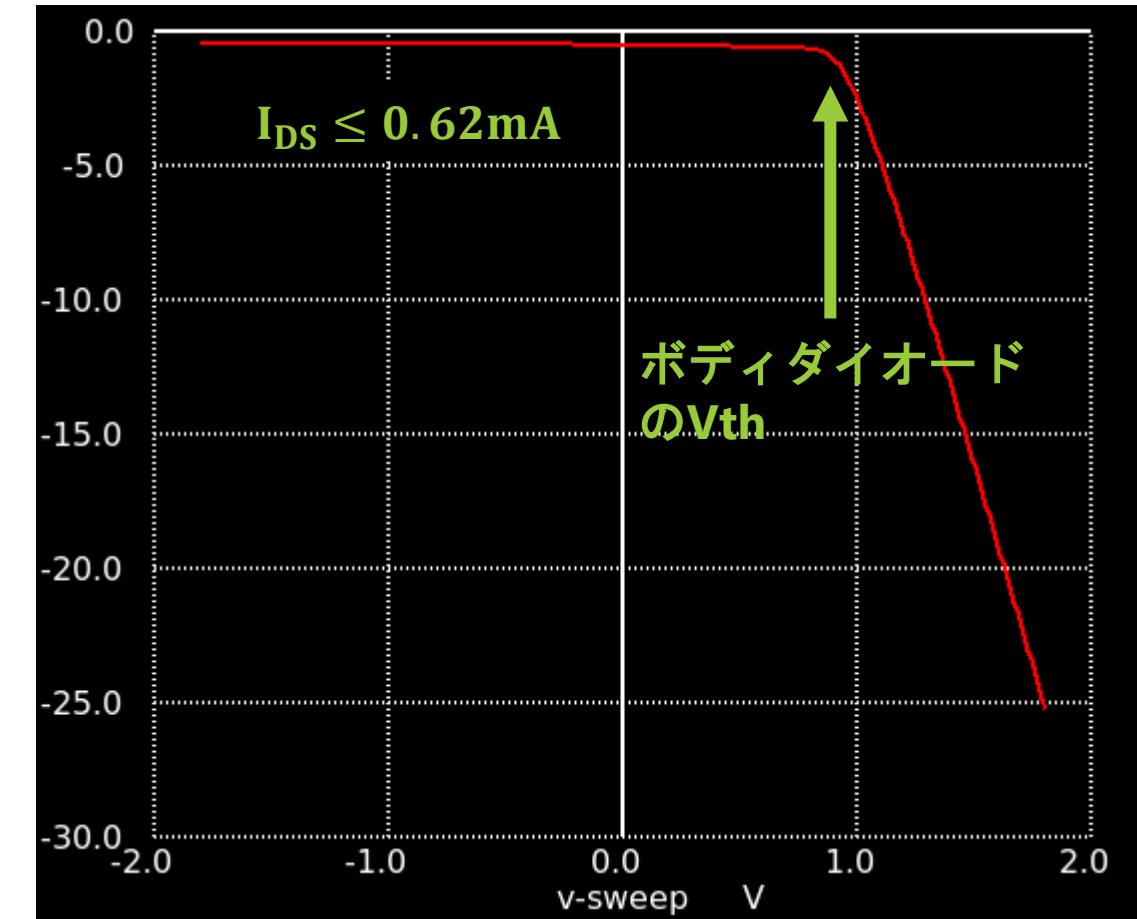
ドレインとボディをショートさせると
 $I_{DS} \ll I_{BS}$ となるため
トランジスタとして機能しない

NMOSの基板バイアス効果

trbench_body.sch



$$V_{BS} - V_{TH}$$



$$V_{BS} - (I_{DS} + I_{BS})$$

ボディにバイアス電圧をかけるには…

ボディダイオードの V_{TH} を超えるとトランジスタが導通・故障するので、それを越さない範囲でバイアス電圧をかける

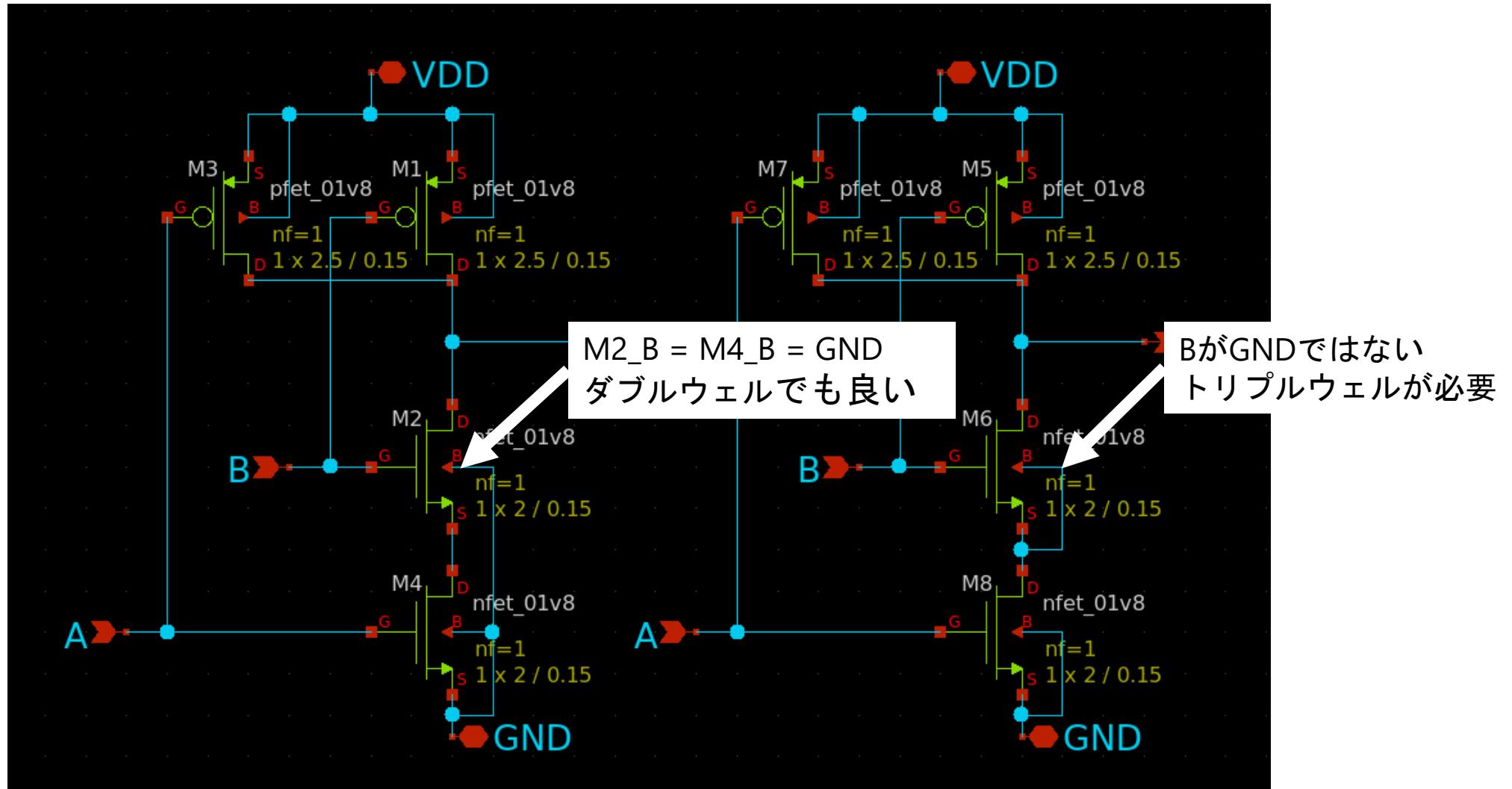
ボディ電圧を変更する場合はトリプルウェル構造が必須

- ダブルウェル構造よりも描画が面倒かつ面積が大きい
- ロジック回路でトリプルウェル構造は不利（面積大→集積密度低下）

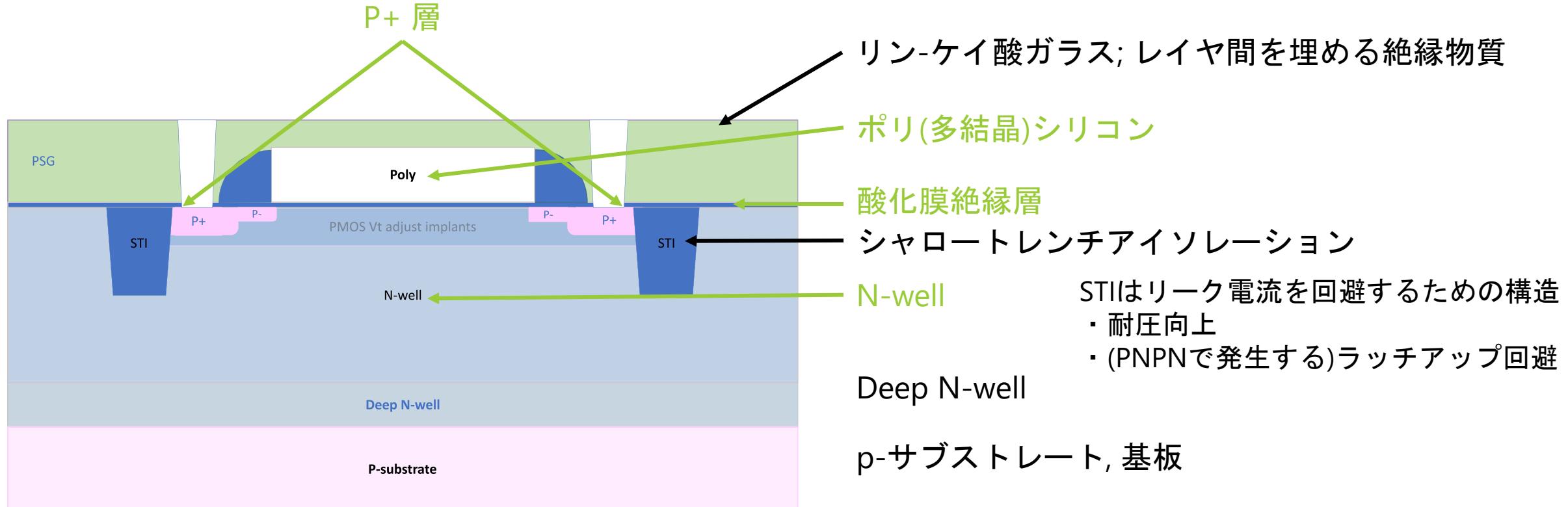
集積度で不利なので特にこだわりが無ければダブルウェル構造がオススメ

- PMOSのボディはVDD、NMOSのボディはGND(VSS, ±0V)に接続する

例：2入力NAND



プレーナ型pMOSトランジスタ（トリプルウェル）



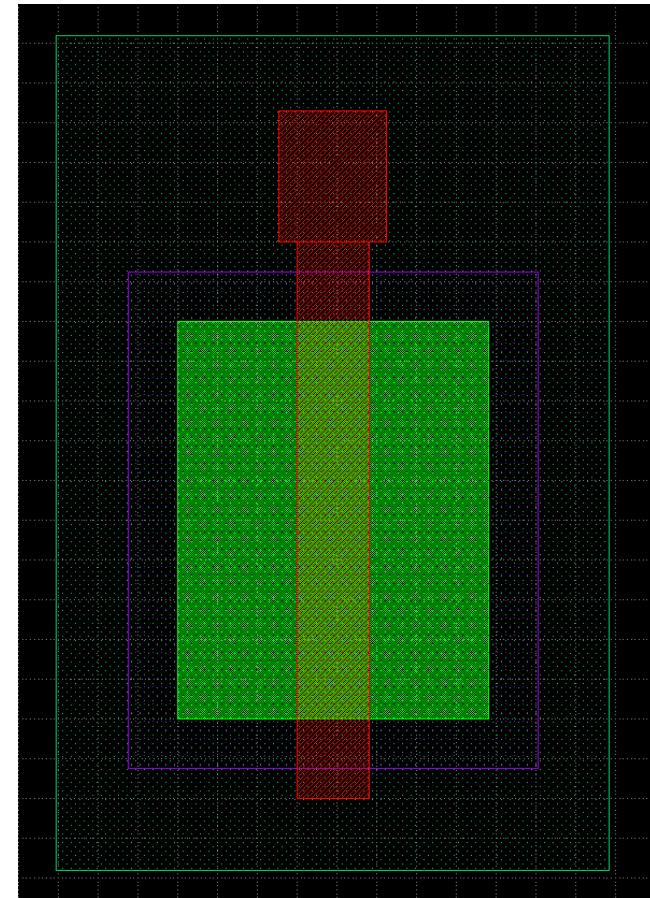
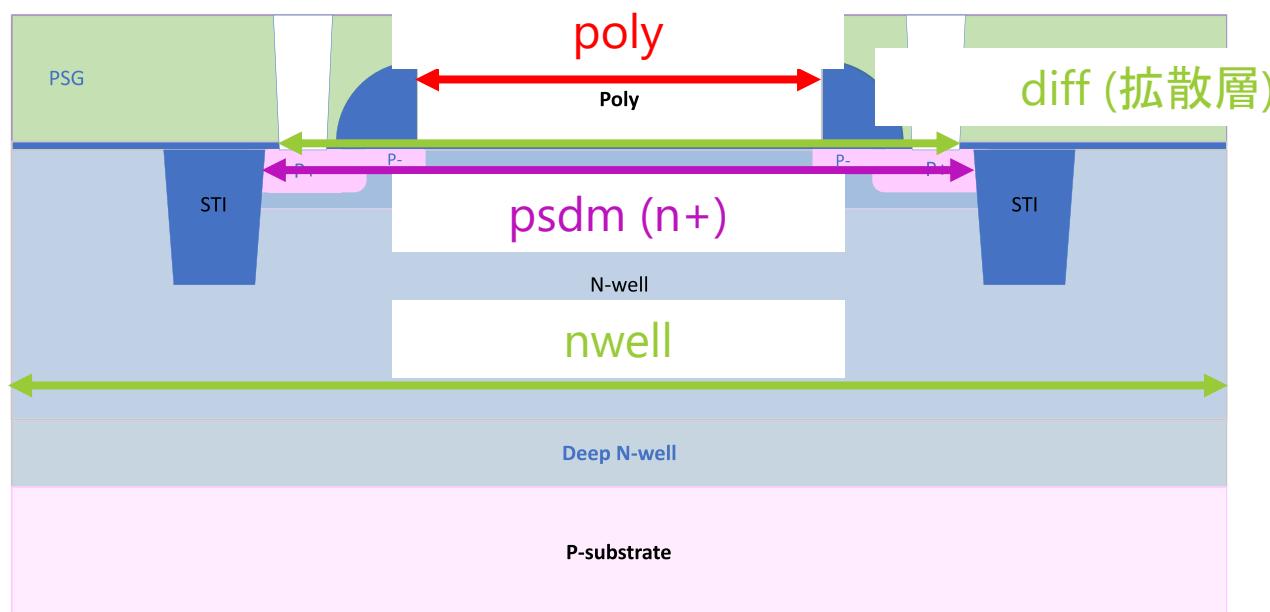
プレーナ型pMOS

nwell
diff / active
psdm
poly

N-well

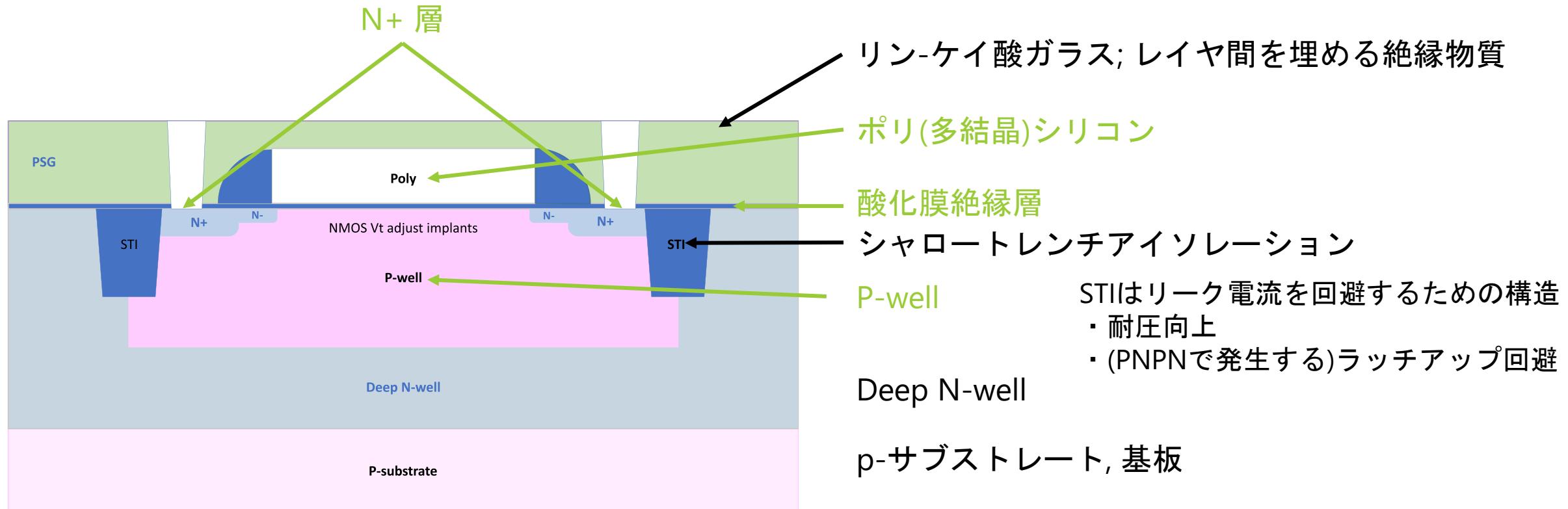
wellまたはsubstrateと対になるタイプの拡散層
不純物注入によるP+イオン領域
ポリシリコン

PCELLでは、dnwellは描かれない（なくても良い）



green	nwell.drawing - 64/20
green	diff.drawing - 65/20
purple	psdm.drawing - 94/...
red	poly.drawing - 66/20

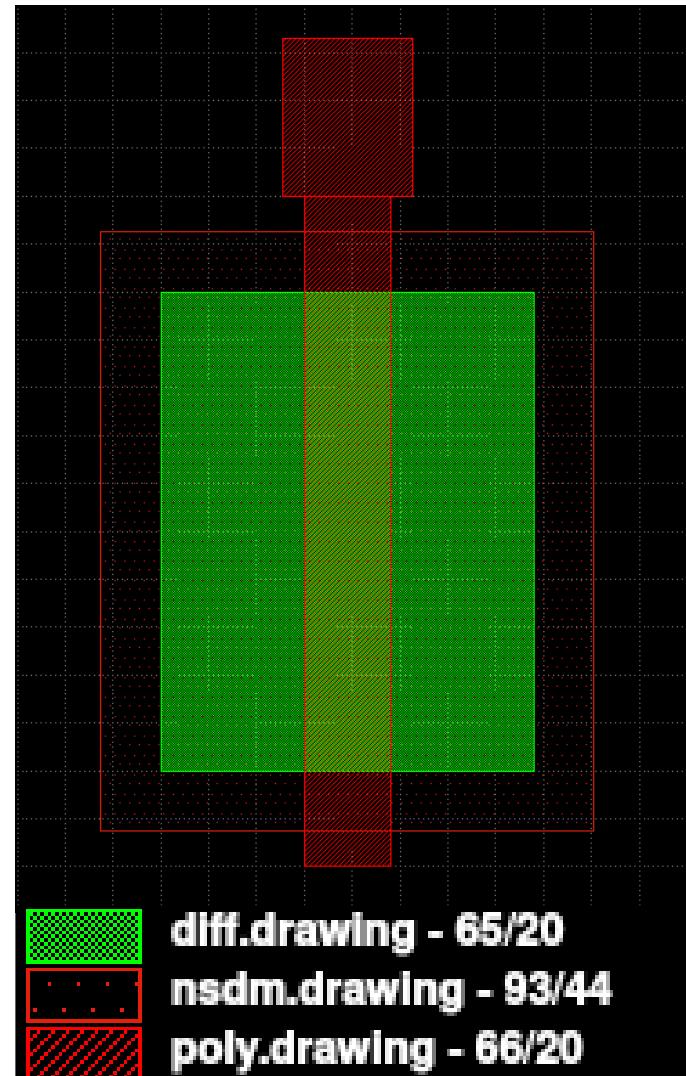
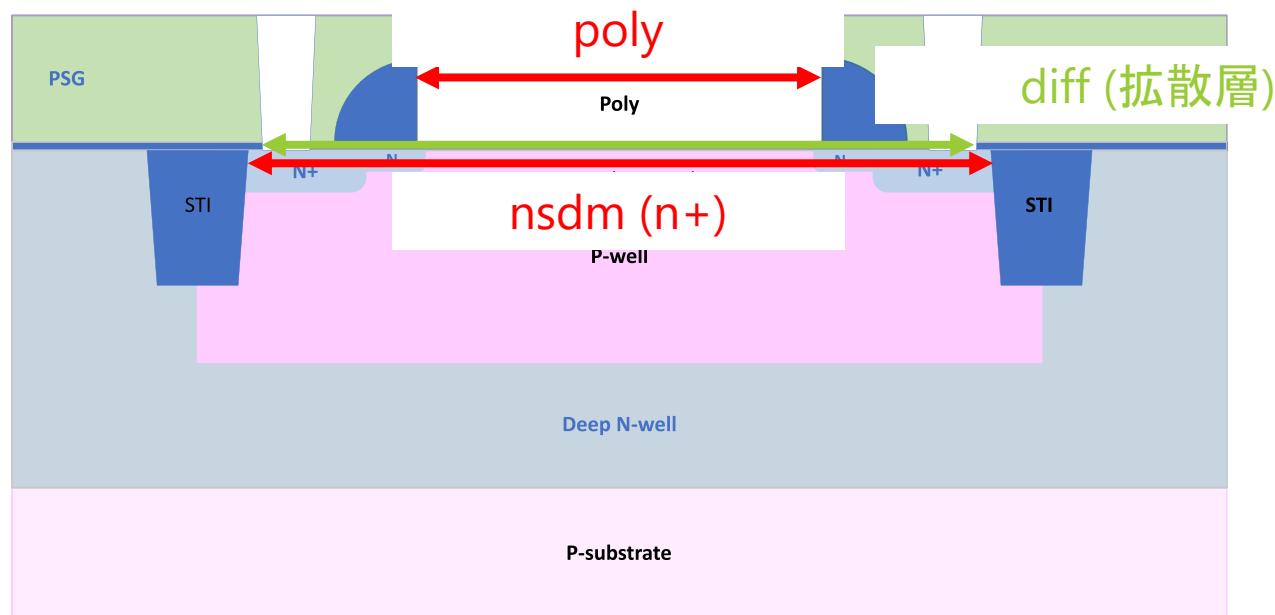
プレーナ型nMOSトランジスタ（トリプルウェル）



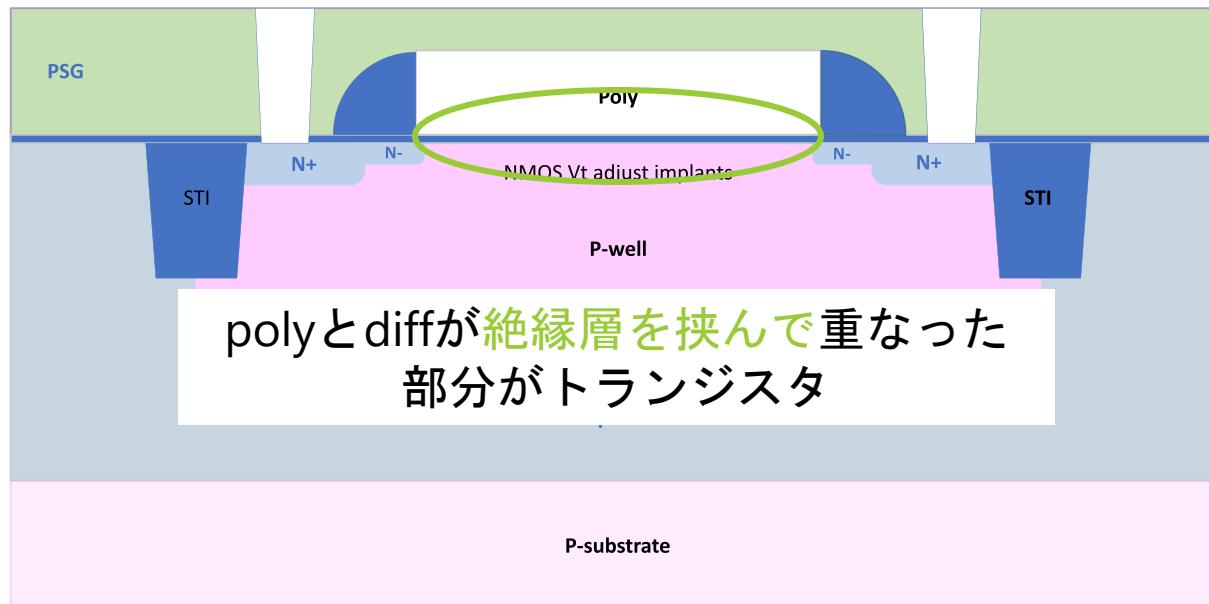
プレーナ型nMOS

diff / active	wellまたはsubstrateと対になるタイプの拡散層
nsdm	不純物注入によるN+イオン領域
poly	ポリシリコン

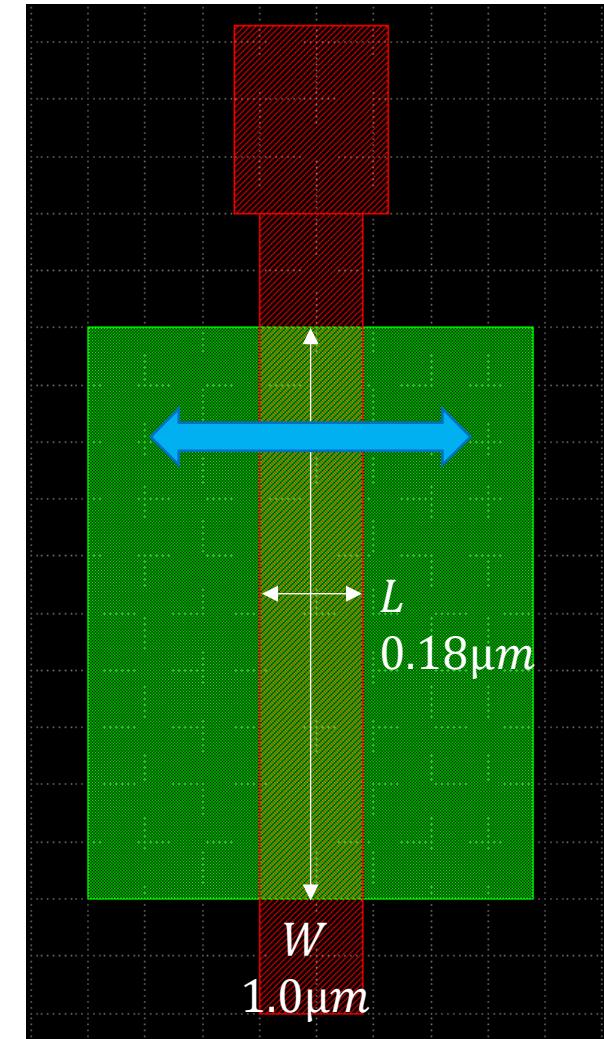
PCELLでは、pwell, dnwellは描かれない



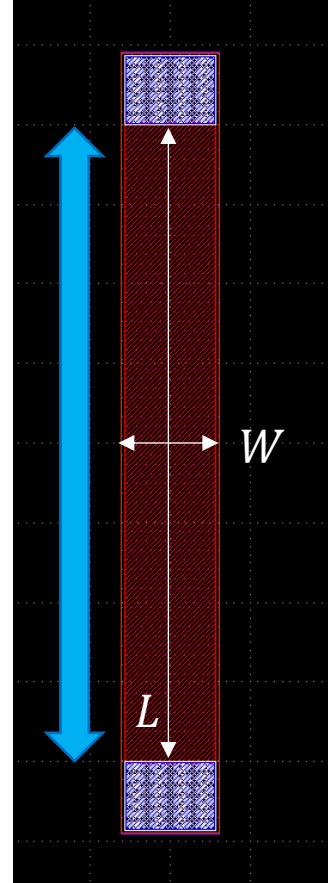
プレーナ型nMOSトランジスタ



電流が流れる方向を基準として
LとWの方向が定まる

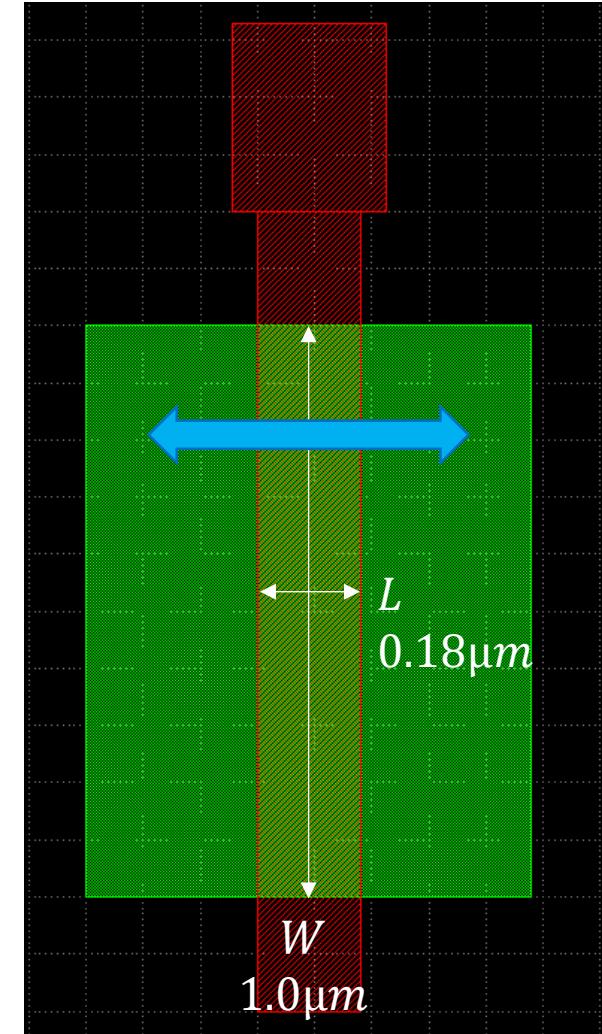


ポリ抵抗



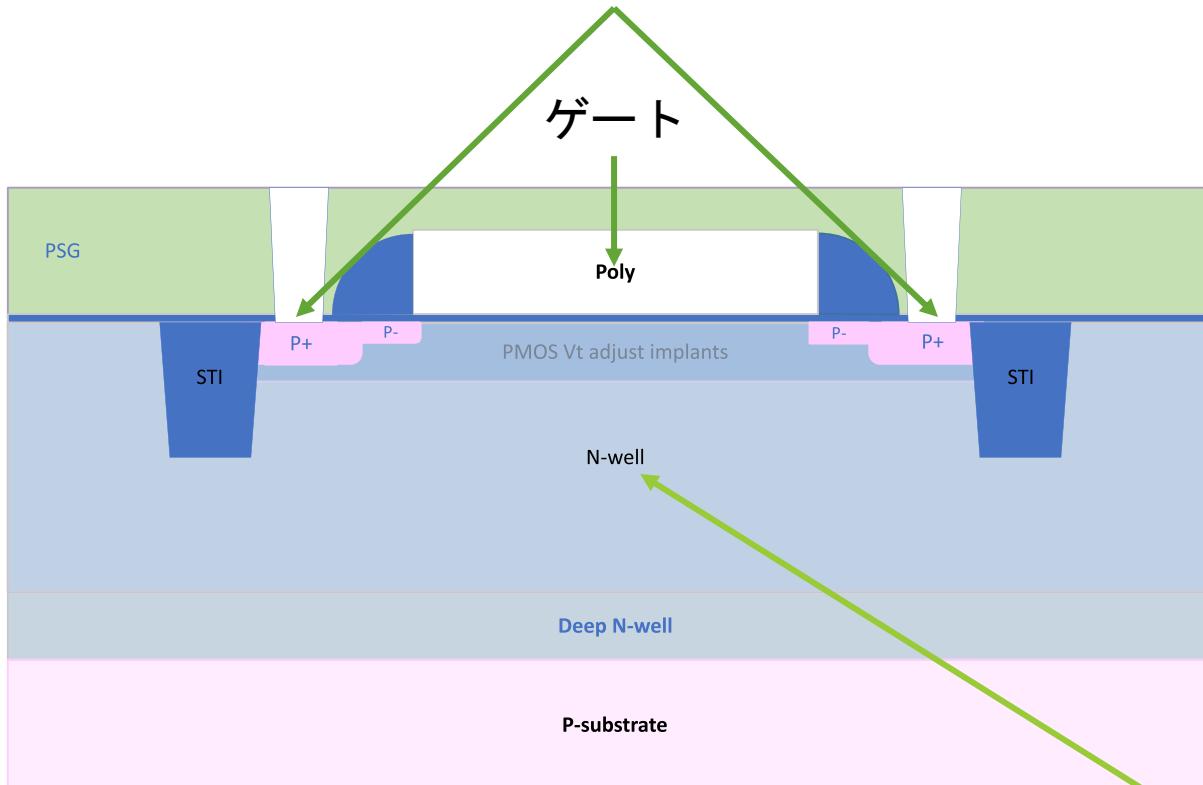
ポリ抵抗のW/Lは
MOSFETと逆なので
要注意

電流が流れる方向を基準として
LとWの方向が定まる

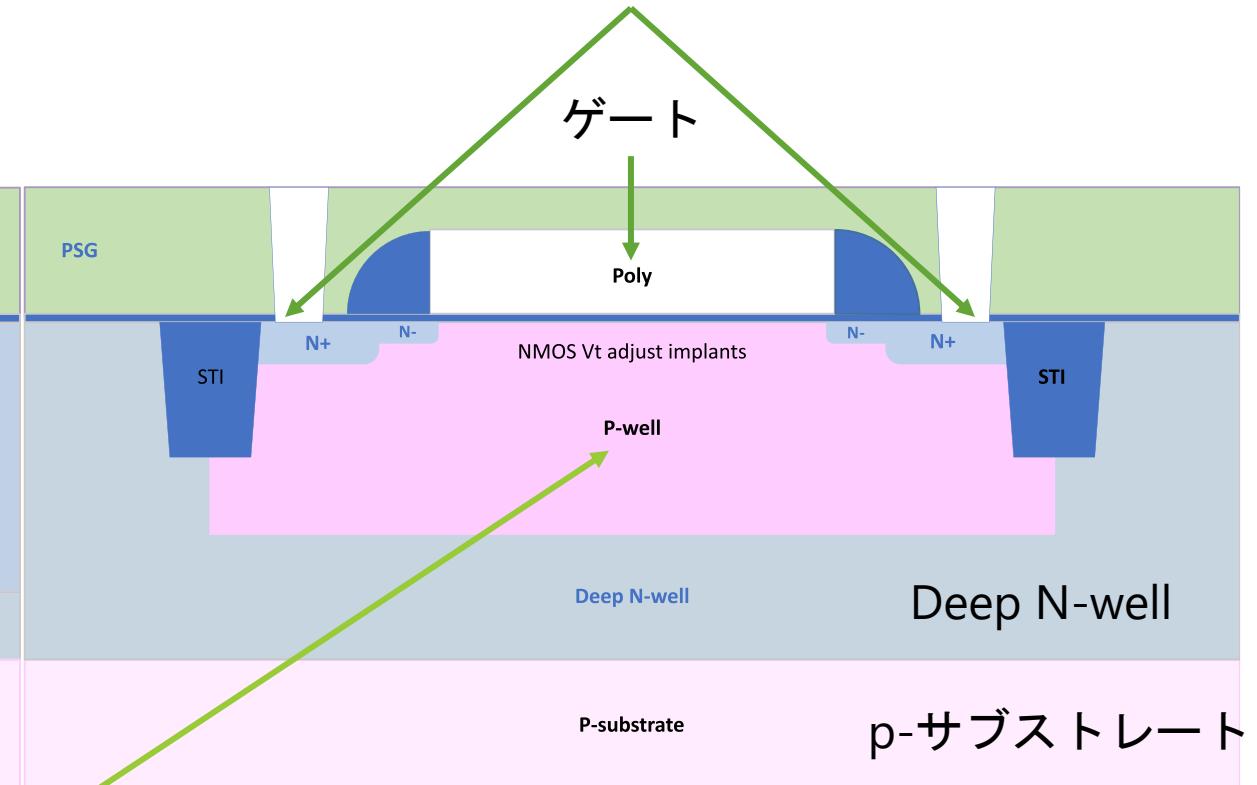


プレーナ型MOSトランジスタ（トリプルウェル）

pMOSでは電圧が高い方がソース、低いほうがドレイン



nMOSでは電圧が高い方がドレイン、低いほうがソース



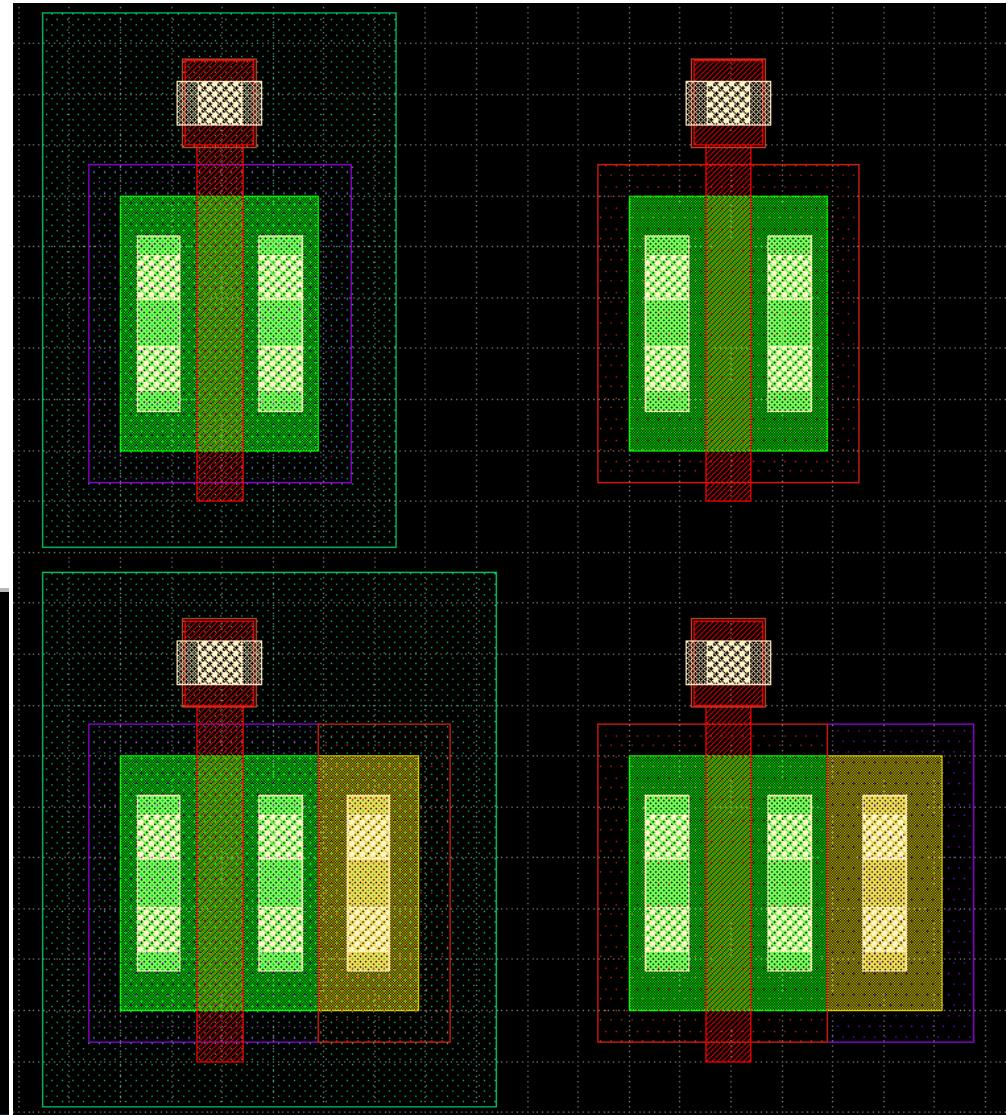
トリプルウェルでは各トランジスタのwellがボディ
nMOSのP-wellがDeep N-wellによりP-Substrateから絶縁されている

ボディの取り出し方: tap

デフォルトのPFET, NFETは3端子でボディの端子がない

ボディを引き出すにはtapというレイヤーを使う

	nwell.drawing - 64/20
	diff.drawing - 65/20
	tap.drawing - 65/44
	psdm.drawing - 94/20
	nsdm.drawing - 93/44
	poly.drawing - 66/20
	licon1.drawing - 66/44
	npc.drawing - 95/20
	ll1.drawing - 67/20



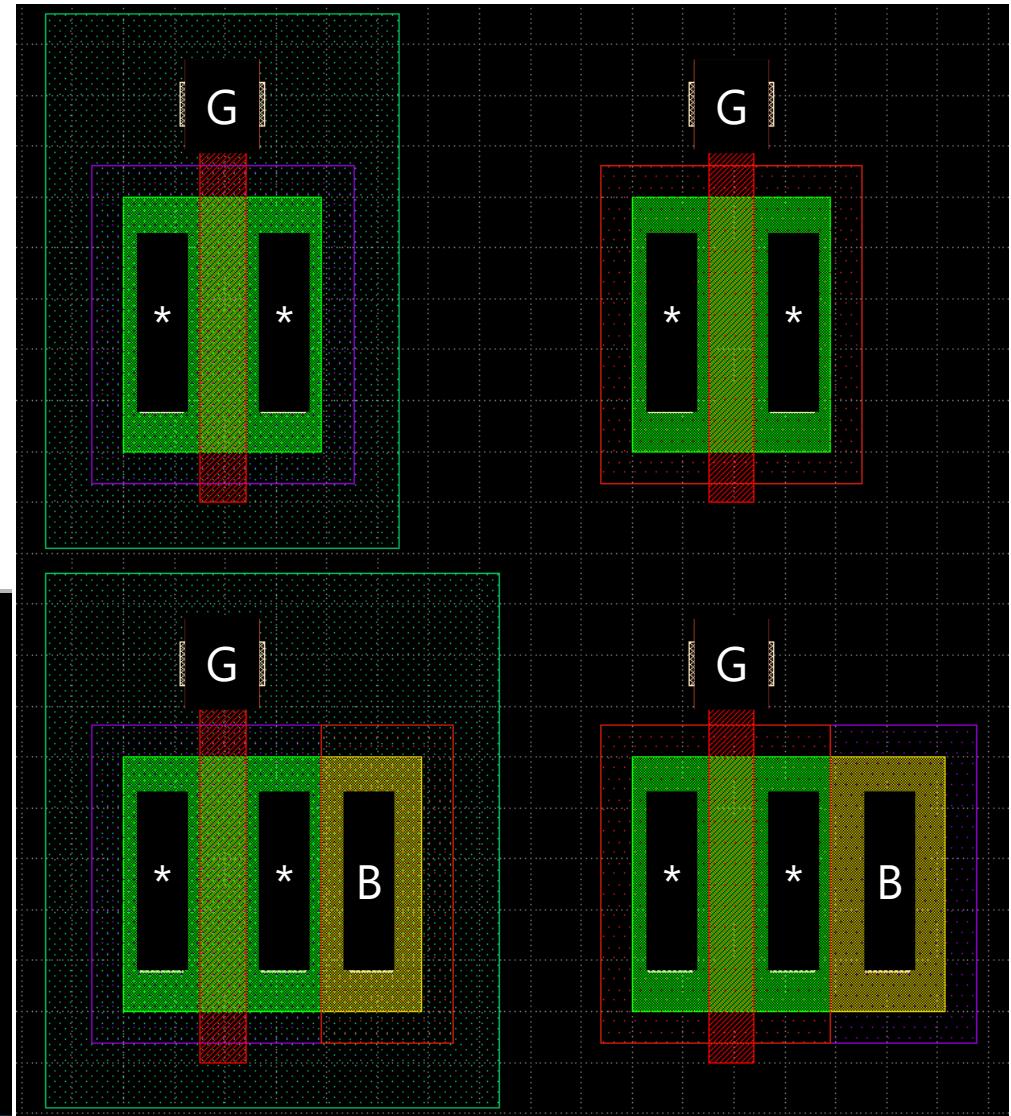
ボディの取り出し方: tap

デフォルトのPFET, NFETは3端子でボディの端子がない

ボディを引き出すにはtapというレイヤーを使う

* = DまたはS

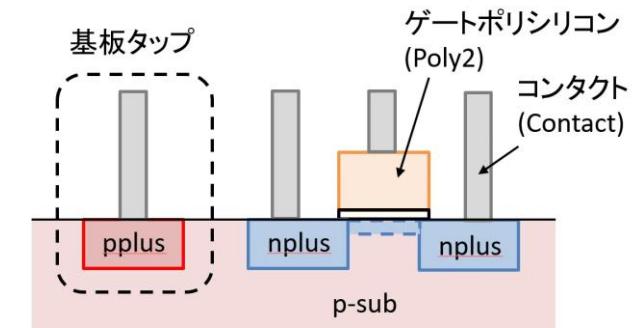
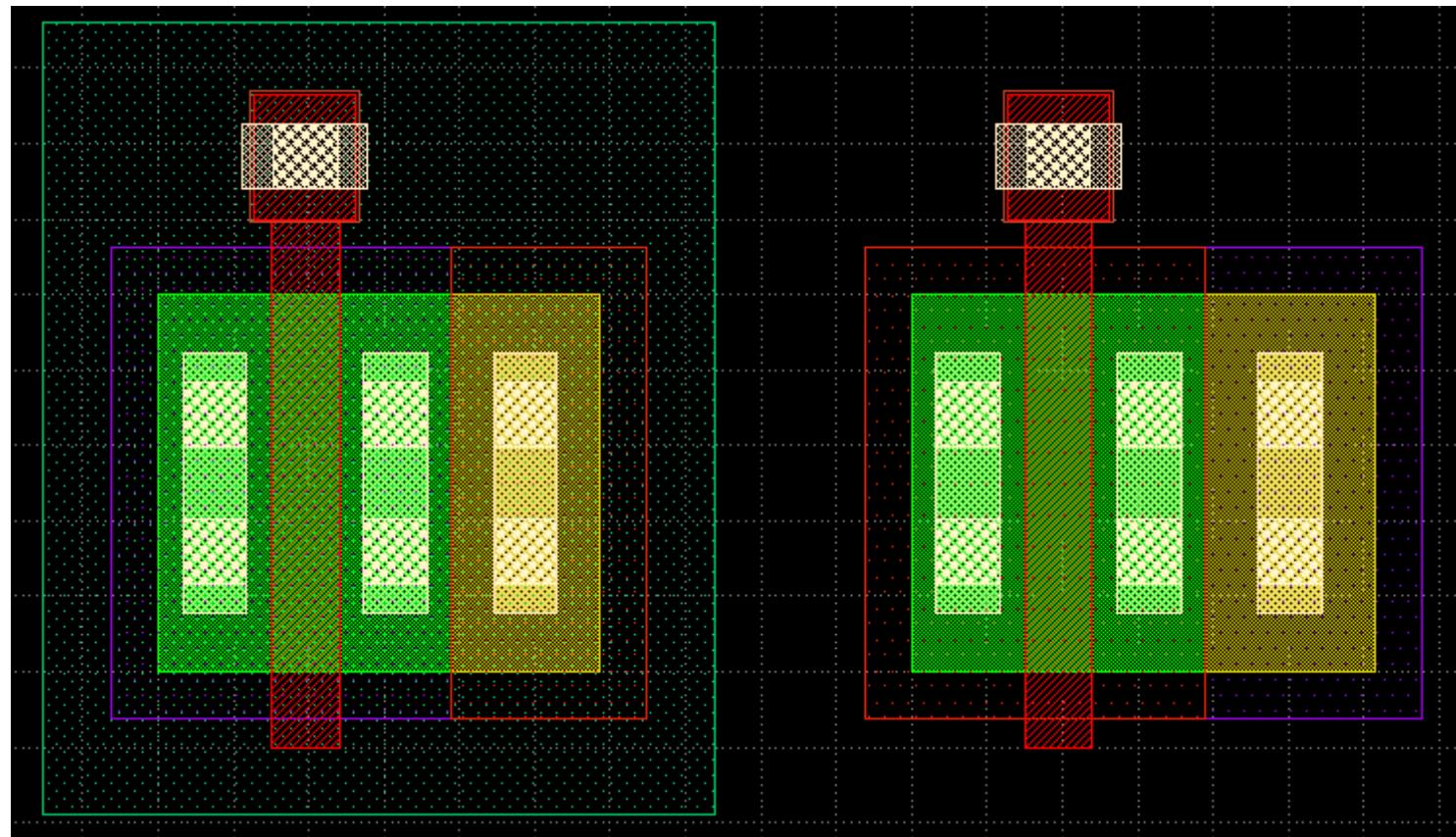
	nwell.drawing - 64/20
	diff.drawing - 65/20
	tap.drawing - 65/44
	psdm.drawing - 94/20
	nsdm.drawing - 93/44
	poly.drawing - 66/20
	licon1.drawing - 66/44
	npc.drawing - 95/20
	ll1.drawing - 67/20



PCELLでのボディの扱い

PFETはN+を介してN-well に繋がっている

NFETはP+を介してP-substrateに繋がっている



https://note.com/akira_tsuchiya/n/nd4444304f013

	nwell.drawing - 64/20
	diff.drawing - 65/20
	tap.drawing - 65/44
	psdm.drawing - 94/20
	nsdm.drawing - 93/44
	poly.drawing - 66/20
	licon1.drawing - 66/44
	npc.drawing - 95/20
	ll1.drawing - 67/20

レイヤーの名前と意味

nwell N-well

diff wellまたはsubstrateと対になるタイプの拡散層

tap wellまたはsubstrateと同タイプの拡散層

psdm 不純物注入によるP+イオン領域

nsdm 不純物注入によるN+イオン領域

poly Polysilicon

licon1 li1層へのコンタクト(VIA)

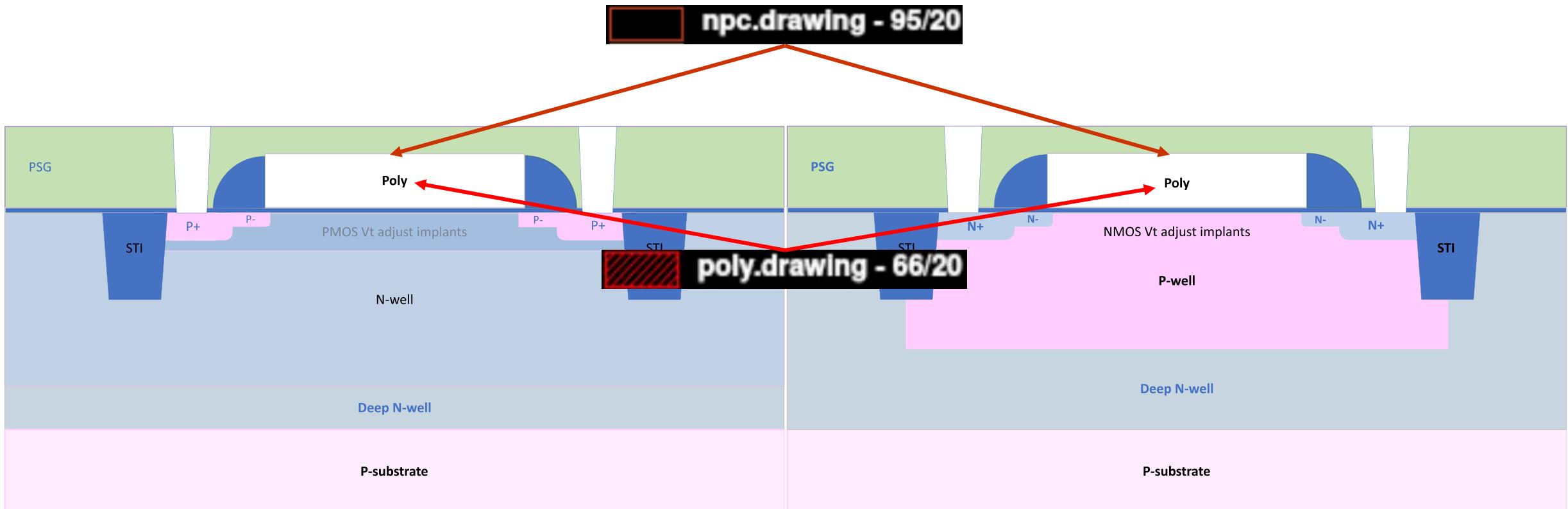
npc Nitride Poly Cut; ポリシリコン露出部

li1 local interconnect; “電極層”

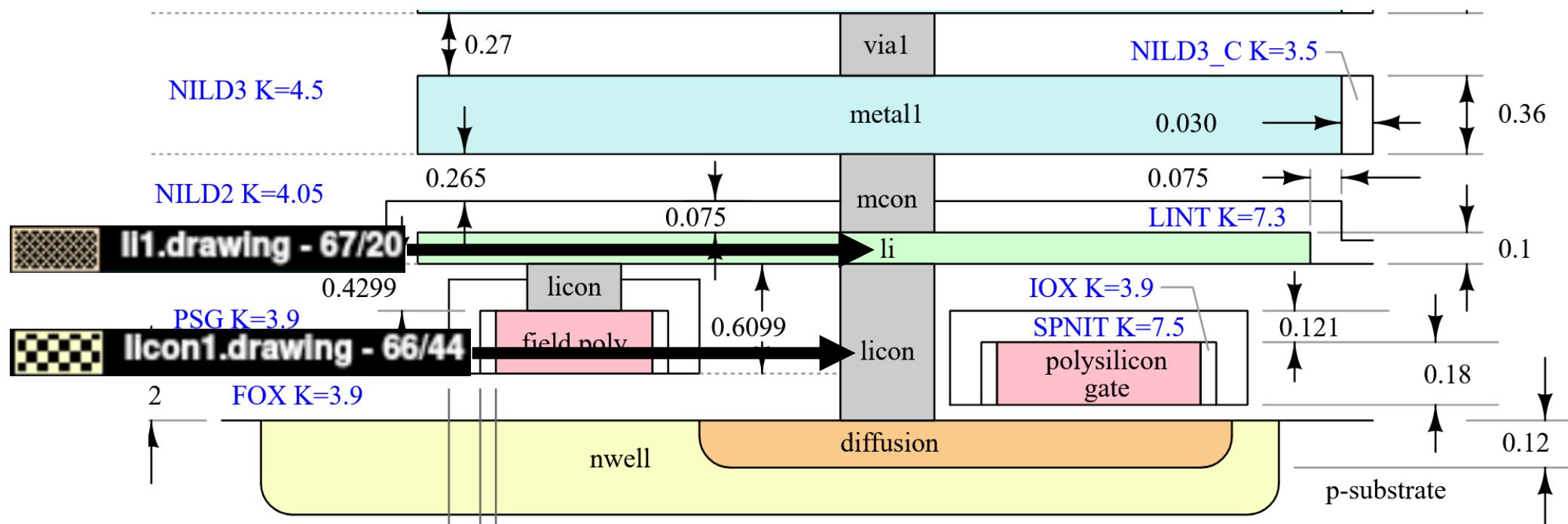


レイヤーの名前と意味

ポリシリコン上部は窒化物（PSGではありません）で絶縁されているが
 npcで明示された領域だけ開口する
 liconを立てる際はnpcを併せて描画する必要がある



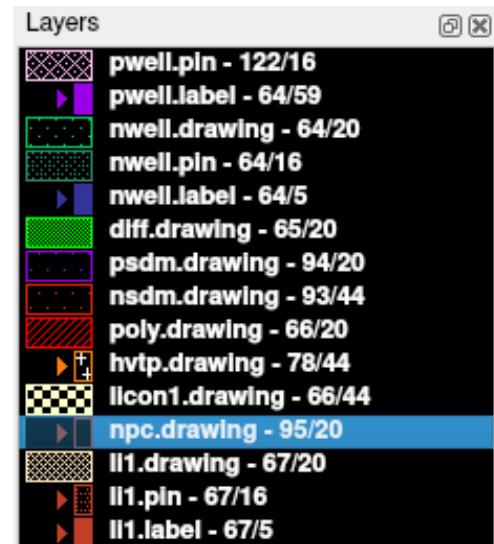
レイヤーの名前と意味



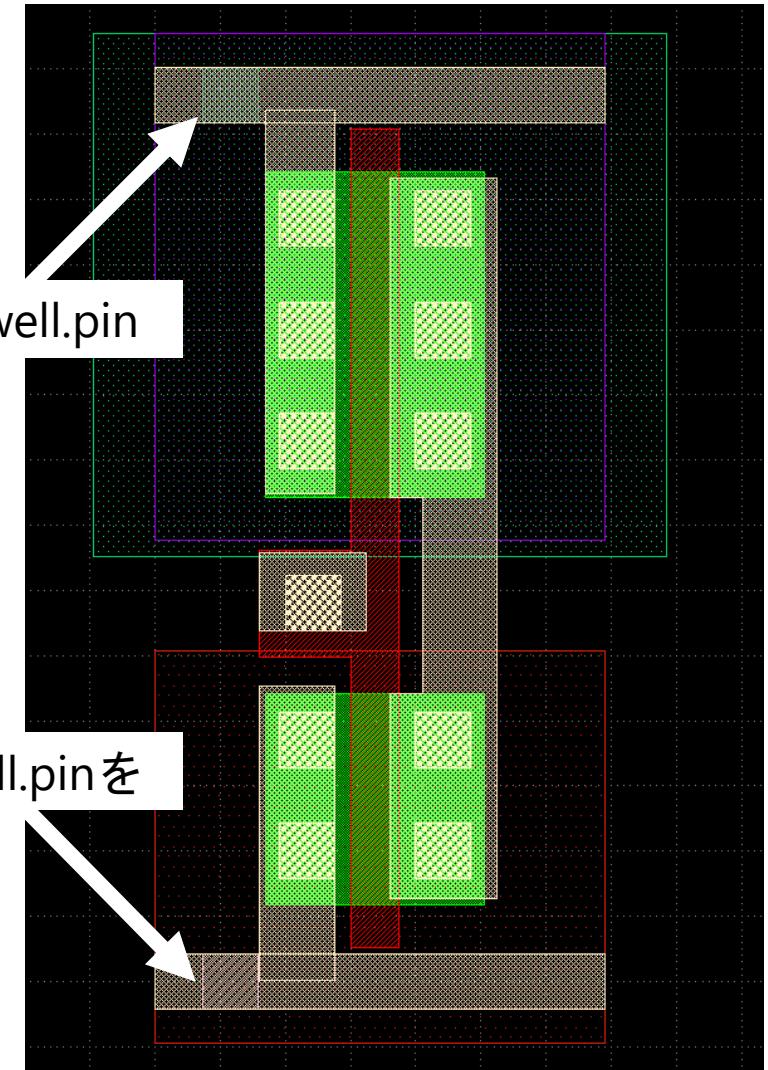
参考:tapless設計でLVSを通すためのピン: well.pin

taplessセルを作成する際は well.pin レイヤを使用することでLVSが通る

LVSの際にサブストレート/ウェルのネット名を指定するためのもので、
実際はtapcellを用意して接続する必要がある。



P substrate でも pwell.pin を
使う



レイアウトを描く

プレーナ型FETプロセス レイアウト編

レイアウトを描く手順

1. LVS用回路図の作成 (Xschem)

- ・電源、コマンド、モデル等のシンボルは削除
- ・VDD, GNDのシンボルはiopinに置き換える

2. レイアウトを描く

- ・慣れないうちは2～3回操作する度にDRCを実行したほうがいい

3. DRC(Design Rules Check)

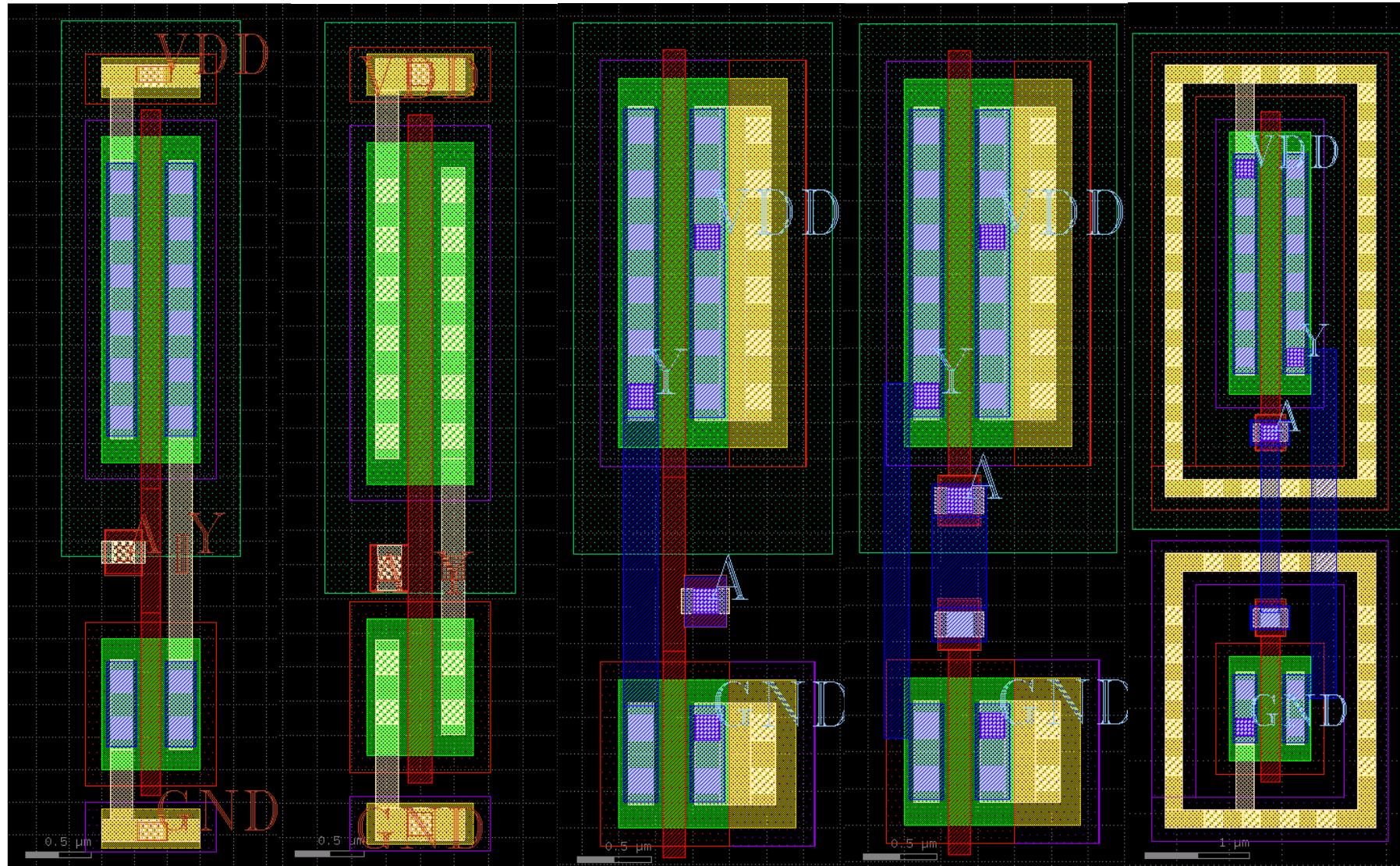
- ・これをパスしないとファウンダリはチップを作ってくれない

4. LVS (Layout Versus Schematic)

- ・完成したと思ったらLVS用回路図と比較して正しく描けているか確認

5. 寄生抽出

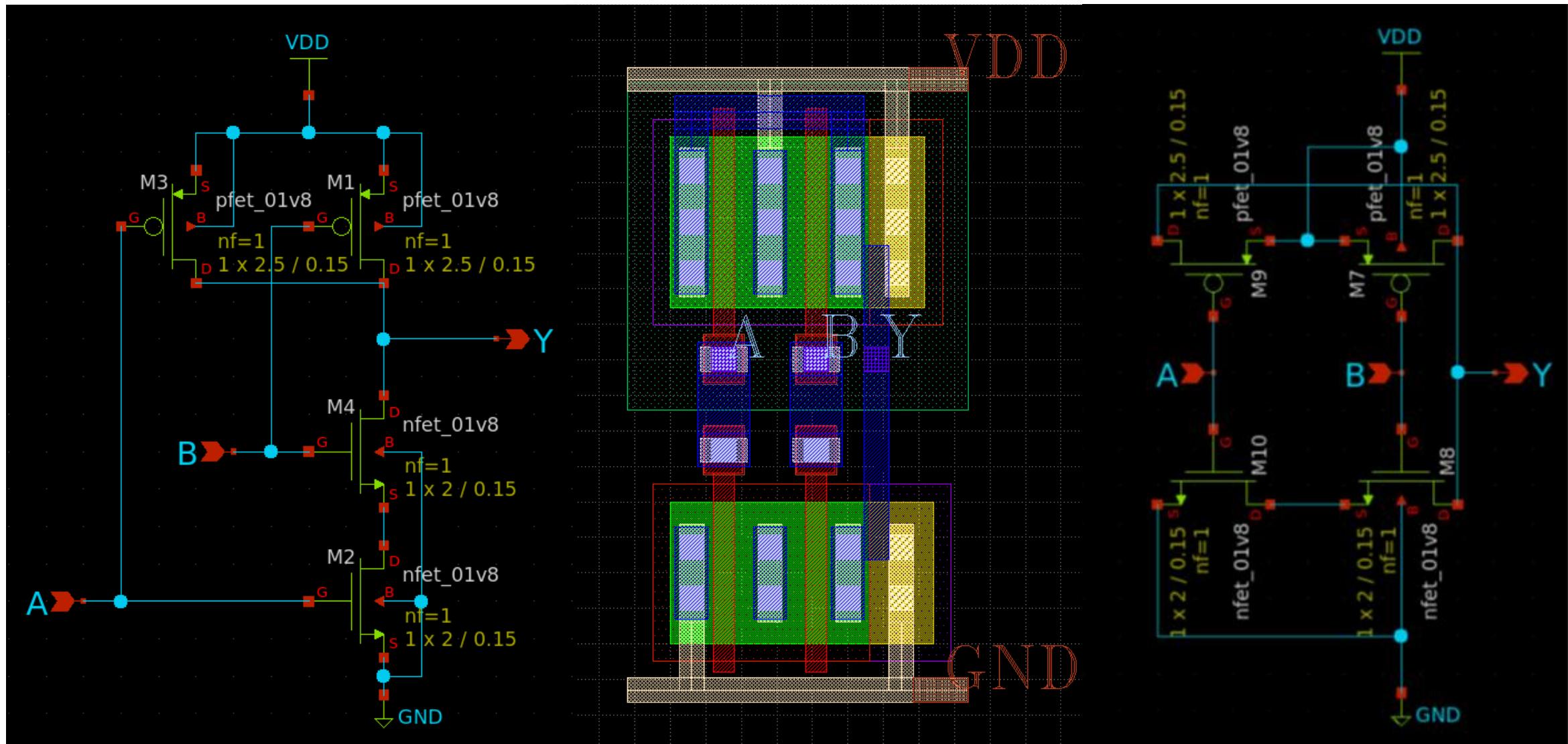
インバータレイアウト例



注意

- デザインルールに則って描く
<https://skywater-pdk.readthedocs.io/en/main/rules/periphery.html>
- Klayout DRC (Caravel) でエラー0の状態にする
- Klayout LVS を実行してレイアウトが回路図と等しい事を確認する
 - 合っていないとポストレイアウトシミュレーションが正常に動かない
 - Magic LVS は実行してもよい

参考：Pcellをベースに作成した2入力NAND



主なデザインルール

SKY130

デザインルール

ここに全部書いてあります

- <https://skywater-pdk.readthedocs.io/en/main/rules/periphery.html>

設計する際はデザインルールを遵守しなければならない！

- DRCエラーが 1 個でもあるとファブは製作してくれない

ここではインバータを描く上で重要なデザインルールを紹介

- トランジスタはPcellが自動で作ってくれるので配線に絞って紹介

GRID / OFFGRID

x.1b: 全レイヤーは $0.005\mu\text{m}$ ステップで描画する必要がある

- 一部レイヤー(met1, via, met2) は $0.001\mu\text{m}$ まで可能ですが (x.1a)、DRCチェックでは全レイヤー $0.005\mu\text{m}$ ステップに統一されています
- 全レイヤーを $0.005\mu\text{m}$ ステップで描写しておけば基本問題ない
- 例えば四角形であれば、四点の座標が $0.005\mu\text{m}$ でないとダメ

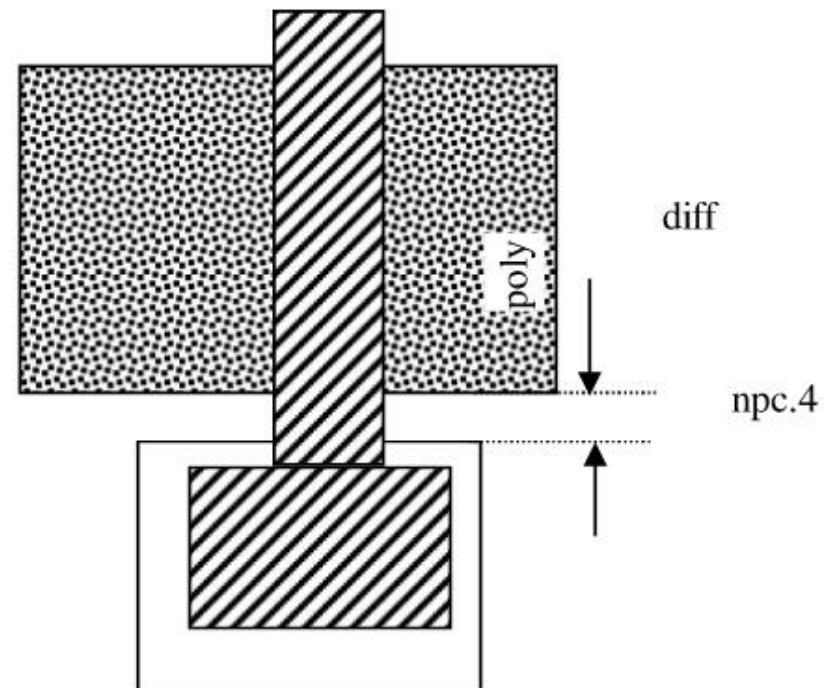
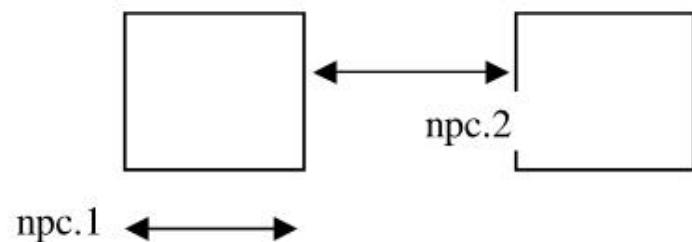
$0.005\mu\text{m}$ より細かい座標を指定すると **OFFGRID** エラーが出る

npc.drawing : Nitride Poly Cut

npc.1 : 最小幅 : 0.270 μm

npc.2 : npc同士で必要な最小スペース : 0.270 μm

npc.4 : npcとdiff/tap間で必要な最小スペース : 0.090 μm



licon1.drawing : Local Interconnect Contact

licon.1 : サイズは $0.170\text{ }\mu\text{m} \times 0.170\text{ }\mu\text{m}$ でなければならない

licon.2 : licon同士で必要な最小スペース : $0.170\text{ }\mu\text{m}$

diff上のliconで必要な最小マージン :

licon.5a : x,y軸のうち、どちらか一軸では : $0.040\text{ }\mu\text{m}$

licon.5c : x,y軸のうち、もう片方の軸では : $0.060\text{ }\mu\text{m}$

tap上のliconで必要な最小マージン :

licon.7 : x,y軸のうち、どちらか一軸では: $0.120\text{ }\mu\text{m}$

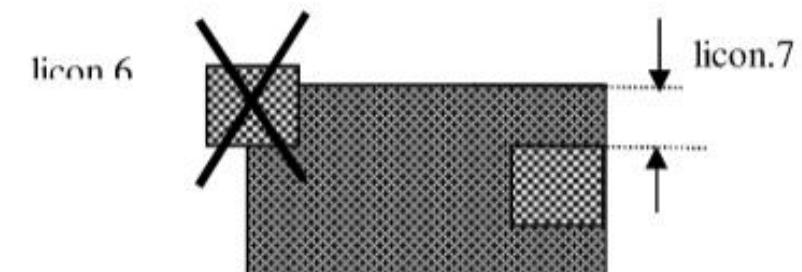
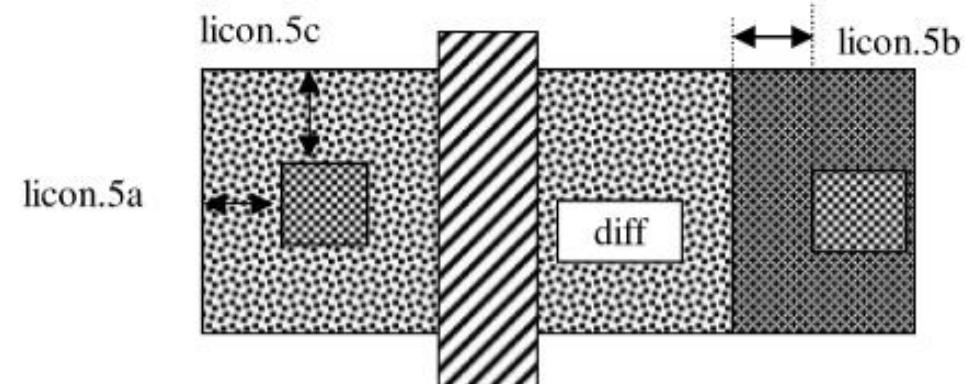
licon.6 : もう片方の軸方向では接触しても良いが、はみ出してはいけない

licon.5b : diffと接触しているtap上のliconで必要なマージン : diffから $0.060\mu\text{m}$

poly上のliconで必要な最小マージン :

licon.8 : x,y軸のうち、どちらか一軸では : $0.050\text{ }\mu\text{m}$

licon.8a : x,y軸のうち、もう片方の軸では : $0.080\text{ }\mu\text{m}$



li1.drawing : Local Interconnect

li.1 : 最小幅 : 0.170 μm

li.2 : liconまたはmconが無い領域での 長/幅 の最大比 : 10 DRCで検知できないルール

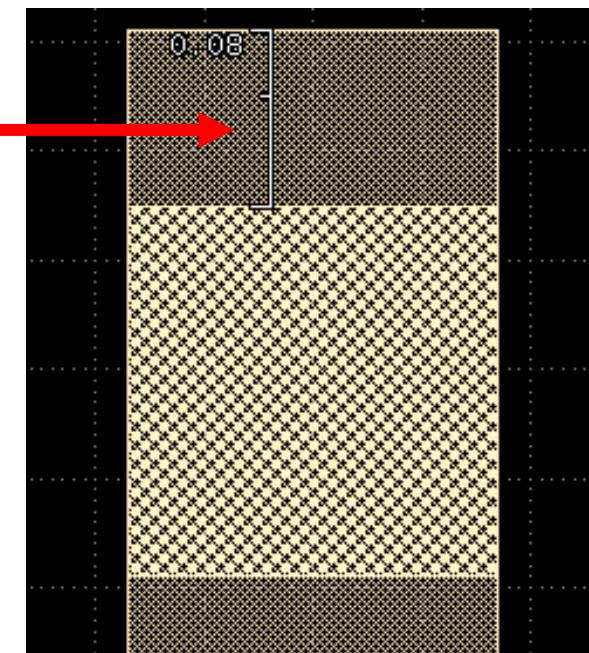
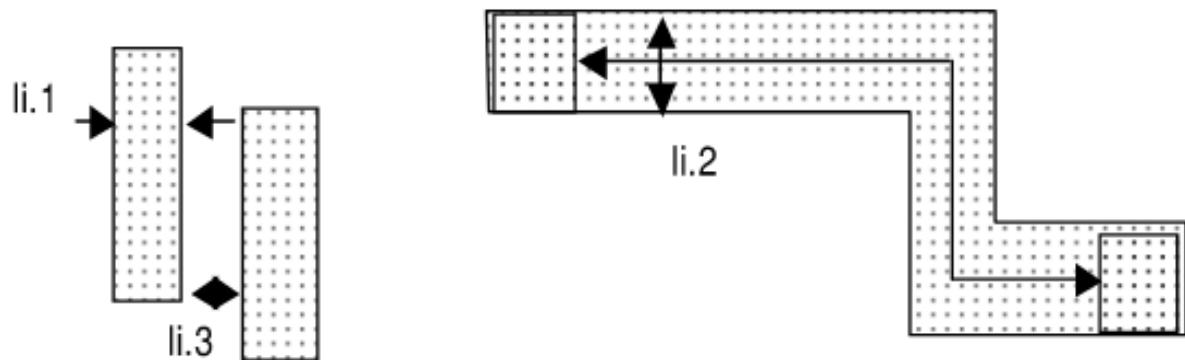
li.3 : li 同士で必要な最小スペース : 0.170μm

li.6 : 最小面積 : 0.0561 μm²

liconとliの最小マージンについて :

li.5 : x,y軸のうち、どちらか一軸では: 0.080 μm

licon.6 : もう片方の軸方向では接触しても良いが、はみ出してはいけない



mcon.drawing : Metal Contact (li1 - met1)

ct.1 : サイズは $0.170\text{ }\mu\text{m} \times 0.170\text{ }\mu\text{m}$ でなければならない

ct.2 : mcon同士で必要な最小スペース : $0.190\text{ }\mu\text{m}$

liとmconの最小マージンについて :

ct.4 : mconはli1.drawing上からはみ出してはいけない ($0.000\text{ }\mu\text{m}$)

mconとmet1の最小マージンについて :

m1.4 : x,y軸のうち、どちらか一軸では : $0.030\text{ }\mu\text{m}$

m1.5 : x,y軸のうち、もう片方の軸では : $0.060\text{ }\mu\text{m}$

met1/2/3.drawing : Metal

met1.drawing :

m1.1 : 最小幅 : 0.140 μm

m1.2 : met1同士で必要な最小スペース : 0.140 μm

m1.6 : 最小面積 : 0.083 μm^2

met2.drawing :

m2.1 : 最小幅 : 0.140 μm

m2.2 : met2同士で必要な最小スペース : 0.140 μm

m2.6 : 最小面積 : 0.0676 μm^2

met3.drawing :

m3.1 : 最小幅 : 0.300 μm

m3.2 : met3同士で必要な最小スペース : 0.300 μm

m3.6 : 最小面積 : 0.240 μm^2

via.drawing : Via (met1 - met2)

via.1 : サイズは $0.150\text{ }\mu\text{m} \times 0.150\text{ }\mu\text{m}$ でなければならない

via.2 : via同士で必要な最小スペース : $0.170\text{ }\mu\text{m}$

met1とviaの最小マージンについて :

via.4 : x,y軸のうち、どちらか一軸では : $0.055\text{ }\mu\text{m}$

via.5 : x,y軸のうち、もう片方の軸では : $0.085\text{ }\mu\text{m}$

viaとmet2の最小マージンについて :

m2.4 : x,y軸のうち、どちらか一軸では : $0.055\text{ }\mu\text{m}$

m2.5 : x,y軸のうち、もう片方の軸では : $0.085\text{ }\mu\text{m}$

via2.drawing : Via (met2 – met3)

via2.1 : サイズは $0.200\text{ }\mu\text{m} \times 0.200\text{ }\mu\text{m}$ でなければならない

via2.2 : via同士で必要な最小スペース : $0.200\text{ }\mu\text{m}$

met2とvia2の最小マージンについて :

via2.4 : x,y軸のうち、どちらか一軸では : $0.040\text{ }\mu\text{m}$

via2.5 : x,y軸のうち、もう片方の軸では : $0.085\text{ }\mu\text{m}$

via2とmet3の最小マージンについて :

m3.4 : 四方で $0.065\text{ }\mu\text{m}$

ポストレイアウトシミュレーション

SKY130

レイアウト検証 : PEX, Parasitic Extraction

RC抽出

寄生抵抗と寄生容量を含むネットリストを抽出

```
File Edit View Search Terminal Help
Nets output: 4 (1.000000)
exttospice finished.
* NGSPICE file created from TOP.ext - technology: sky130A

.subckt TOP A Y VDD GND
X0 Y.t1 A.t0 VDD.t1 VDD.t0 sky130_fd_pr_pfet_01v8 ad=0p pd=0u as=0p ps=0u w=2.5
e+06u l=180000u
X1 Y.t0 A.t1 GND.t1 GND.t0 sky130_fd_pr_nfet_01v8 ad=0p pd=0u as=0p ps=0u w=1e+
06u l=180000u
R0 A.n0 A.t0 472.895
R1 A.n0 A.t1 207.794
R2 A A.n0 79.296
R3 VDD VDD.t0 1385.67
R4 VDD VDD.t1 254.38
R5 Y Y.t1 270.032
R6 Y Y.t0 134.44
R7 GND GND.t0 4820.4
R8 GND GND.t1 150.035
C0 Y A 0.05fF
C1 A VDD 0.18fF
C2 Y VDD 0.17fF
.ends
```

C抽出

寄生容量のみを含むネットリストを抽出

```
File Edit View Search Terminal Help
Loading sky130A Device Generator Menu ...
Loading "/home/user/.klayout/macros/sky130_magic_pex.tcl" from command line.
Warning: Calma reading is not undoable! I hope that's OK.
Library written using GDS-II Release 6.0
Library name: LIB
Reading "TOP".
CIF file read warning: CIF style sky130(): units rescaled by factor of 5 / 1
Extracting TOP into TOP.ext:
exttosim finished.
exttospice finished.
exttospice finished.
* NGSPICE file created from TOP.ext - technology: sky130A

.subckt TOP A Y VDD GND
X0 Y A VDD VDD sky130_fd_pr_pfet_01v8 ad=7.5e+11p pd=5.6e+06u as=7.5e+11p ps=5.
6e+06u w=2.5e+06u l=180000u
X1 Y A GND GND sky130_fd_pr_nfet_01v8 ad=3e+11p pd=2.6e+06u as=3e+11p ps=2.6e+0
6u w=1e+06u l=180000u
C0 A Y 0.05fF
C1 VDD Y 0.17fF
C2 A VDD 0.18fF
.ends
```

PEX前の下準備 : Flatten

1. コピーを保存する
 - ファイル名で大文字を使うとバグが発生する可能性あり
 - 拡張子がデフォルトだと.GDSなので.gdsに直す
2. ドラッグでレイアウト全体を選択する
3. Edit > Flatten Cell > Flatten Instances を All hierarchy levels で実行
4. Cells が TOP のみになっていることを確認して Save

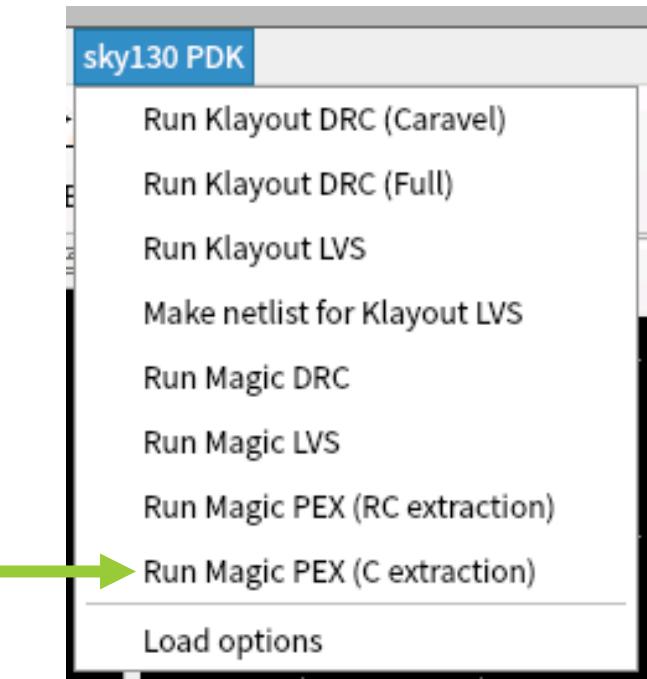
寄生抽出

Magic PEX を用いて寄生容量抽出を行う

- Run Magic PEX (C extraction) を実行

実行すると寄生容量付きネットリストが生成される

- ファイル名 : {レイアウト名}_pex_extracted.spice
 - デフォルトではTOP_pex_extracted.spice

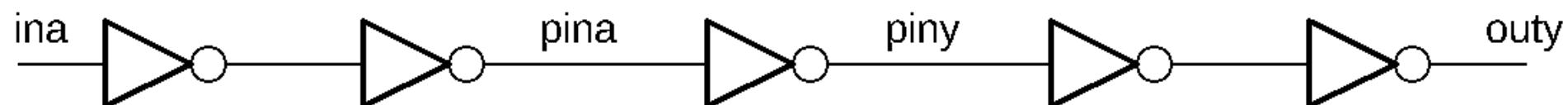


遅延時間シミュレーションの条件例

インバータ設計

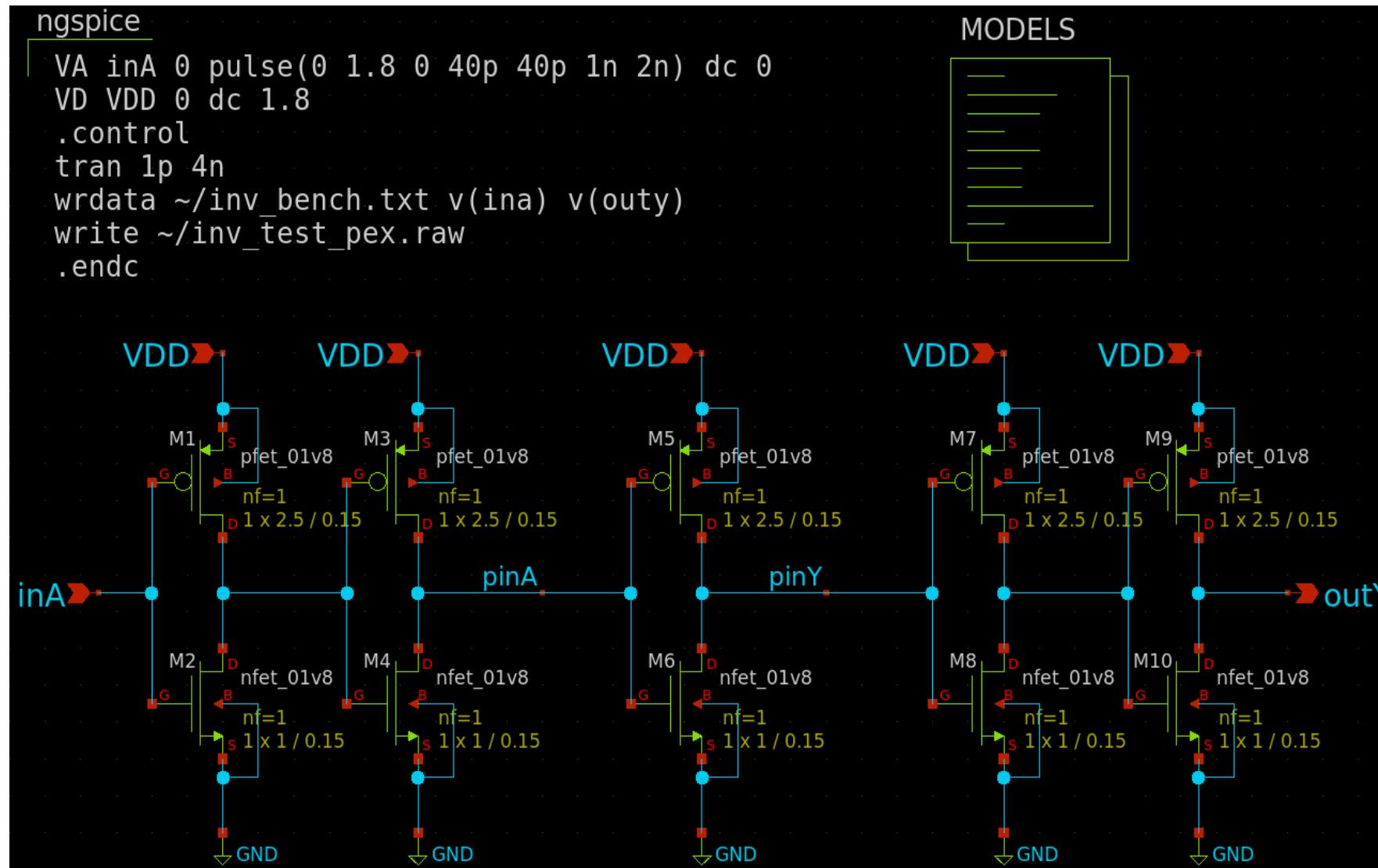
$$P=2.5u / N=1u$$

$$P=2.5u / N=1u$$



- 初段バッファサイズは $w_p = 2.5 \mu\text{m}$, $w_n = 1 \mu\text{m}$
- 出力段の負荷容量は入力容量と同じ $w_p = 2.5 \mu\text{m}$, $w_n = 1 \mu\text{m}$
- 'ina' から 'outy' までの遅延時間を求める

理想遅延時間を計測するテストベンチの例



寄生容量を考慮したテストベンチに改造する

ポストシミュレーションの際は以下 2 点の変更が必要

1. 外部ネットリストへのパス
2. 外部ネットリスト用のシンボル

外部ネットリストへのパスを通す

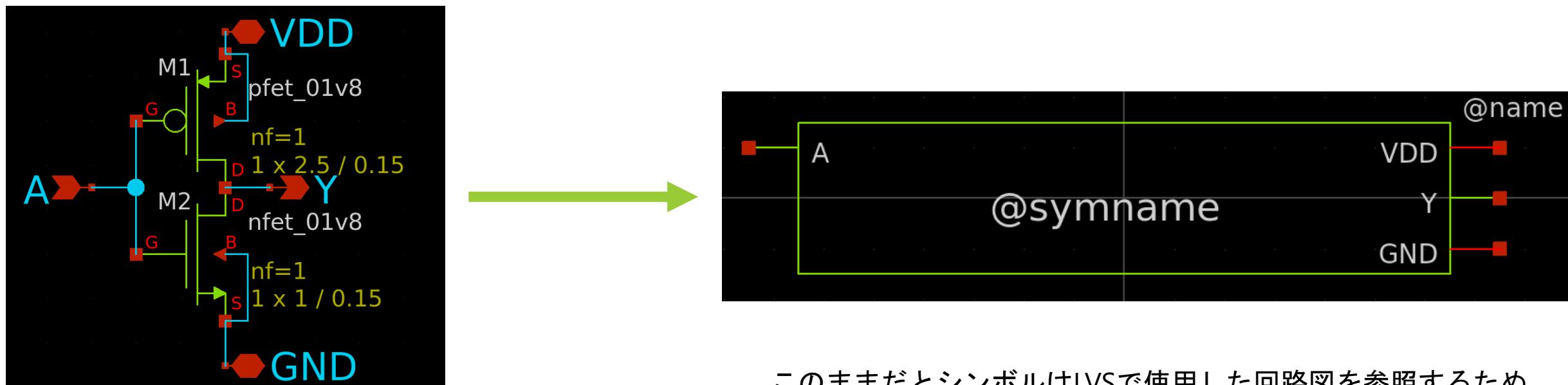
- シミュレーションスクリプトに.includeで寄生抽出ネットリストのパスを通す
 - レイアウト(.gds)と同じ場所に生成される

```
ngspice
  VA inA 0 pulse(0 1.8 0 40p 40p 1n 2n) dc 0
  VD VDD 0 dc 1.8
→ .include ~/TOP_pex_extracted.spice
  .control
    tran 1p 4n
    wrdata ~/inv_bench.txt v(ina) v(outy)
    write ~/inv_test_pex.raw
  .endc
```

外部ネットリスト用のシンボル作成

LVSで使用した回路図からシンボルを作成する

- メニュー Symbol > Make symbol from schematic



このままだとシンボルはLVSで使用した回路図を参照するため
ネットリストを参照するように改変する

外部ネットリスト用のシンボル作成

シンボルのプロパティを開いてこんな感じの中身にする (基本形)

- Qキーで開く

```
type=primitive
format="@name [ネットリスト内のピン] @prefix"
template="name=x1 prefix=[subcktの名前]"
extra="prefix"
highlight=true
```

詳しくは http://repo.hu/projects/xschem/xschem_man/symbol_property_syntax.html を参照

外部ネットリスト用のシンボル作成

インバータならこんな感じ

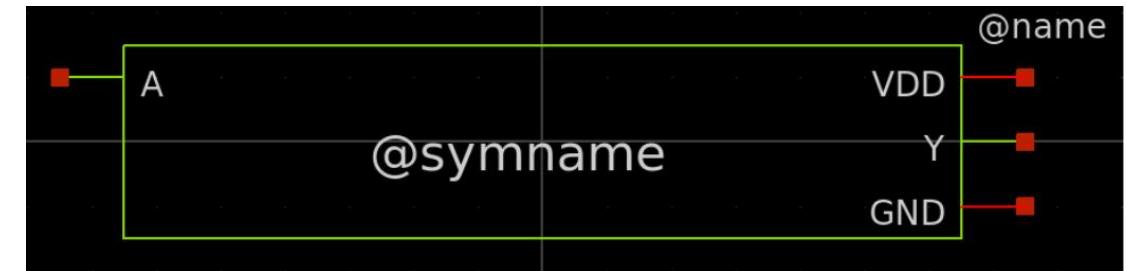
```
type=primitive  
format="@name @@A @@Y @@VDD @@GND @prefix"  
template="name=x1 prefix=TOP"  
extra="prefix"  
highlight=true
```

抽出したネットリストのピンと同じ並びで記述する

```
.subckt TOP A Y VDD GND  
X0 Y A VDD VDD sky130_fd_pr_pfe  
X1 Y A GND GND sky130_fd_pr_nfe  
C0 Y A 0.05fF  
C1 Y VDD 0.17fF  
C2 A VDD 0.18fF  
.ends
```

外部ネットリスト用のシンボル作成

インバータならこんな感じ



```
type=primitive  
format="@name @@A @@Y @@VDD @@GND @prefix"  
template="name=x1 prefix=TOP"  
extra="prefix"  
highlight=true
```

シンボルから出ているピンには"@@"のprefixをつける
これで一応完成

外部ネットリスト用のシンボル作成（応用編）

シンボルのプロパティを開いてこんな感じの中身にする（応用形）

- Qキーで開く

```
type=primitive
format="@name [ネットリスト内のピン] @prefix"
template="name=x1 [プロパティでネットを指定するピン] prefix=[subcktの名前]"
extra=" [プロパティでネットを指定するピン] prefix"
highlight=true
```

詳しくは http://repo.hu/projects/xschem/xschem_man/symbol_property_syntax.html を参照

外部ネットリスト用のシンボル作成

VDDとGNDのピンをシンボルから削除してプロパティで記述してみる

```
type=primitive  
format="@name @@A @@Y @VDD @GND @prefix"  
template="name=x1 VDD=VDD GND=0 prefix=TOP"  
extra="VDD GND prefix"  
highlight=true
```

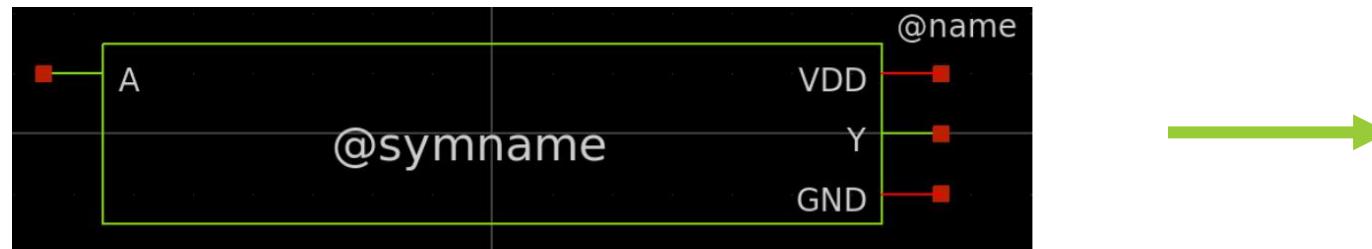
プロパティでピンのネットを指定する場合は "@" の prefix を付けて、
extra にピン、 template にデフォルトの値を追加する

外部ネットリスト用のシンボル作成

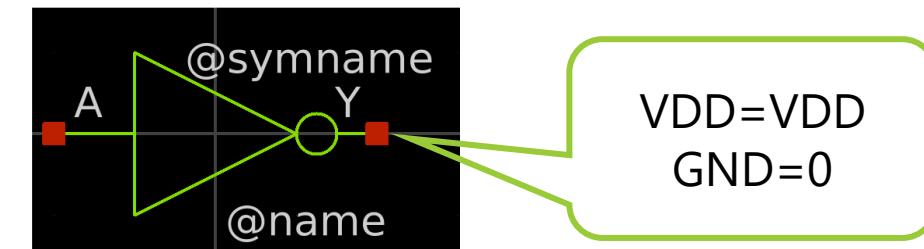
inv.sym

VDDとGNDのピンをシンボルから削除してプロパティで記述してみる

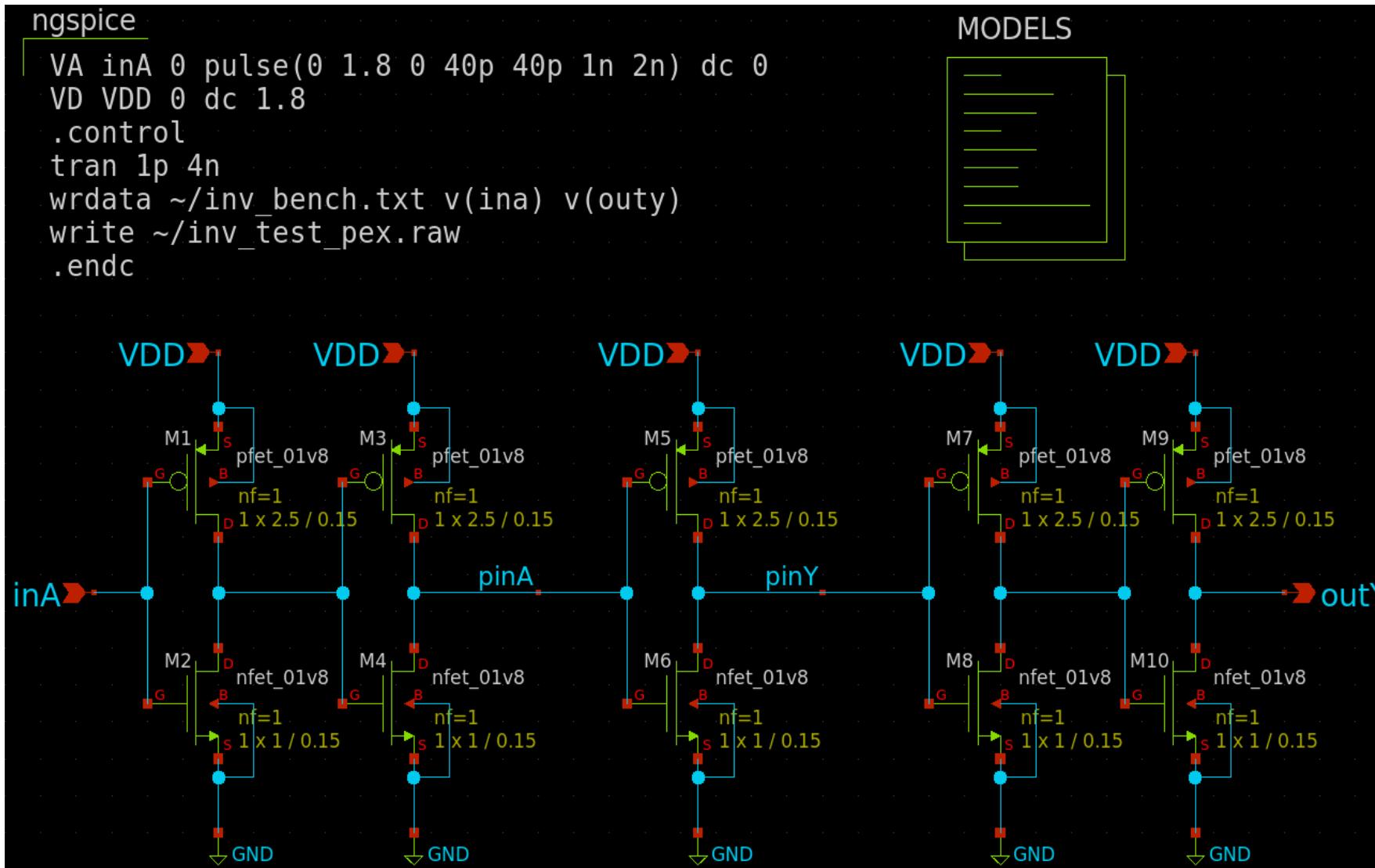
```
type=primitive  
format="@name @@A @@Y @VDD @GND @prefix"  
template="name=x1 VDD=VDD GND=0 prefix=TOP"  
extra="VDD GND prefix"  
highlight=true
```



テストベンチが
見やすくなる



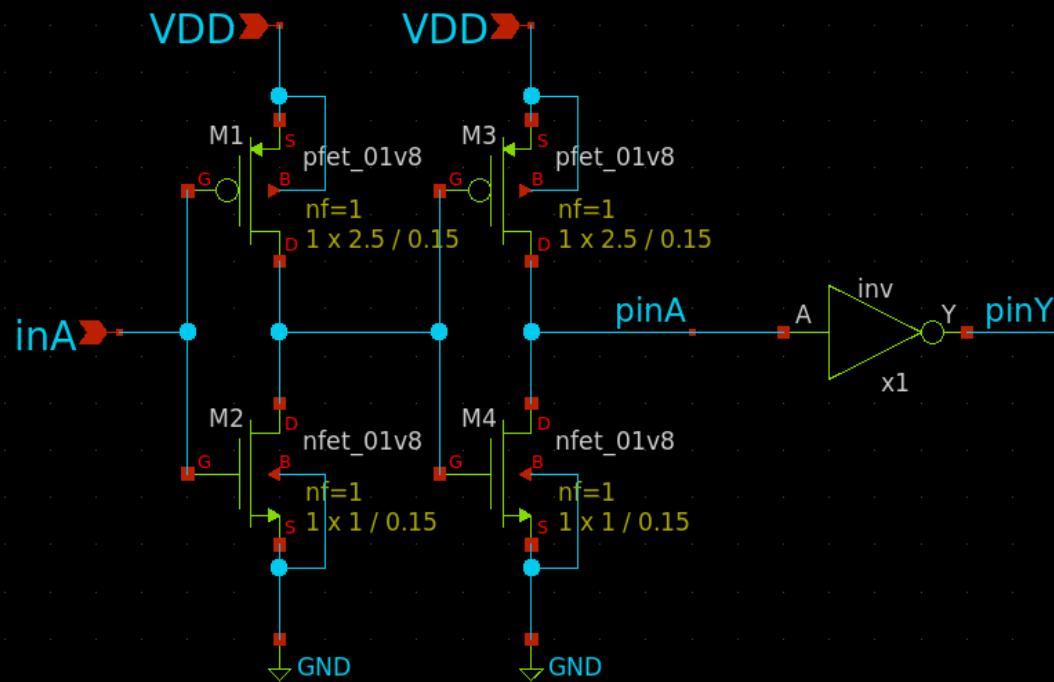
テストベンチ改造 Before



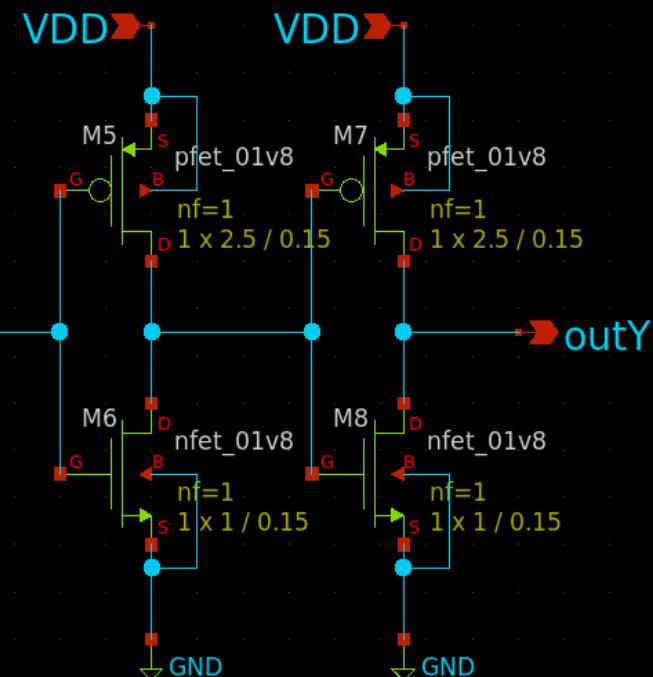
テストベンチ改造 After

inv_pex.sch

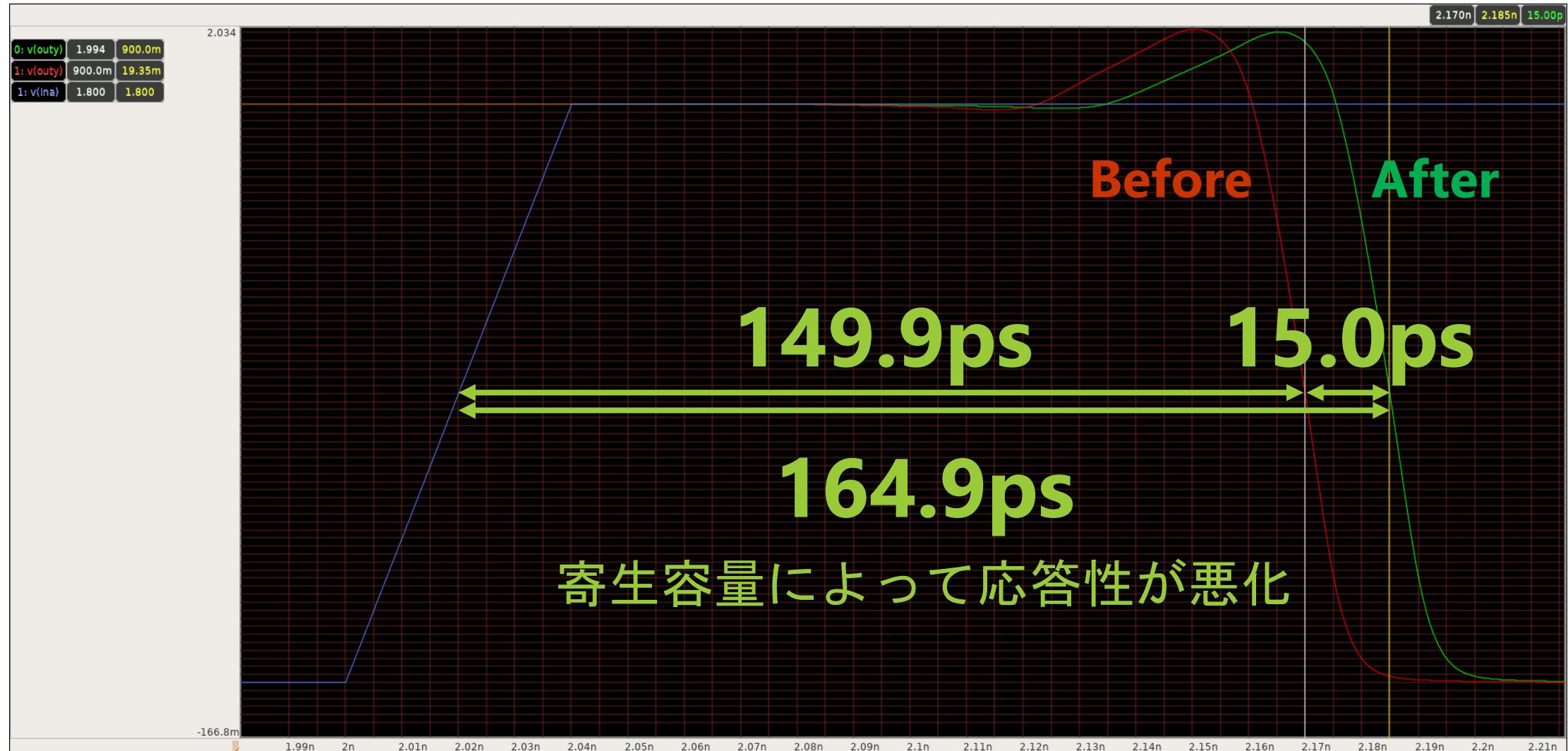
```
ngspice_
VA inA 0 pulse(0 1.8 0 40p 40p 1n 2n) dc 0
VD VDD 0 dc 1.8
.include ~/TOP_pex_extracted.spice
.control
.tran 1p 4n
.wrdata ~/inv_bench.txt v(ina) v(outy)
.write ~/inv_test_pex.raw
.endc
```



MODELS



シミュレーション結果



小夜夕集

応用編

テクニック集

- ・マルチフィンガーとミスマッチシミュレーション
- ・モンテカルロシミュレーション
- ・スイッチについて
- ・DラッチとDフリップフロップ

マルチフィンガーについて

レイアウトテクニック

参考：スタンダードセルにおけるバッファの構成

buf_2 1→2

buf_4 1→4

buf_8 3→8

clkbuf8 2→8

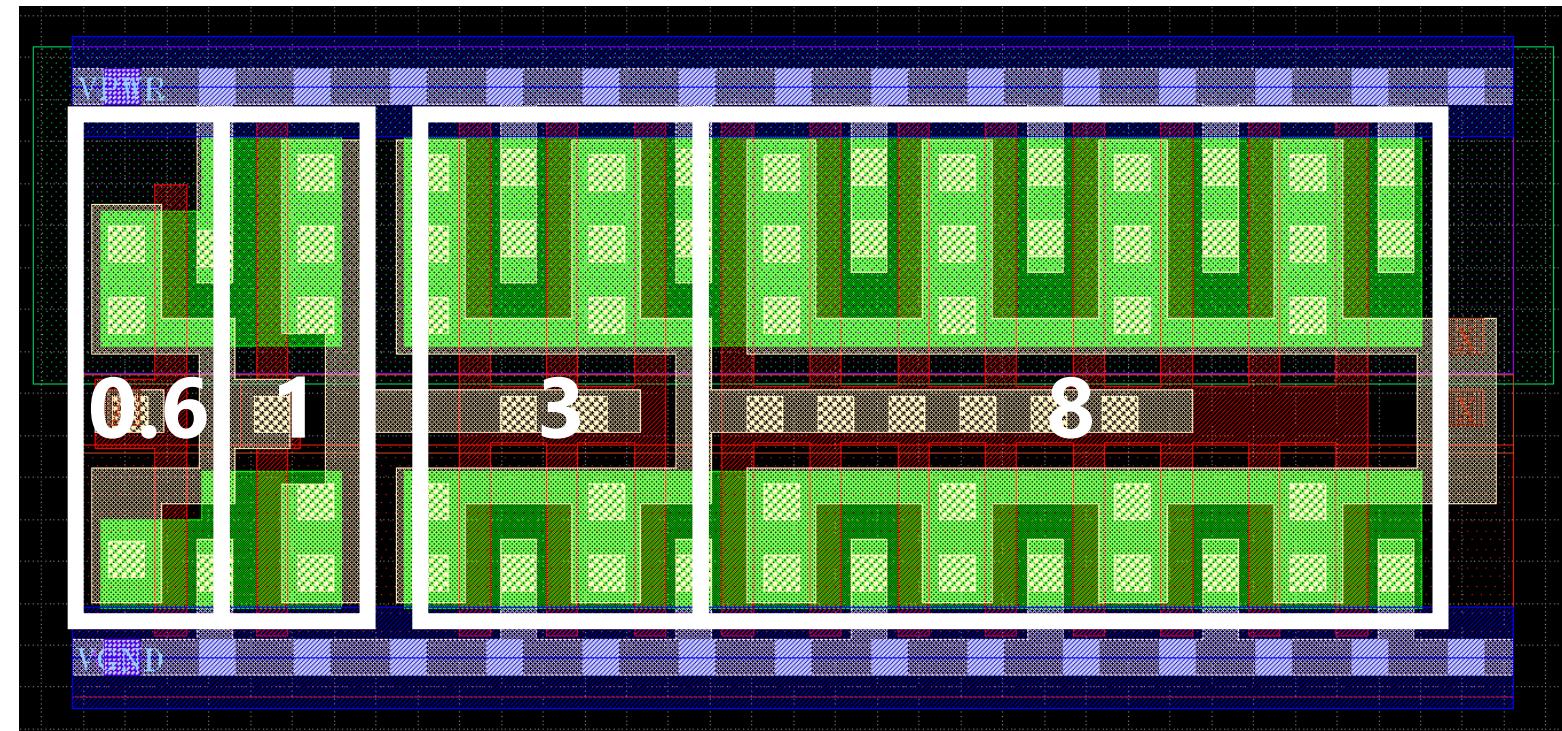
bufbuf8 0.6→1→3→8

buf_12 4→12

buf_16 6→16

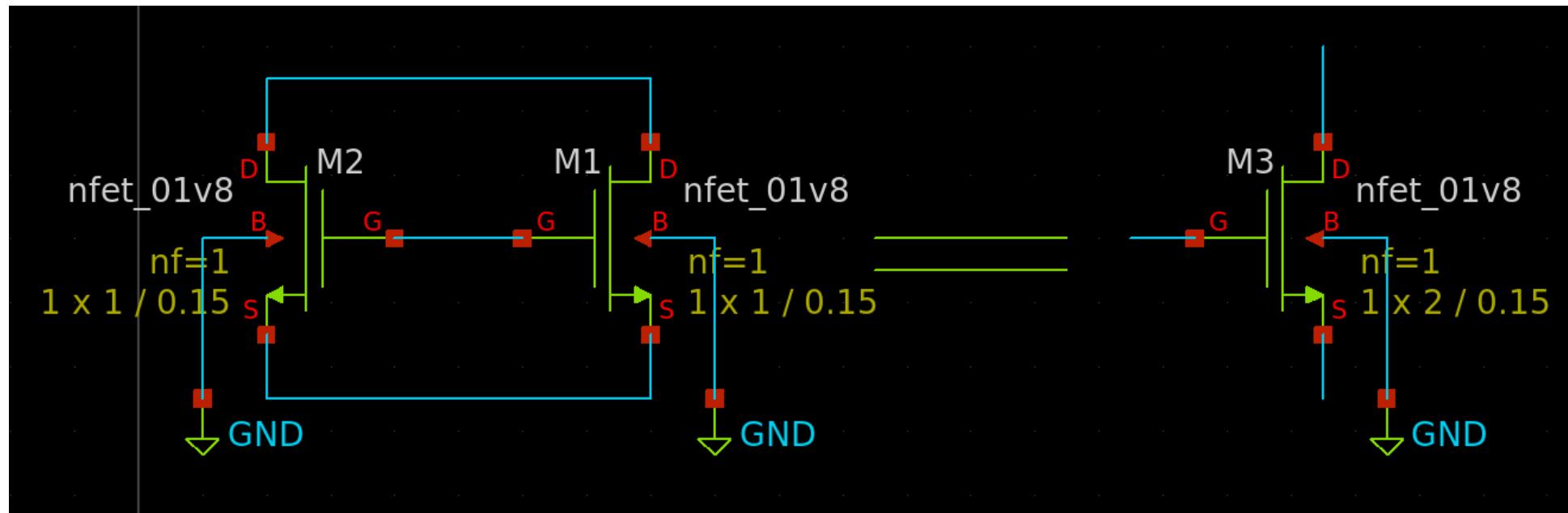
clkbuf16 4→16

bufbuf16 1→3→6→16



3段インバータ

ドレイン・ソース・ゲート・ボディが同じノードである同じサイズのトランジスタは、Wが2倍のトランジスタに等しい

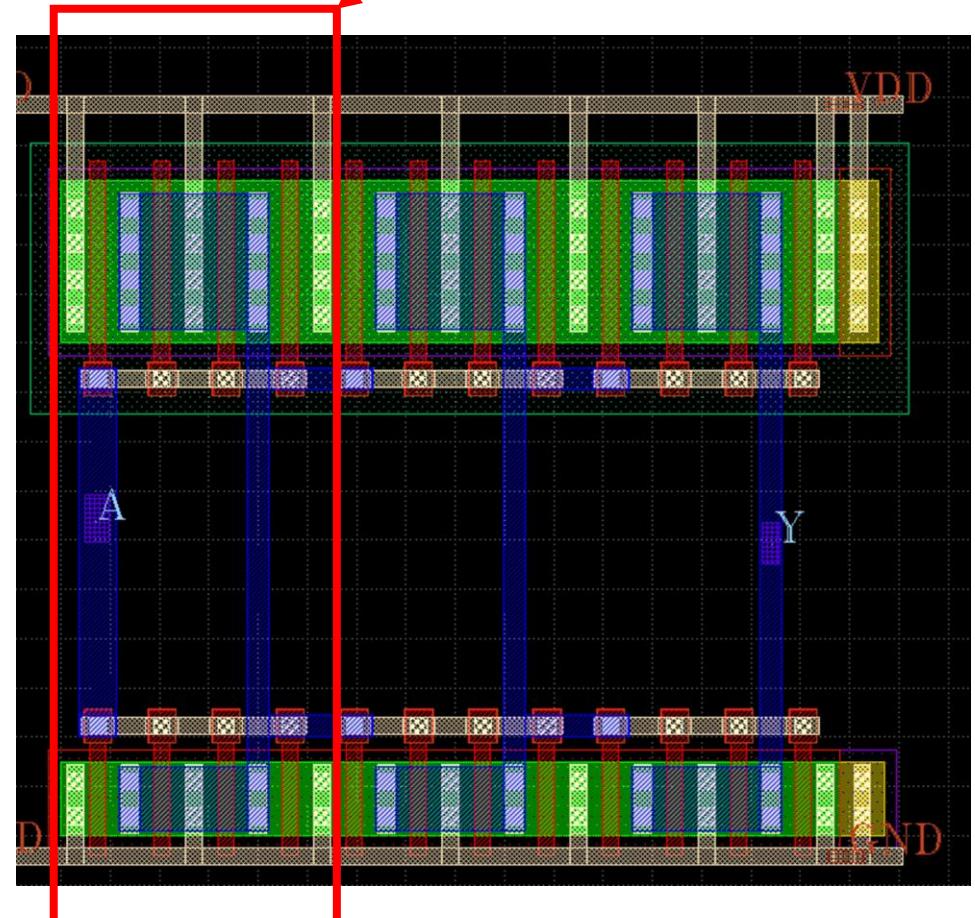
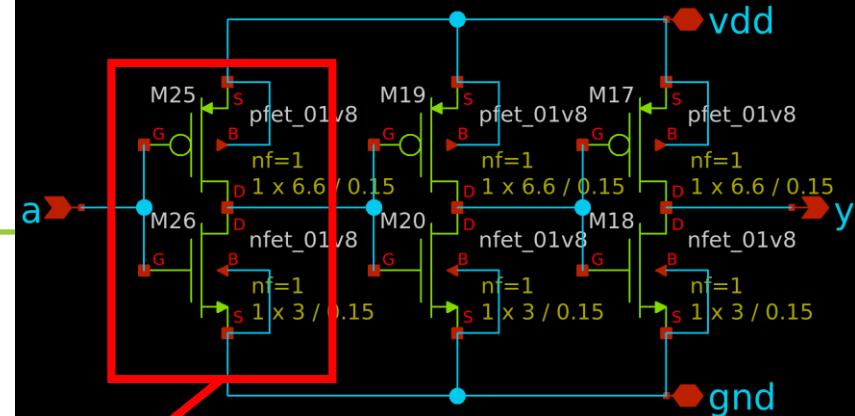


3段インバータ

6.6 / 3 を $(1.65 / 0.75) * 4$ に分割

- 更に3段あるので12個
- Flatten**をして編集

回路図は改変しなくてもLVS通ります

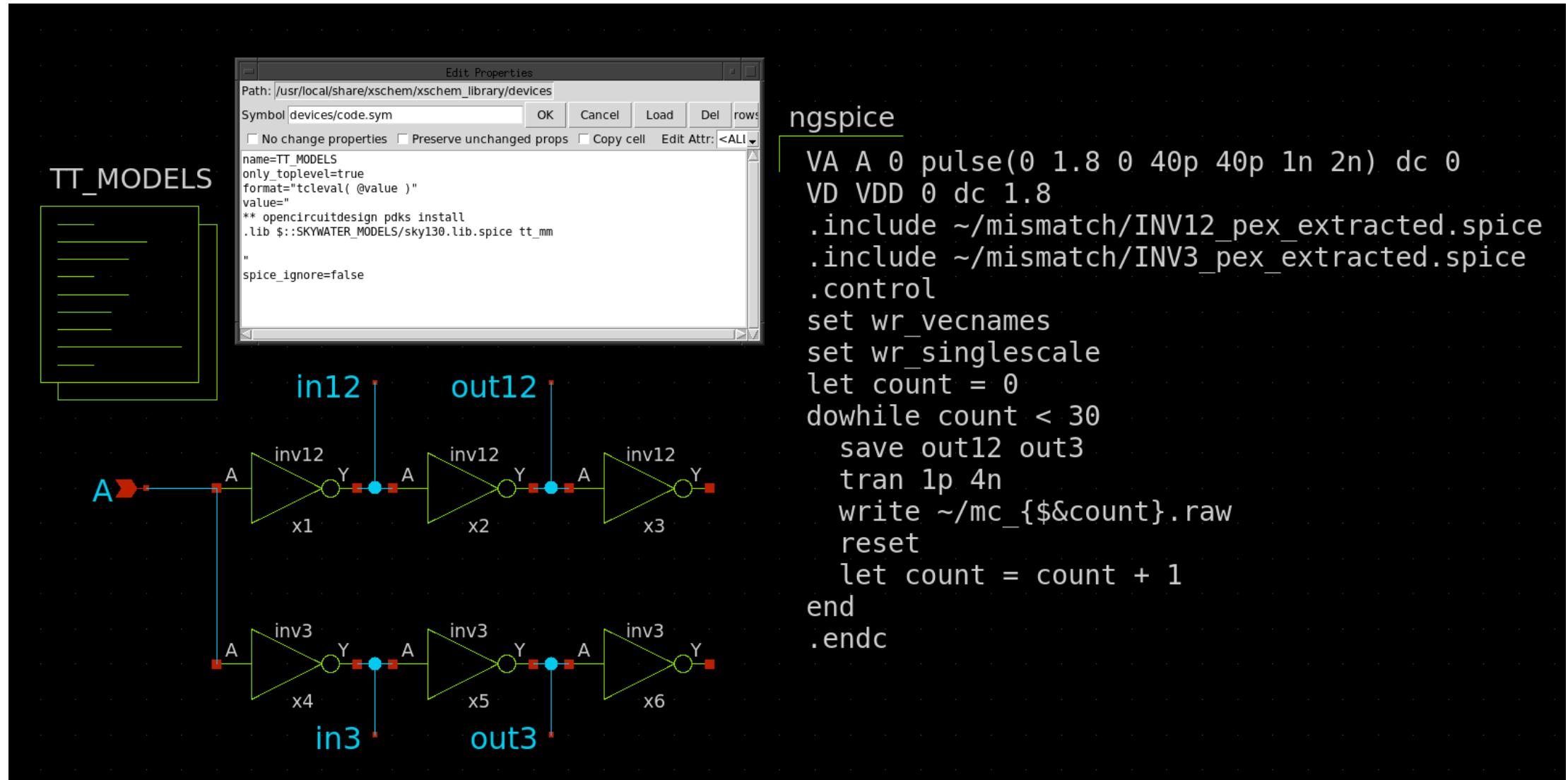


ミスマッチシミュレーション

- マルチフィンガーにはミスマッチ（素子間のばらつき）を緩和させる効果がある
- 素子間のばらつきを見るための ミスマッチシミュレーション がある
 - 各素子にランダムなVthを割り当てる
 - モデルの末尾に_mmをつける tt なら tt_mm

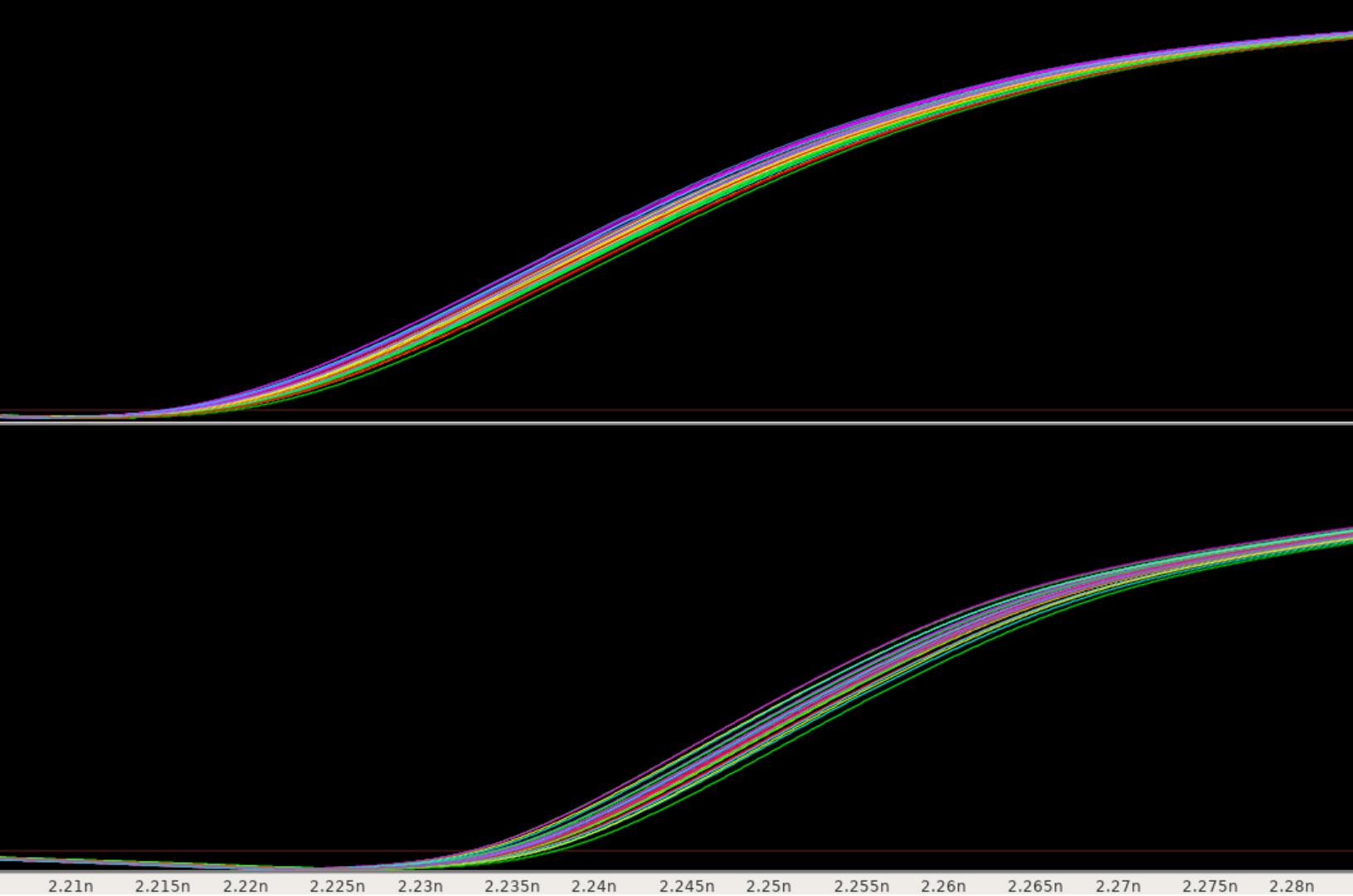
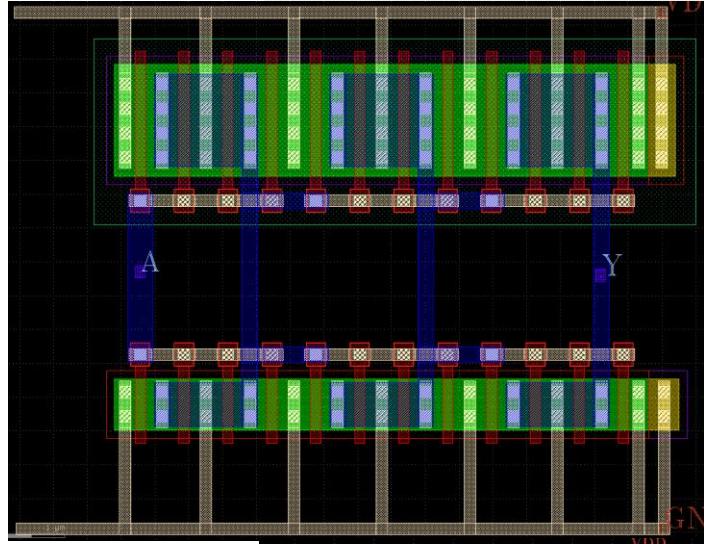
テストベンチ例

mm_sim.sch

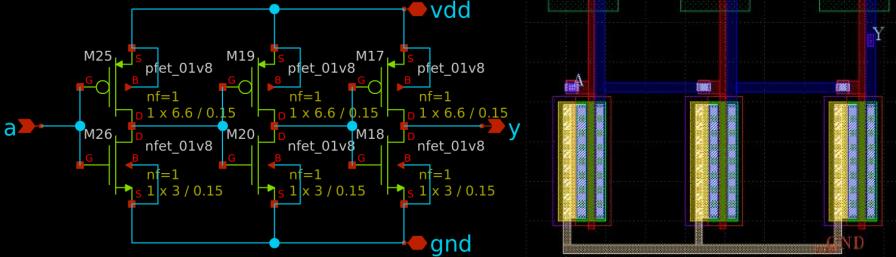


30回、同シードで実行した際のばらつき

inv12.gds, inv12.sym
inv3.gds, inv3.sym



2つともLVS回路図
は同じ



SKY130

Mizuki Mori

2024/12/31

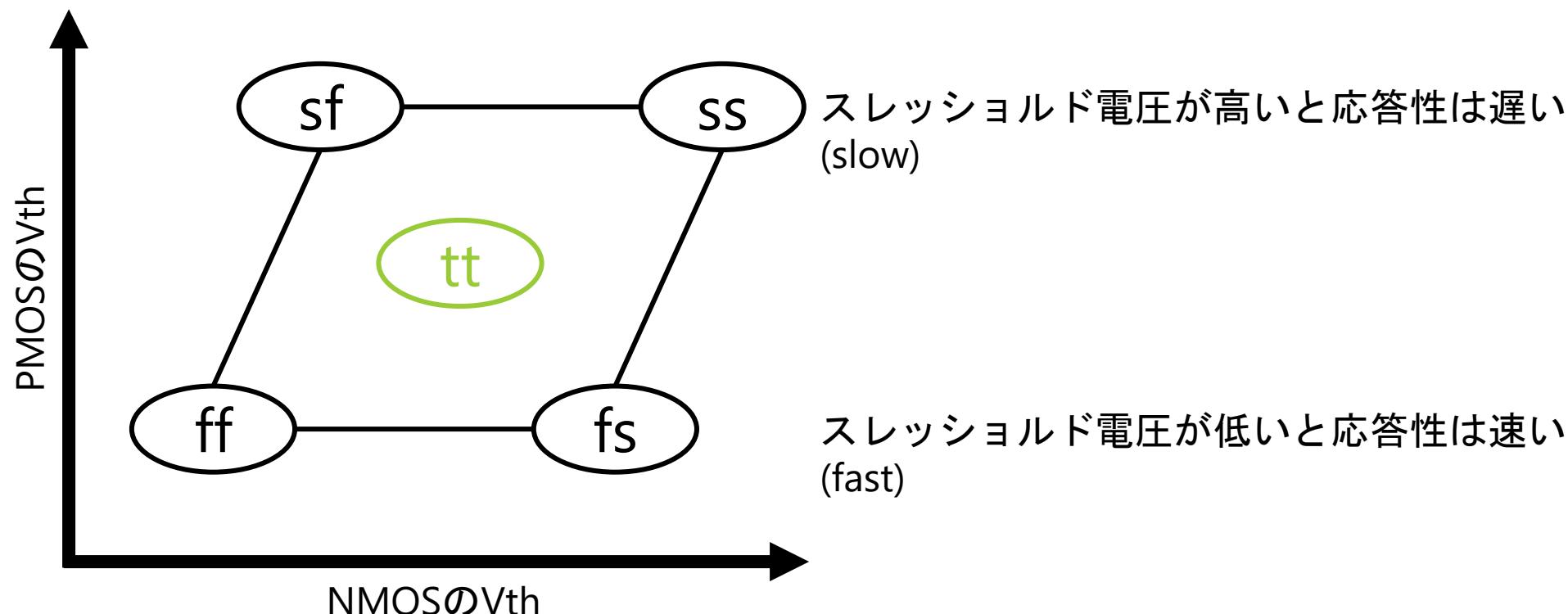
142

モンテカルロシミュレーション

ランダムにプロセスコーナをふってシミュレーション

モンテカルロシミュレーション

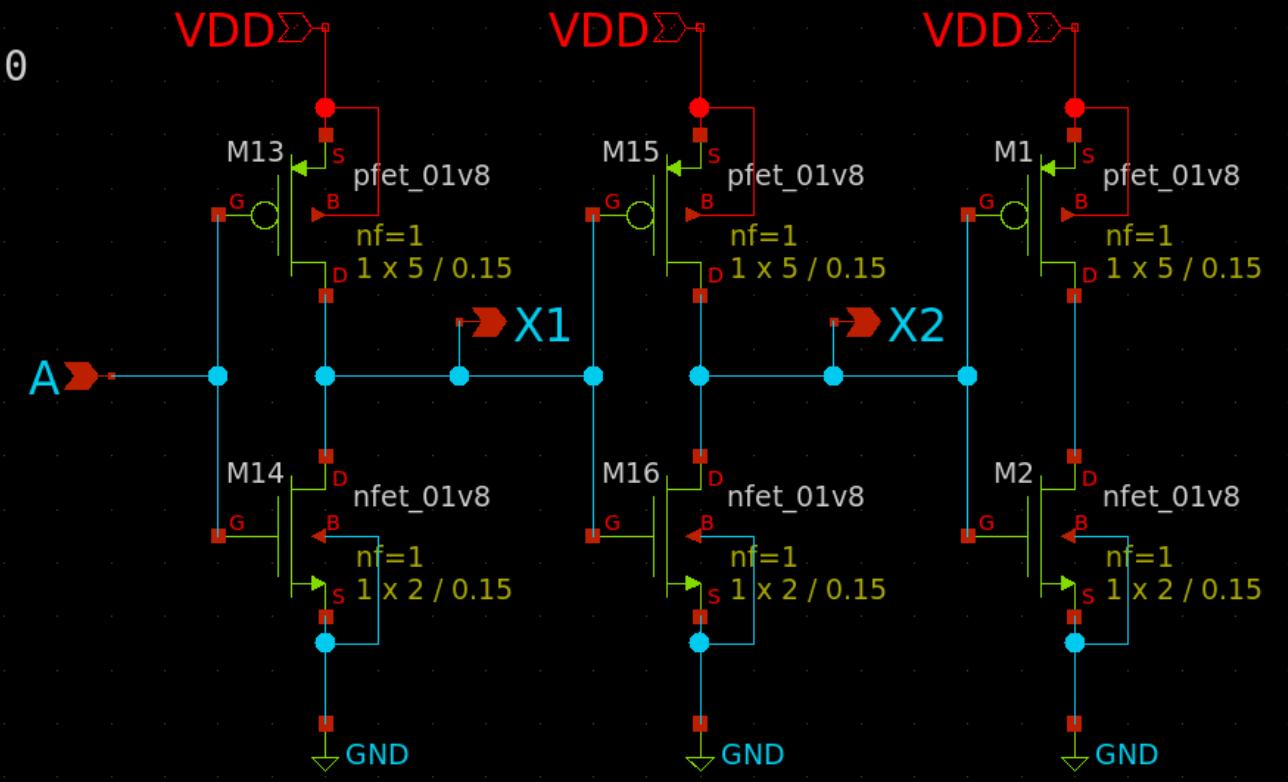
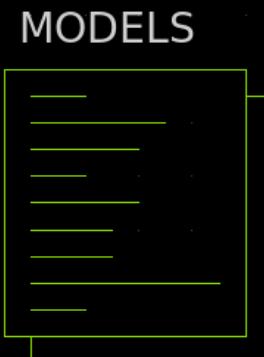
- ・ シミュレーションを実行する際、どのコーナのモデルを使用するか指定する必要あり
- ・ モンテカルロシミュレーションではランダムなコーナでシミュレーションを複数回行う
 - ・ シミュレーションモデルは “mc” を利用する



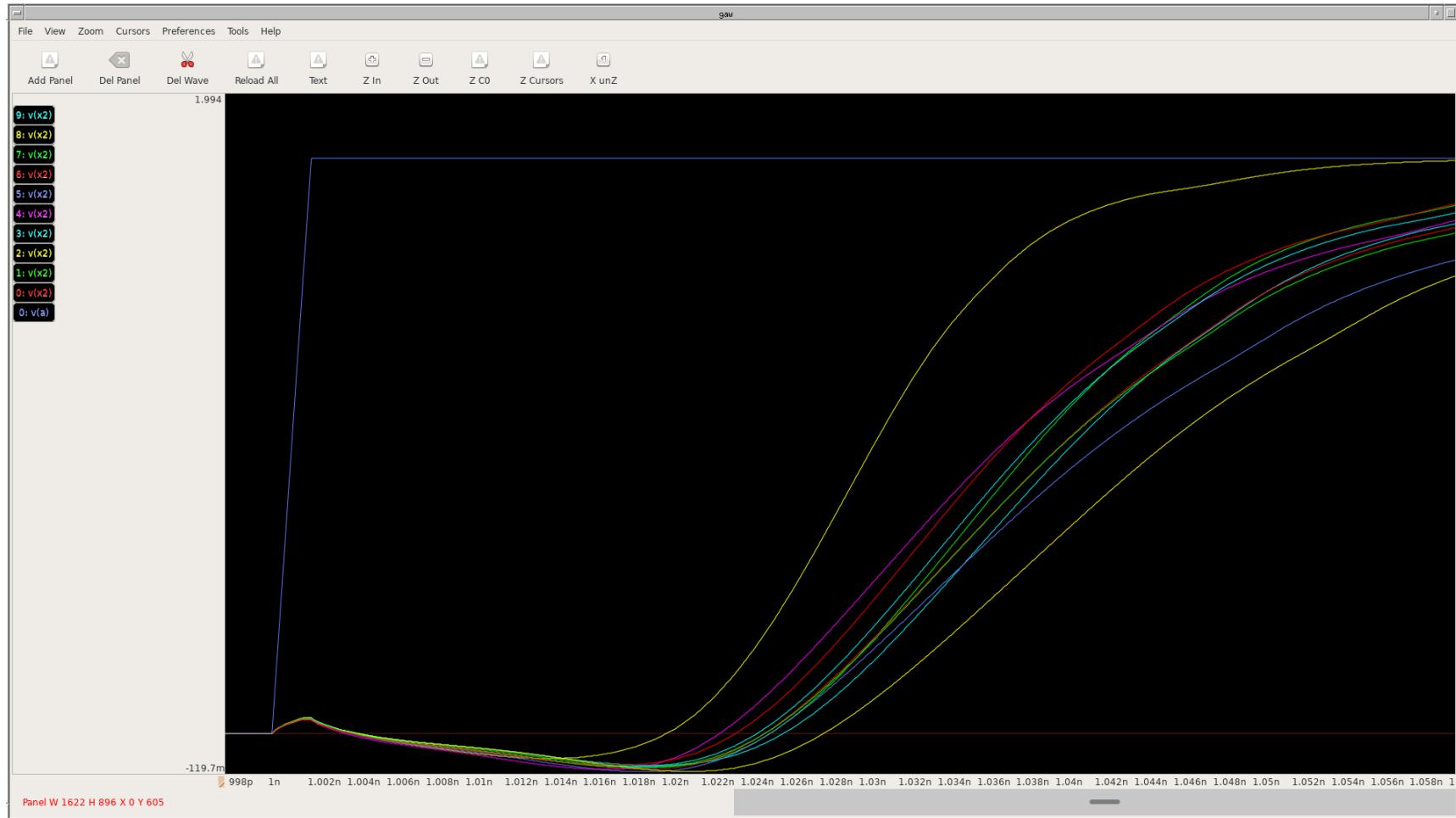
Xschem 構成例 10回 (L=0.15μm)

test_inv_mc.sch

```
ngspice
VA A 0 pulse(0 1.8 0 2p 2p 0.5n 1n) dc 0
VD VDD 0 dc 1.8
.control
set wr_vecnames
set wr_singlescale
let count=0
dowhile count < 10
tran 1p 2n
write ~/mc_{$&count}.raw
reset
let count = count + 1
end
.endc
```



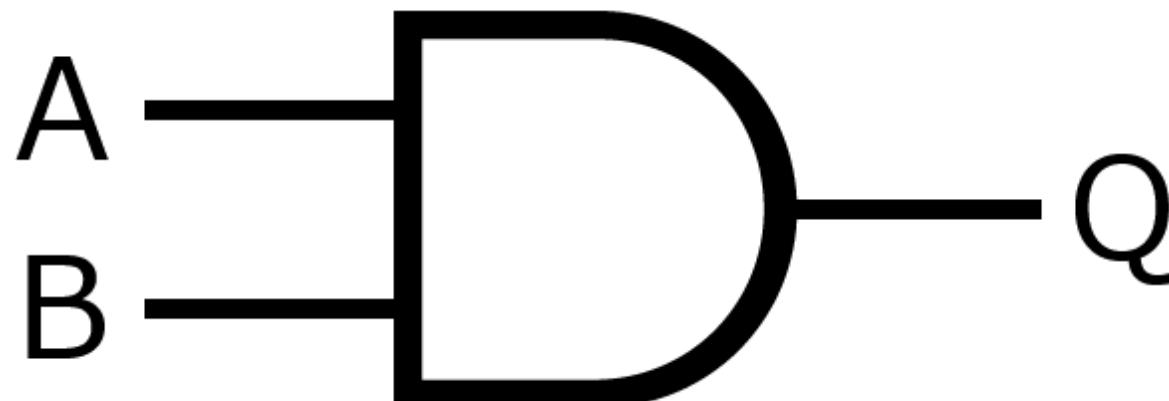
シミュレーション結果 (mc)



スイッチについて

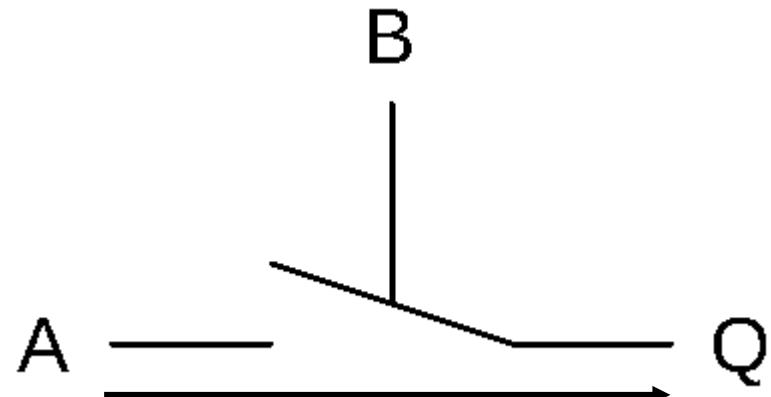
トランスマッショングート

2入力AND

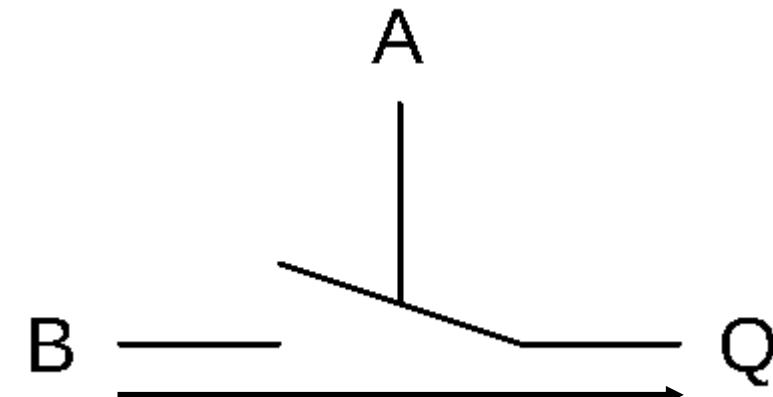


A	B	Q
L	L	L
L	H	L
H	L	L
H	H	H

片方向スイッチとしての2入力AND



B	A	Q
L	L or H	L
H	L	L
H	H	H

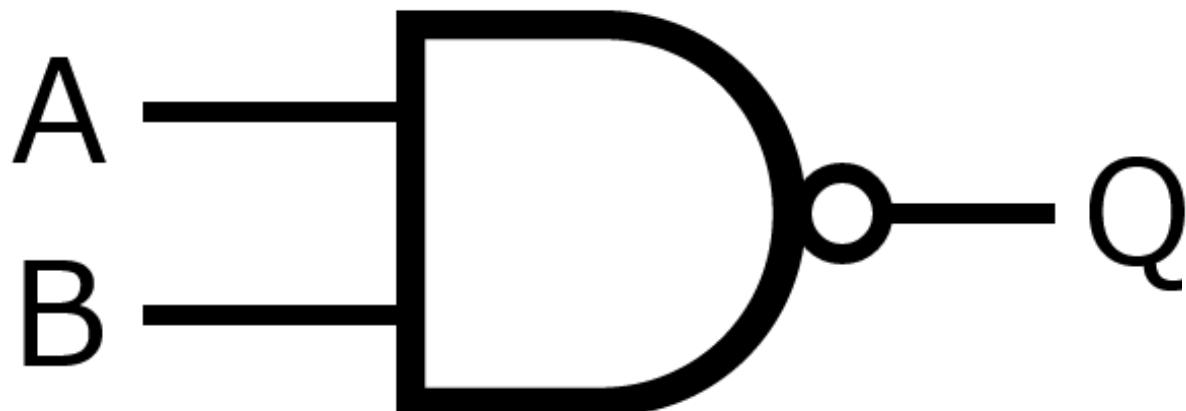


A	B	Q
L	L or H	L
H	L	L
H	H	H

本物のスイッチでは、OFFの時は回路的に遮断された状態になるが、
ANDではGNDに接続されている状態となる

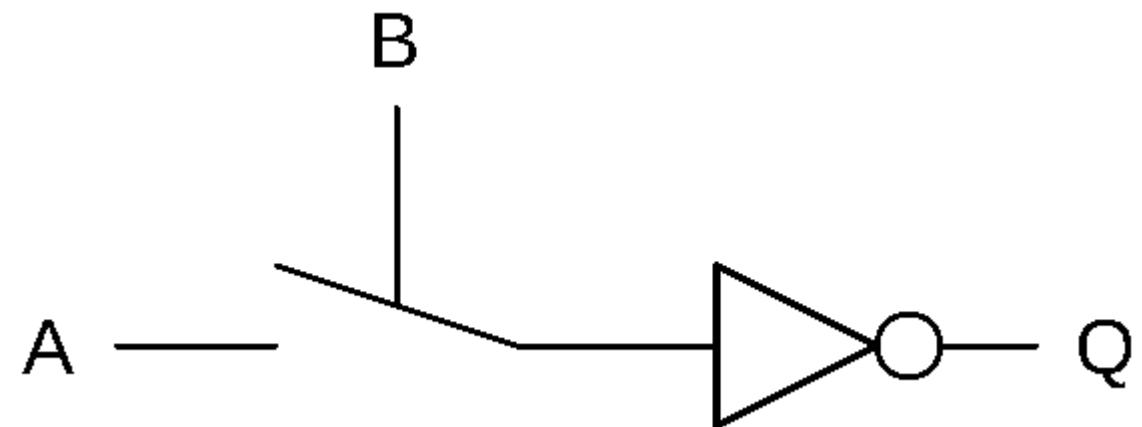
2入力NAND

2入力ANDの出力が反転したもの

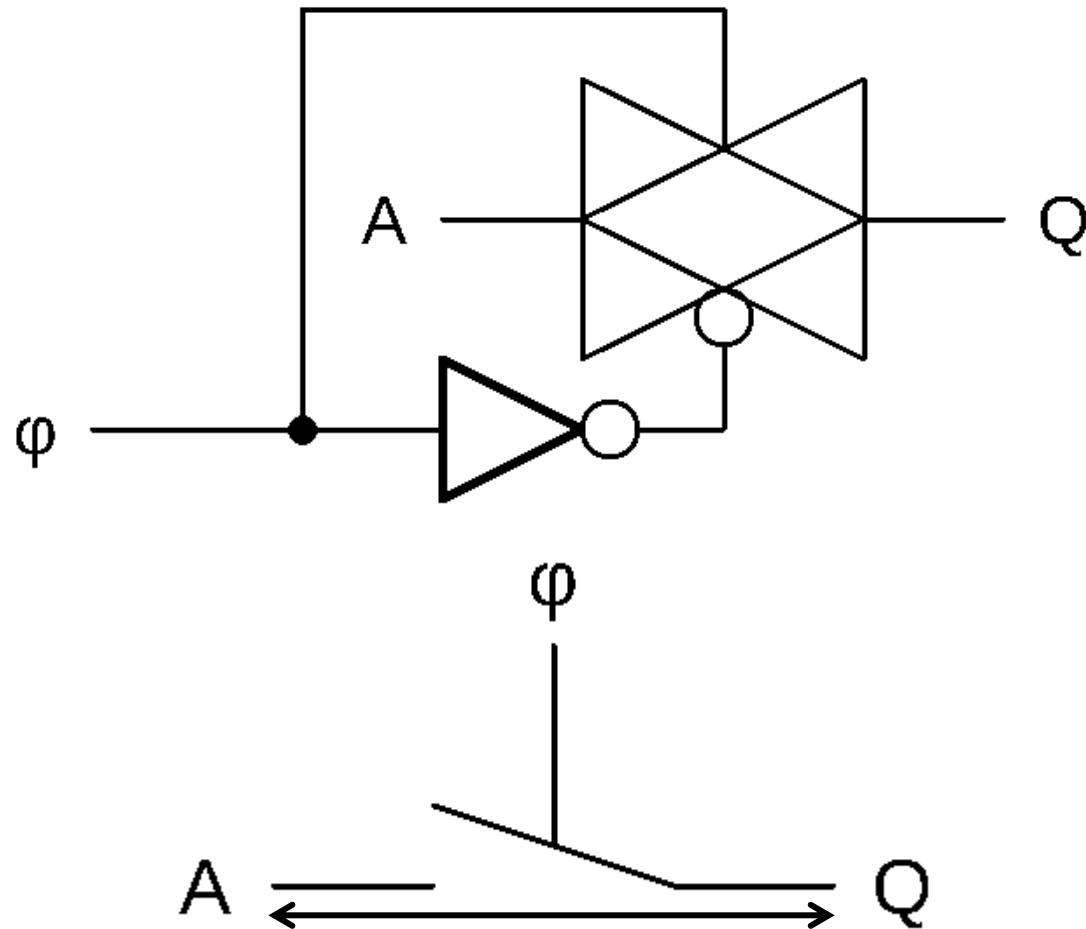


CMOS回路では、NANDを反転したものを
ANDとして扱う

A	B	Q
L	L	H
L	H	H
H	L	H
H	H	L

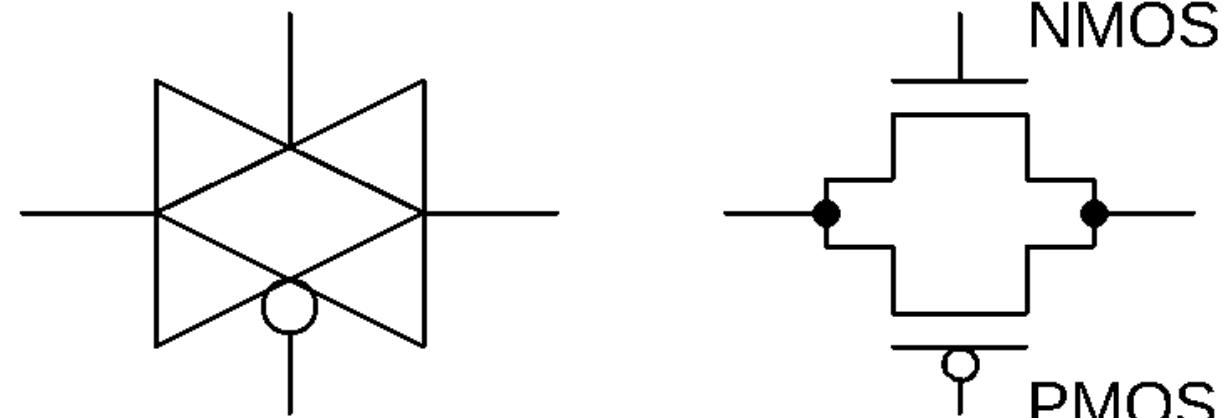


アナログスイッチ (トランスマッショングート)



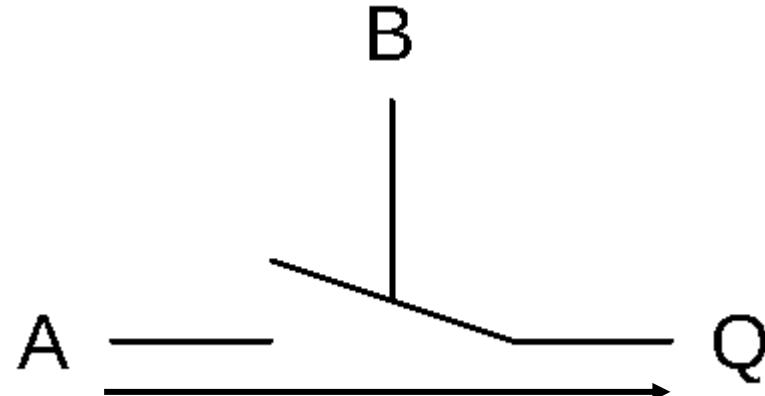
φ	A	Q
L	高抵抗 (Hi-Z)	
H	ほぼ導通 (PASS)	

オン抵抗がある

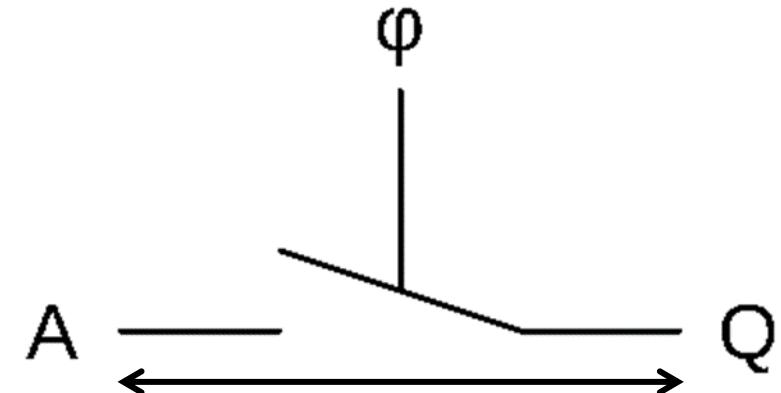


比較

2入力AND (左) と アナログスイッチ (右) の比較



B	A	Q
L	L or H	L
H	L	L
H	H	H

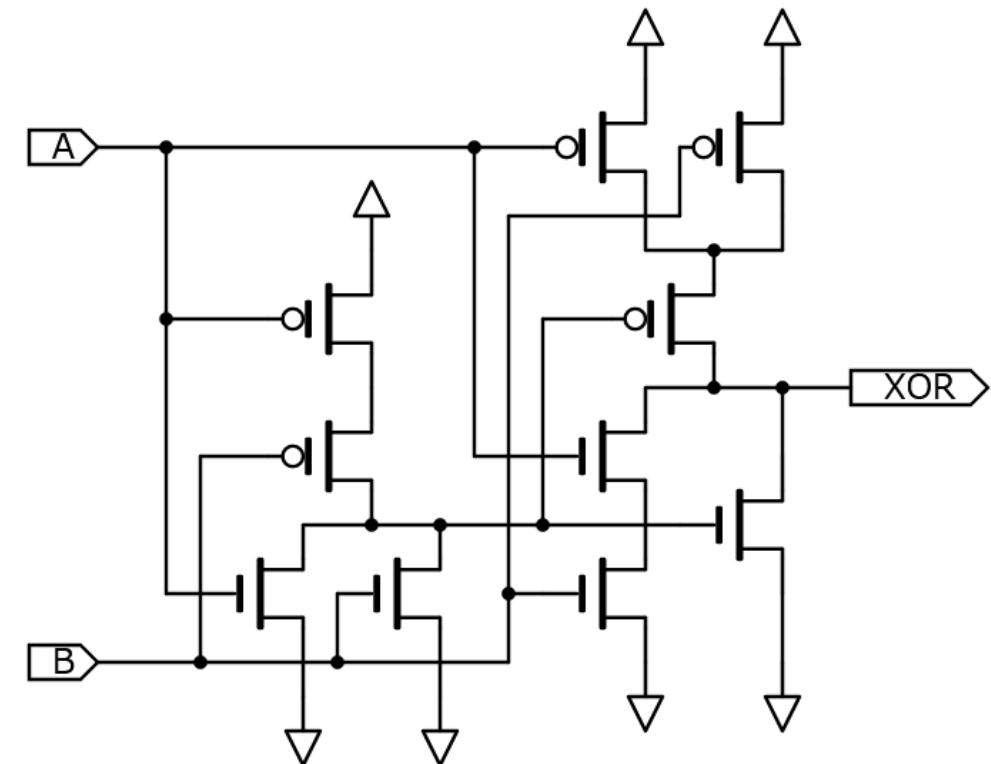
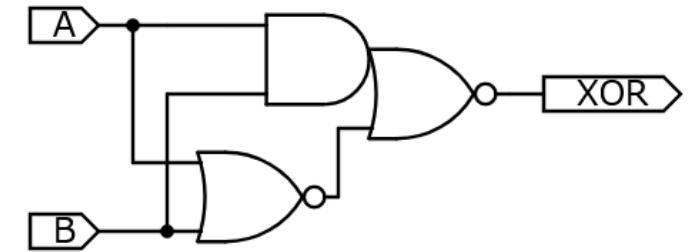


φ	A	Q
L		Hi-Z
H		PASS

アナログスイッチでは、 $\varphi=Low$ の時はほぼ回路的に遮断された状態（高抵抗、Hi-Z）となる
ANDでは、B=Low の時はGNDに接続されている状態(Low)となる

アプリケーション例：2入力XOR

ロジック回路は出力のショートが禁止のため、
ロジック回路だけでXORを組むと少々複雑
(NOR (4) + AOI (6) = 10 transistors)



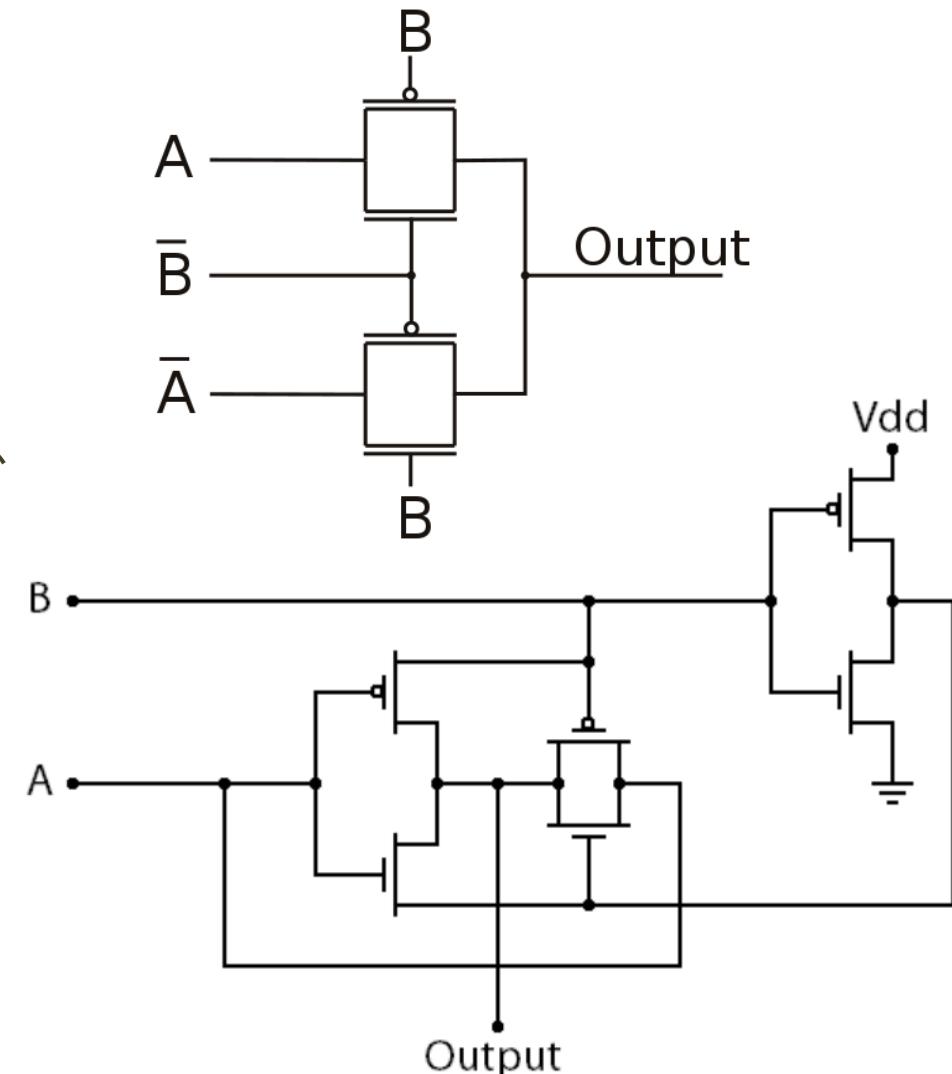
<https://www.pngwing.com/en/free-png-iloox>

アプリケーション例：2入力XOR

アナログスイッチは $\phi=Low$ でHi-Zとなるため、取り扱いに注意すれば**出力をショートできる**

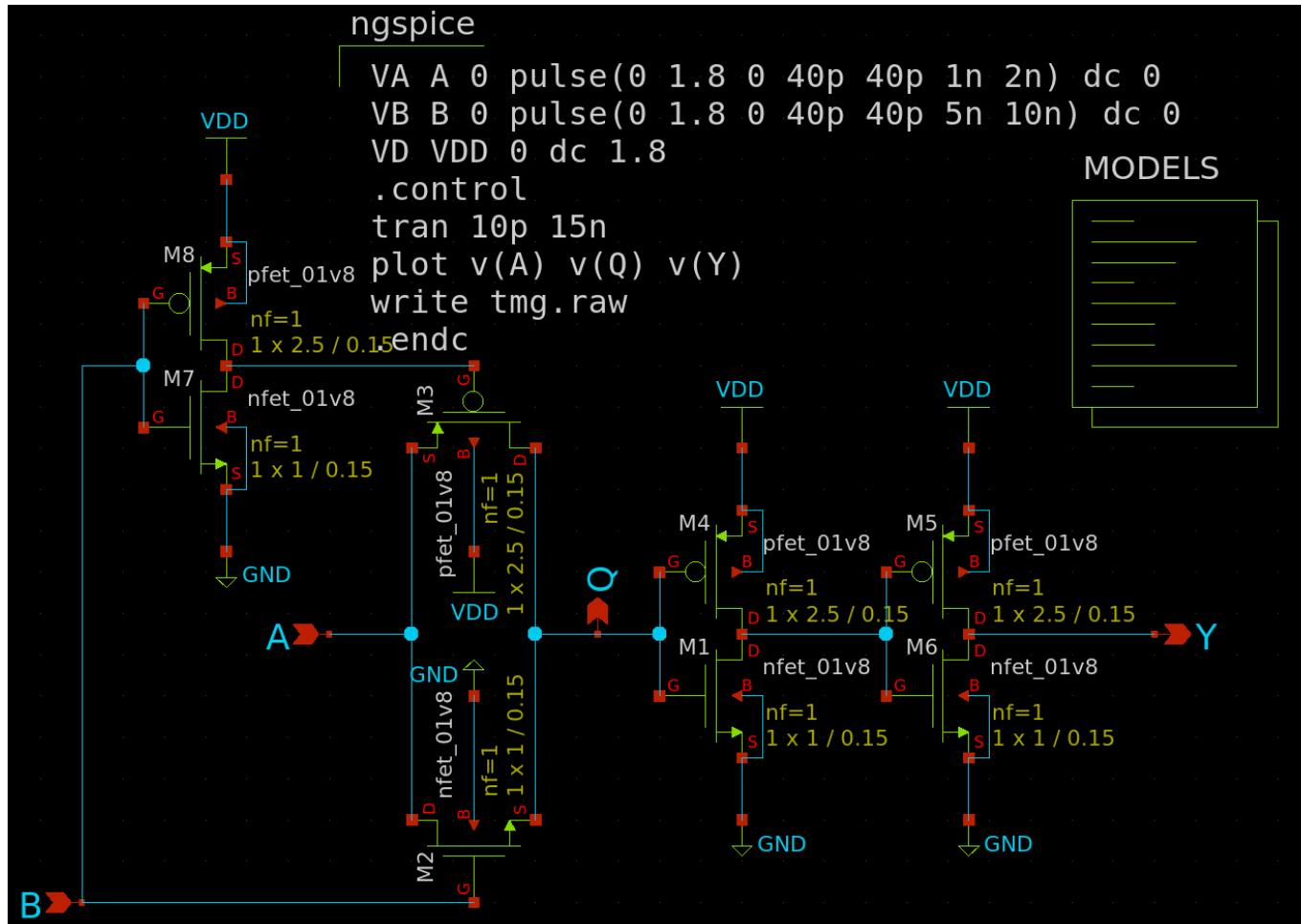
双方向という特徴を用いるとトランジスタの数が削減されることがある

意図せずゲート入力をHi-Z (floating)とするのは好ましくないため、組み合わせる際に気を使う項目が増える。
出力が必ずHighかLowとなるように設計する。

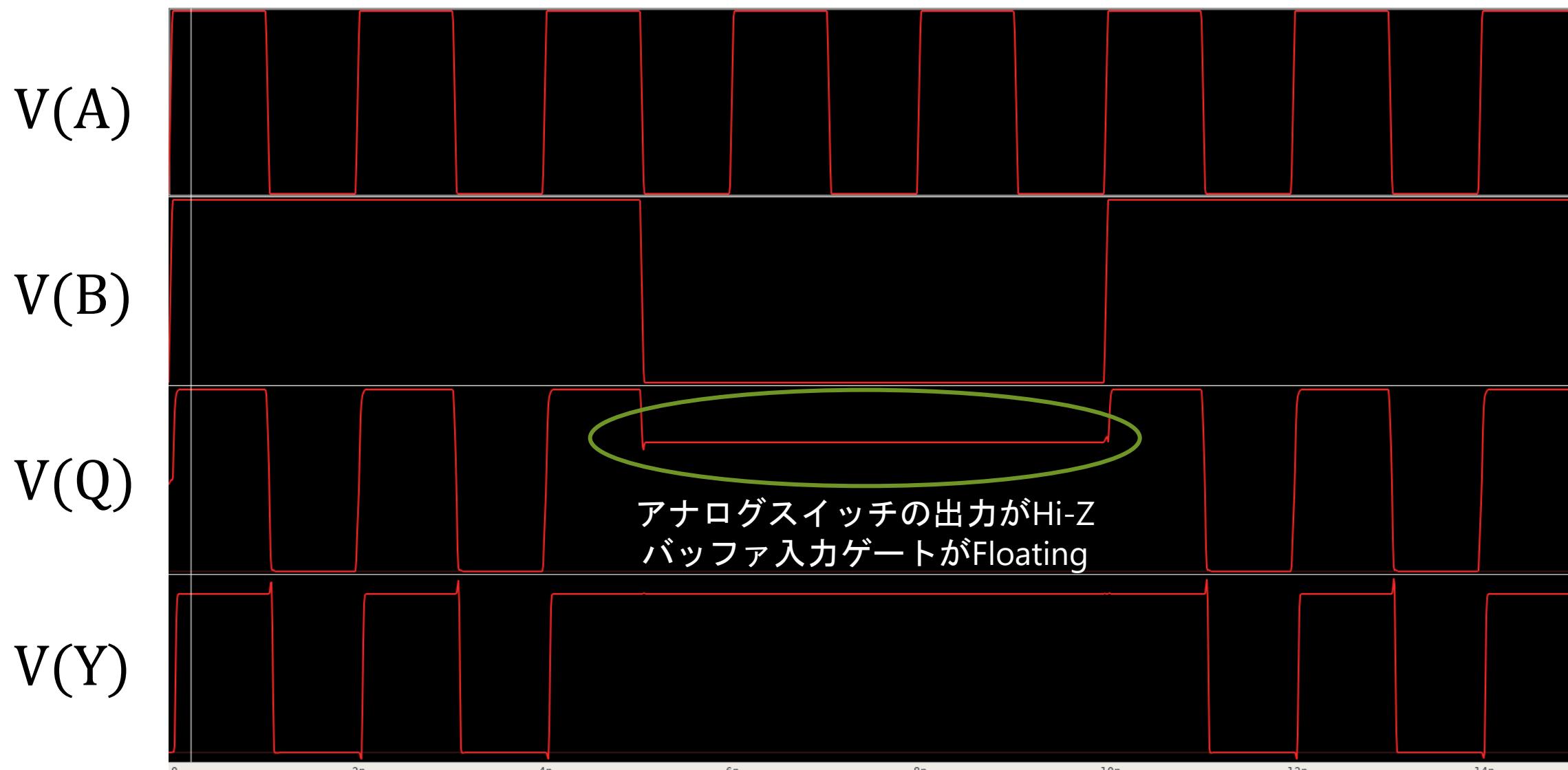


https://en.wikipedia.org/wiki/XOR_gate

アナログスイッチの簡単なシミュレーション



アナログスイッチの簡単なシミュレーション

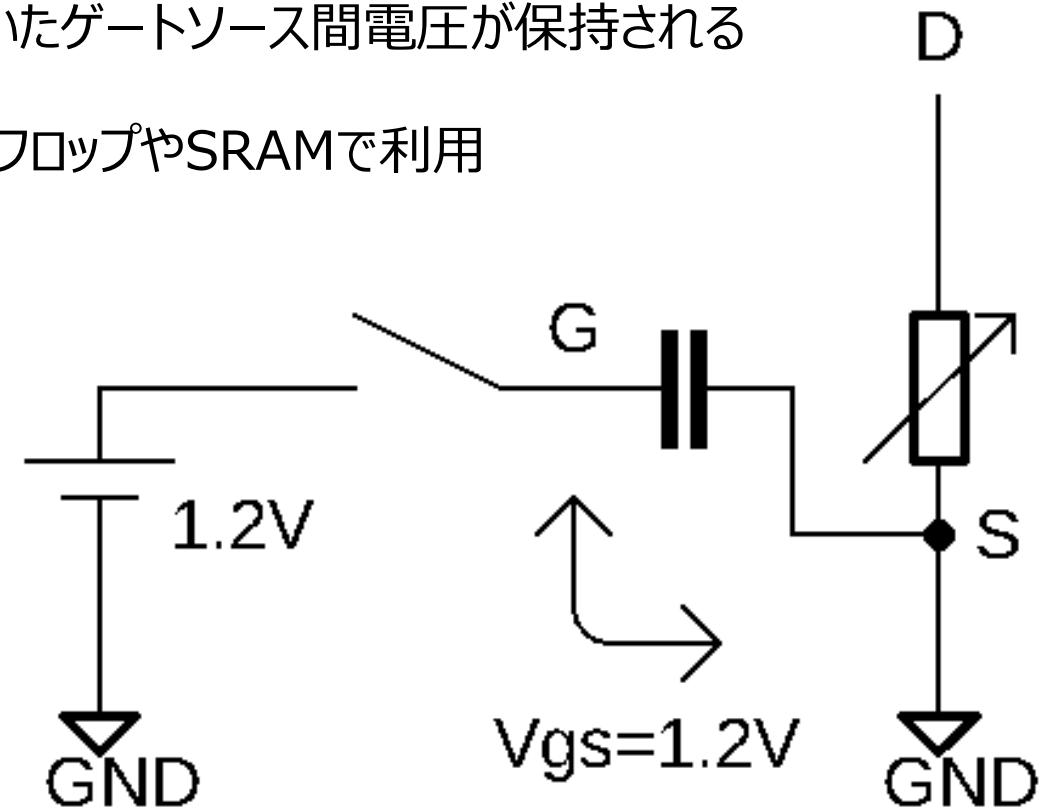


DラッチとDフリップフロップ

フローティングゲートを用いたロジックゲート

NMOSの特性：フローティングゲート

- ドレンソース間電流 I_{DS} はゲートソース間の電圧 V_{GS} によって決定するが、**ゲートには寄生容量がある**
- ゲートがフローティング状態になった時は直前に入力されていたゲートソース間電圧が保持される
 - フローティングゲートと呼ばれる状態でDラッチ/DフリップフロップやSRAMで利用

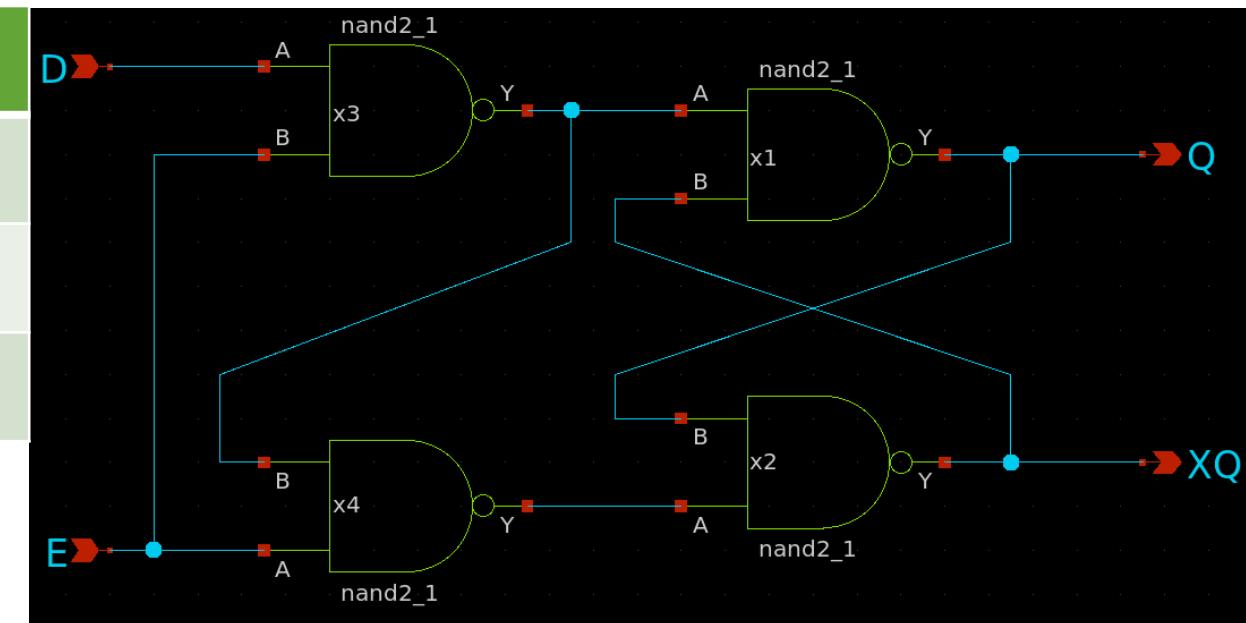


Gated D flip-flop / D latch

EがH→Lとなった時のDの値を保持する

- E=Hの時はDの値がそのままQへ反映される（通過、PASS）
- EはCLKと表記されることもある

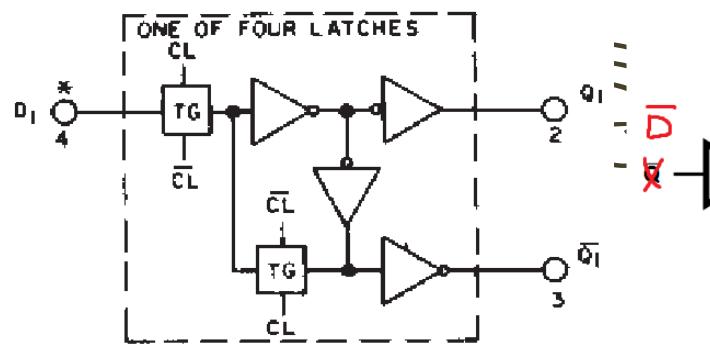
E (CLK)	D	Q
L	X	保持
H	L	L
H	H	H



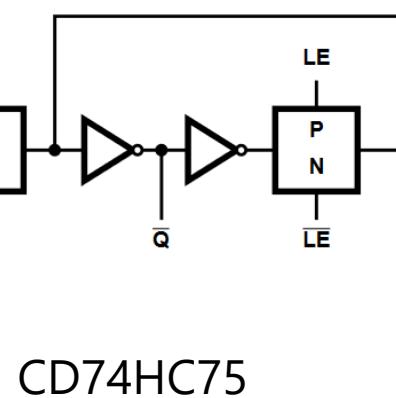
Gated D flip-flop / D latch

トランジションゲートを使用した
Gated D flip-flop / D latch

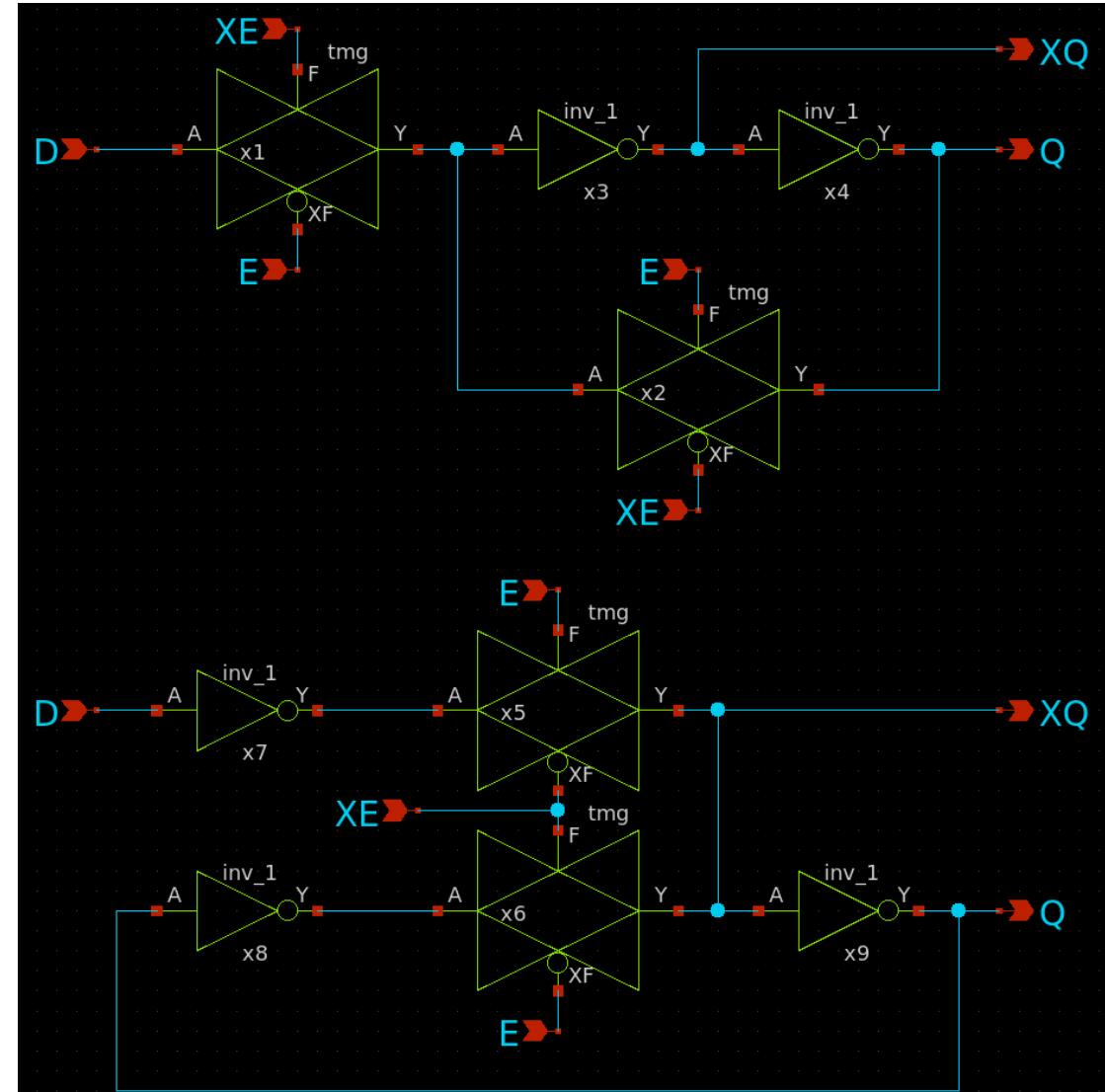
Eを反転させるためのインバータが必要



CD4042B

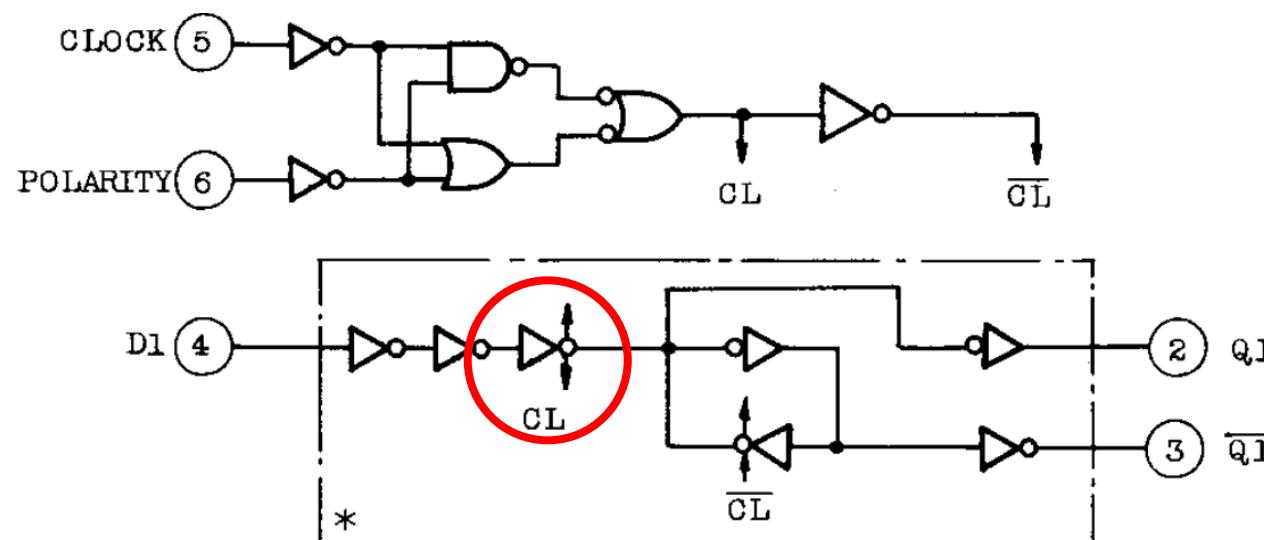


CD74HC75

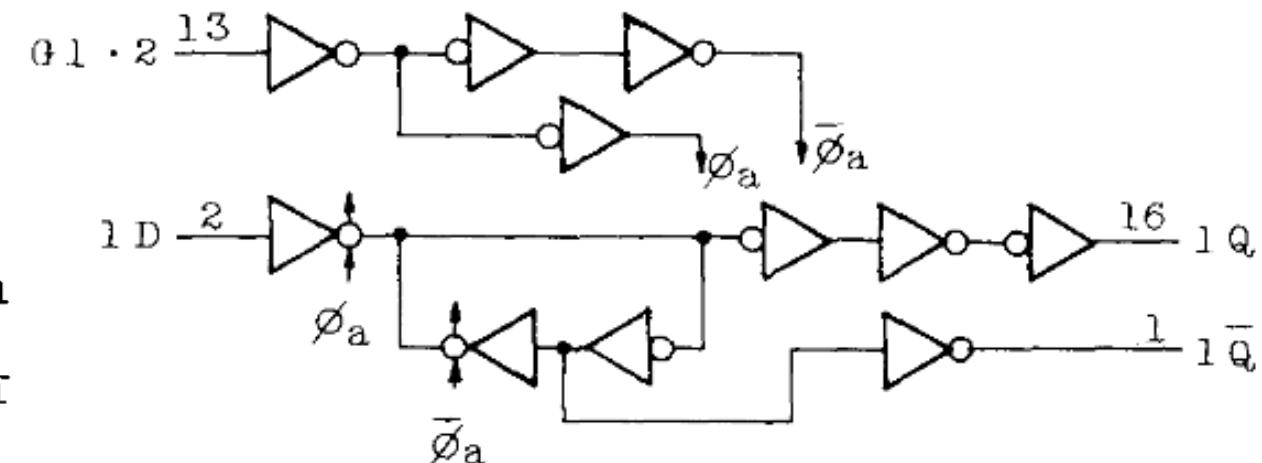


Gated D flip-flop / D latch

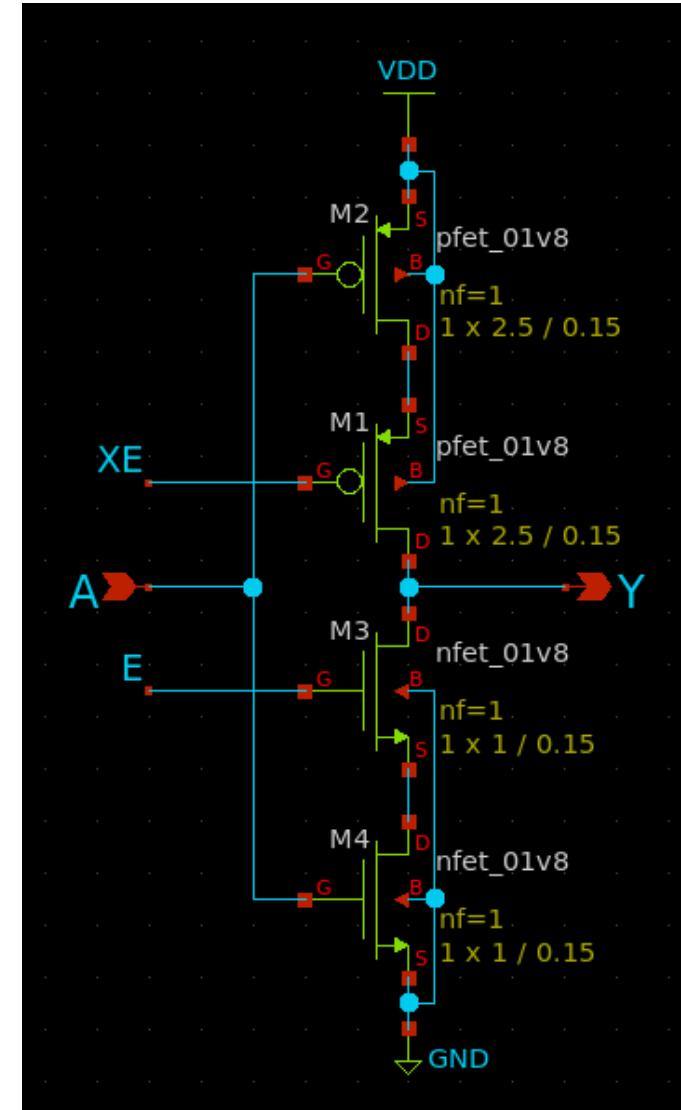
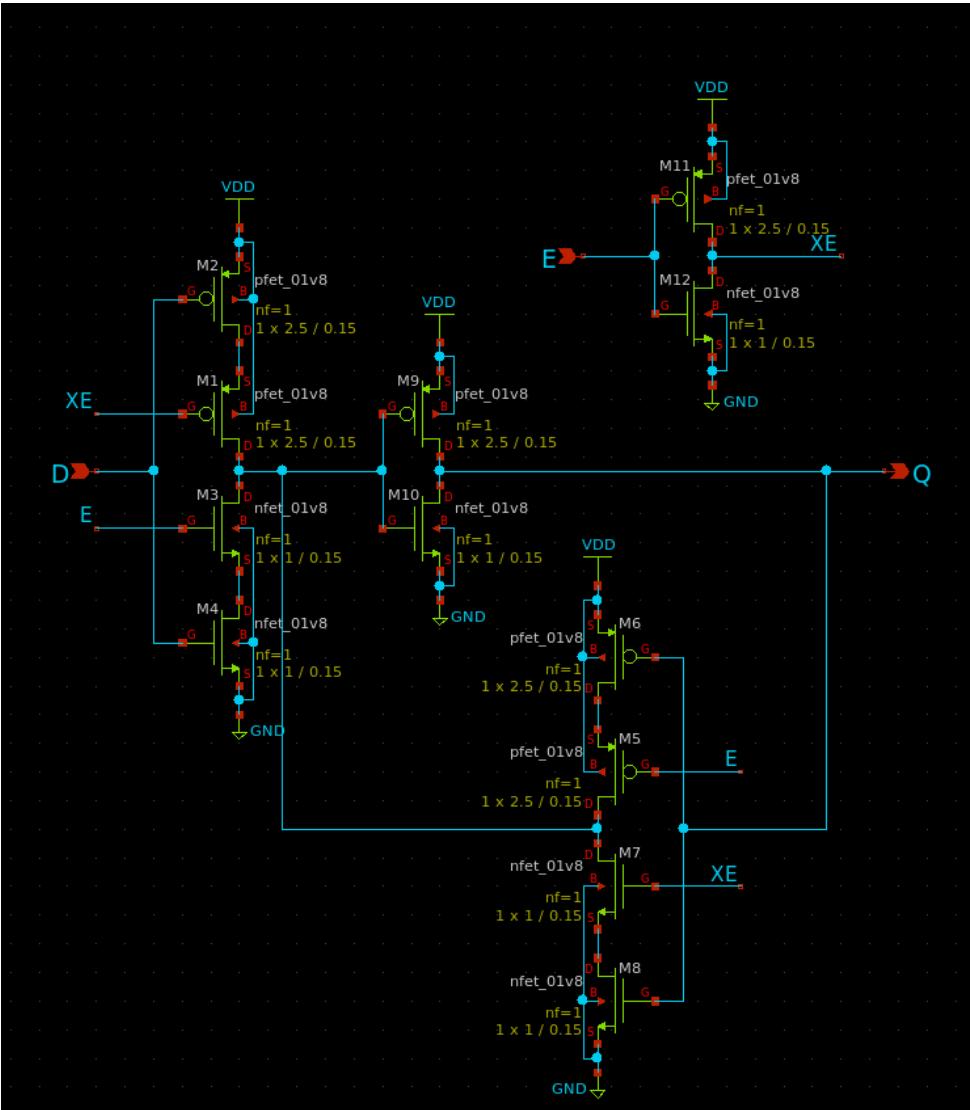
TC4042B(左)とTC74HC75(右)でのDラッチ構成例



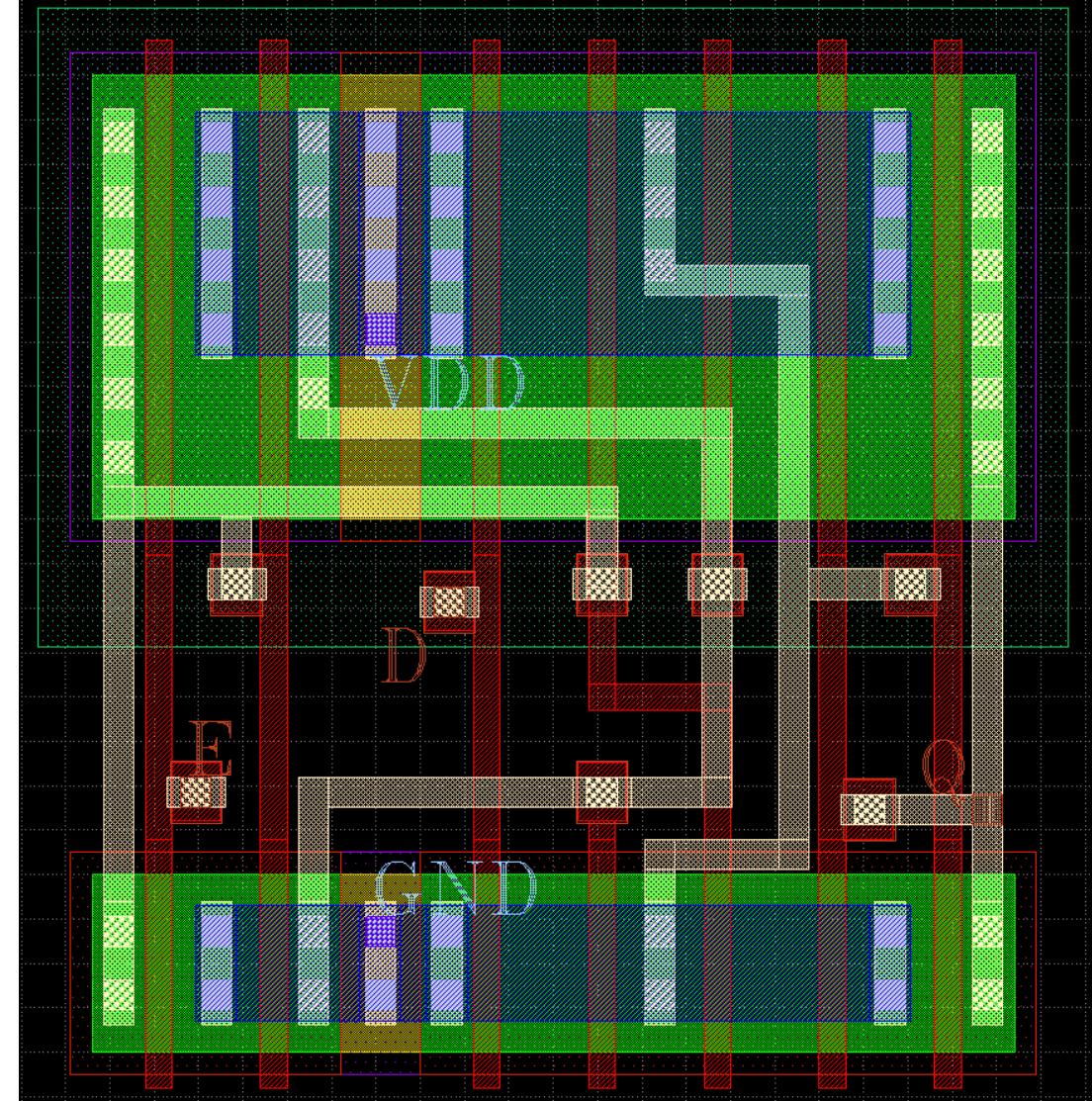
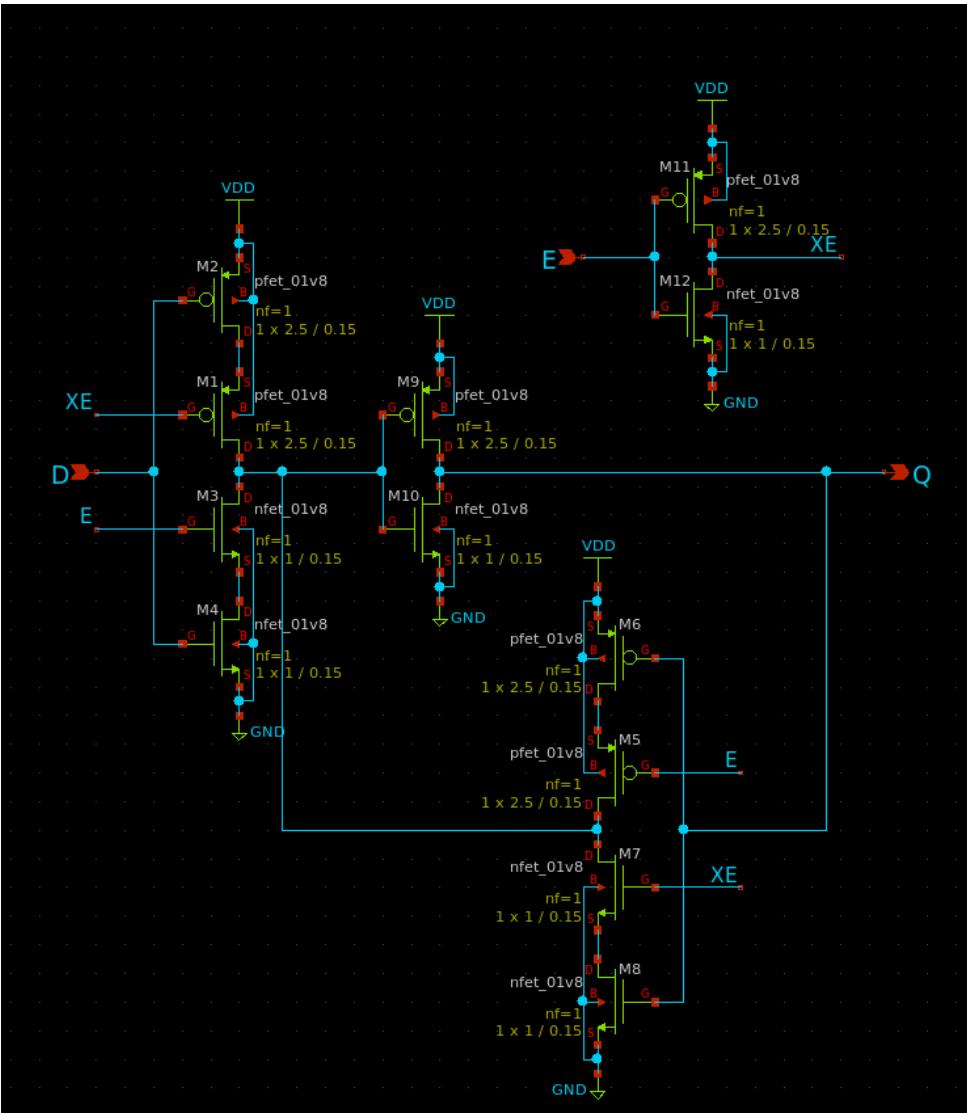
クロックドインバータ



クロックドインバータを用いたDラッチの例



クロックドインバータを用いたDラッチの例



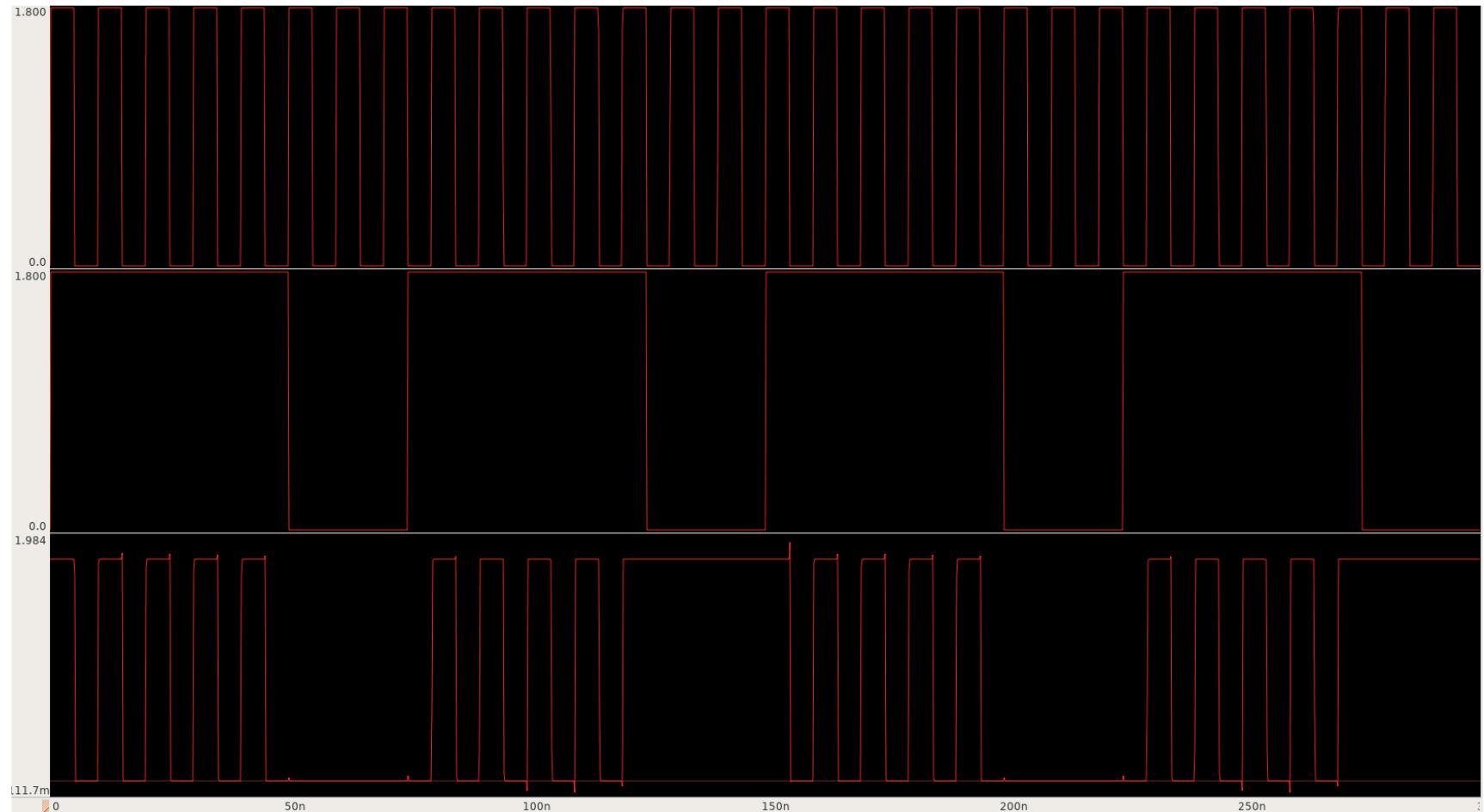
動作確認の波形例

dLatch_bench2.sch

D

E

Q

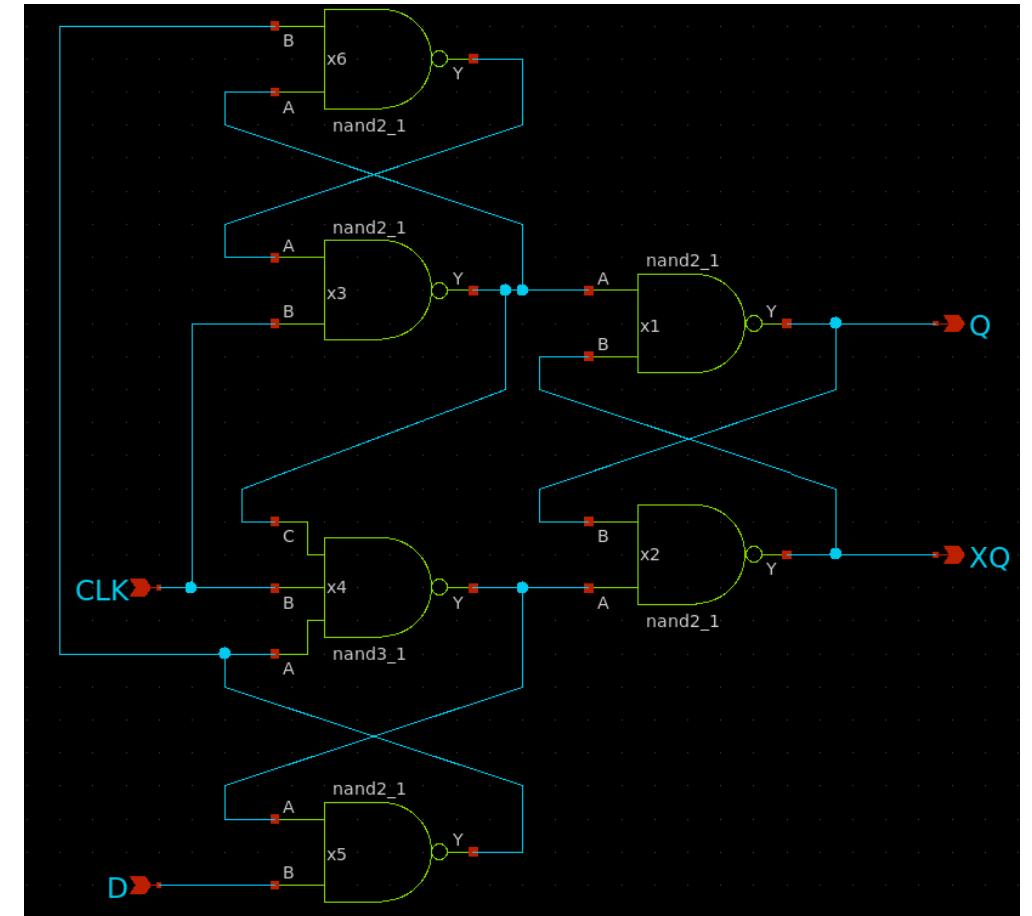


Positive-edge-triggered D flip-flop / D flip-flop

DラッチではCLK(E)がHの時はパスだった

DフリップフロップではCLKがL→Hとなった時だけ出力Qが変化

CLK	D	Q
L→H	L	L
L→H	H	H
上記以外	X	保持



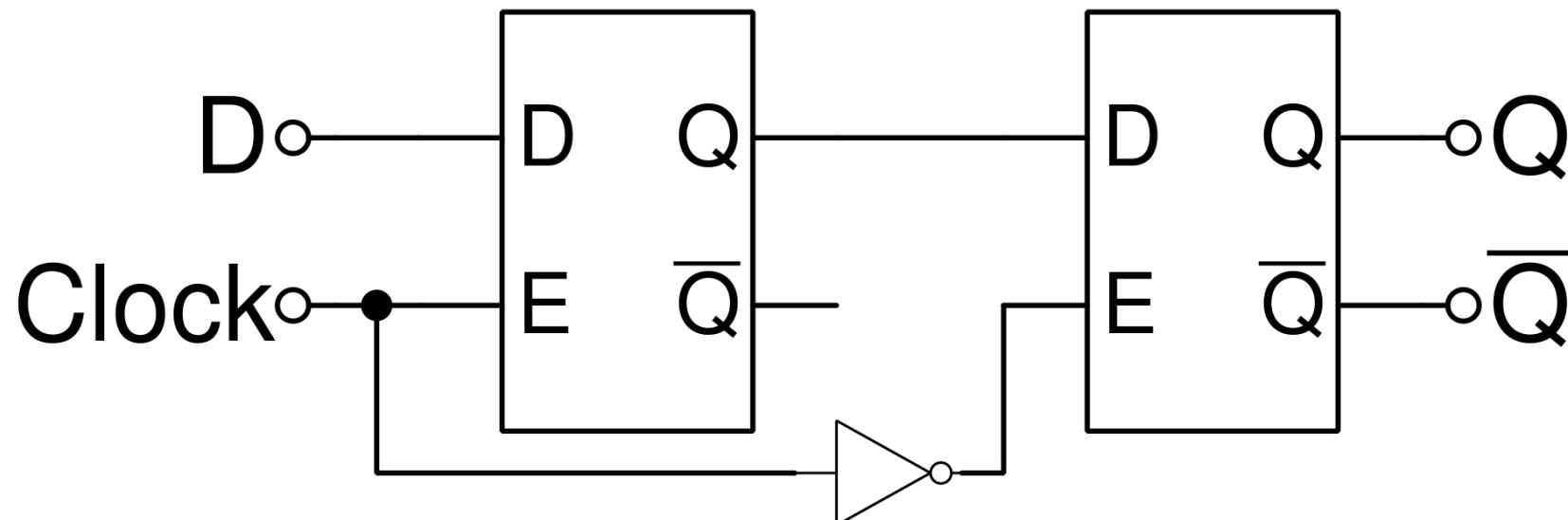
Positive-edge-triggered D flip-flop / D flip-flop

Dラッチ 2 個でも構成可能

Master-slave edge-triggered D flip-flop と呼ばれる

下図はNegative-edge-triggeredなので、Positive-edge-triggeredとするには
入力Clockを反転させる必要アリ

ロジックICではDラッチ 2 個を用いてDフリップフロップを構成していることが多い



Positive-edge-triggered D flip-flop / D flip-flop

CD4013B(左)とCD74HC74(右)でのDフリップフロップ構成例

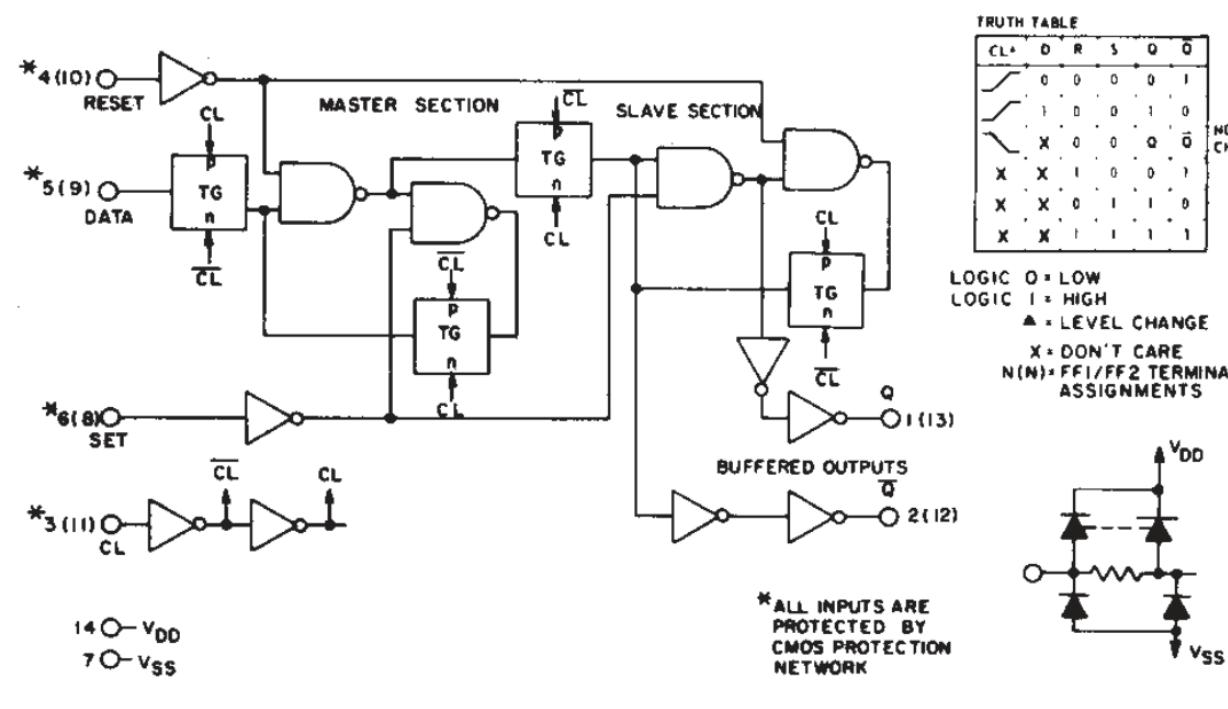
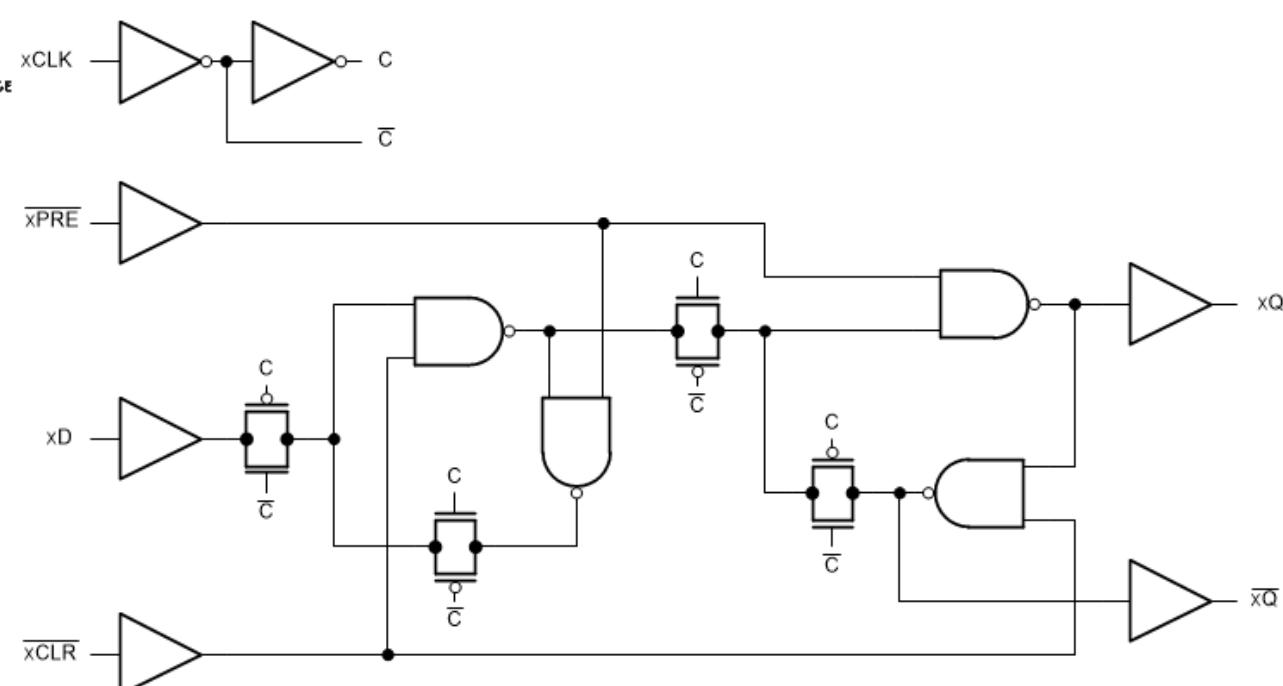
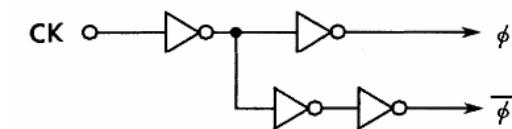
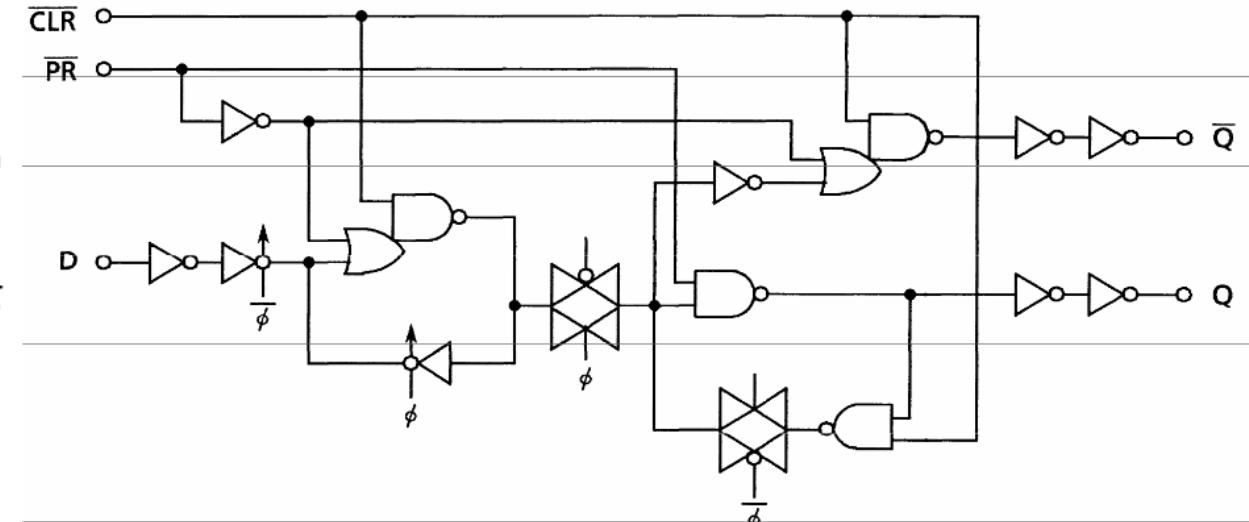
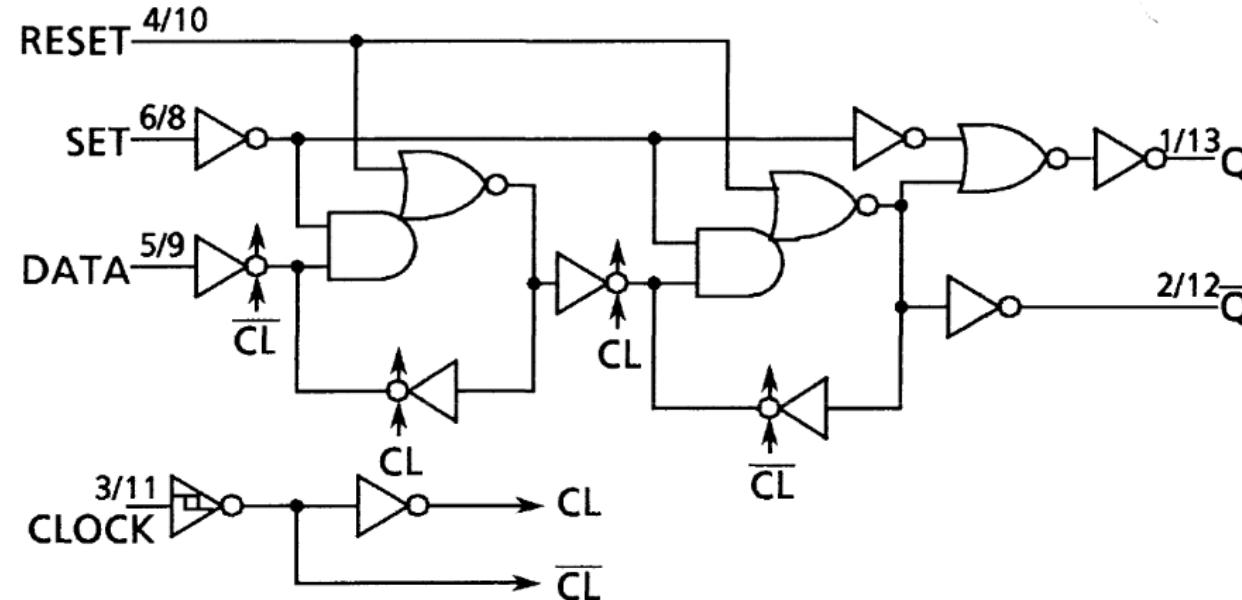


Fig. 7 Logic diagram and truth table for CD4013B (one of two identical flip-flops).

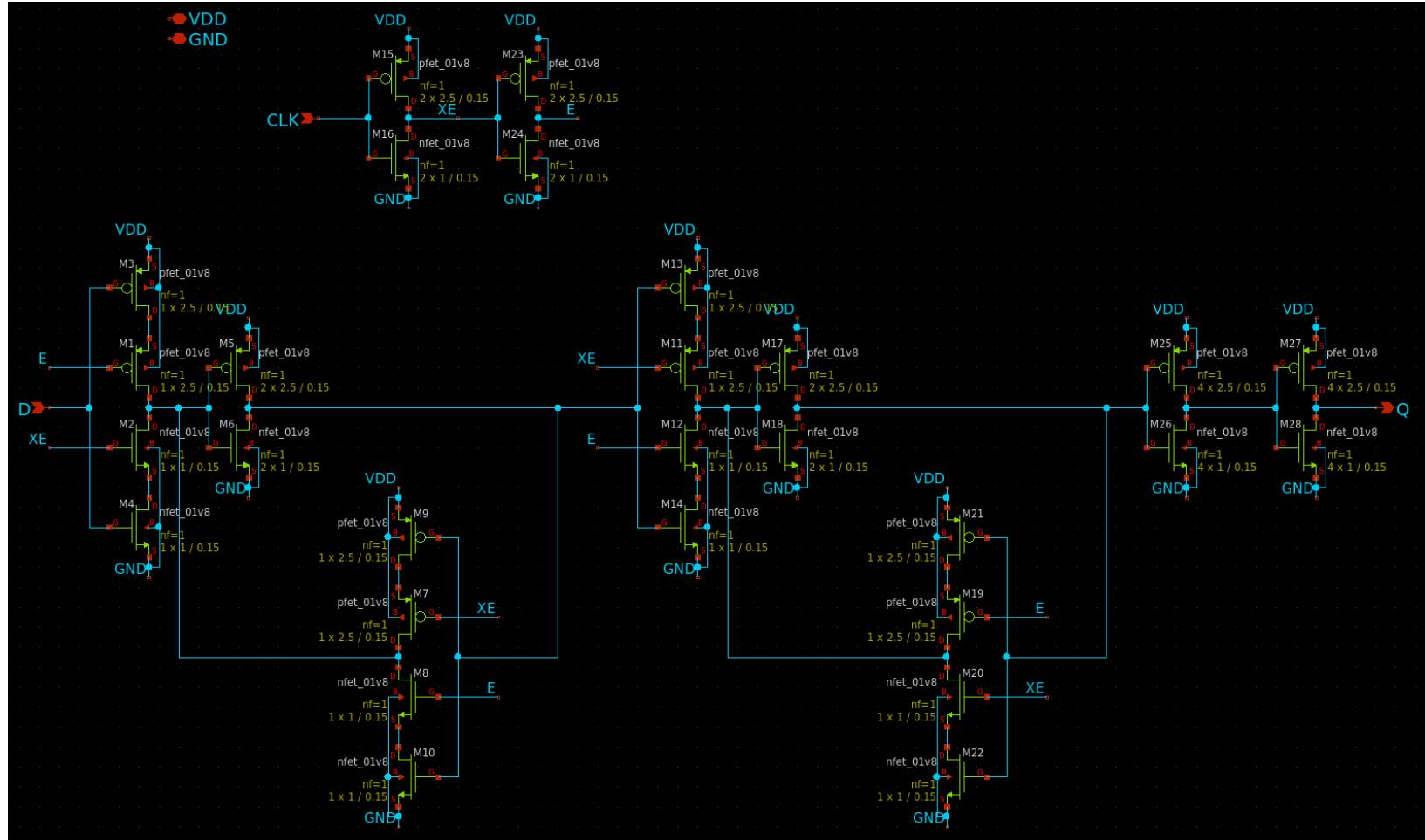


Positive-edge-triggered D flip-flop / D flip-flop

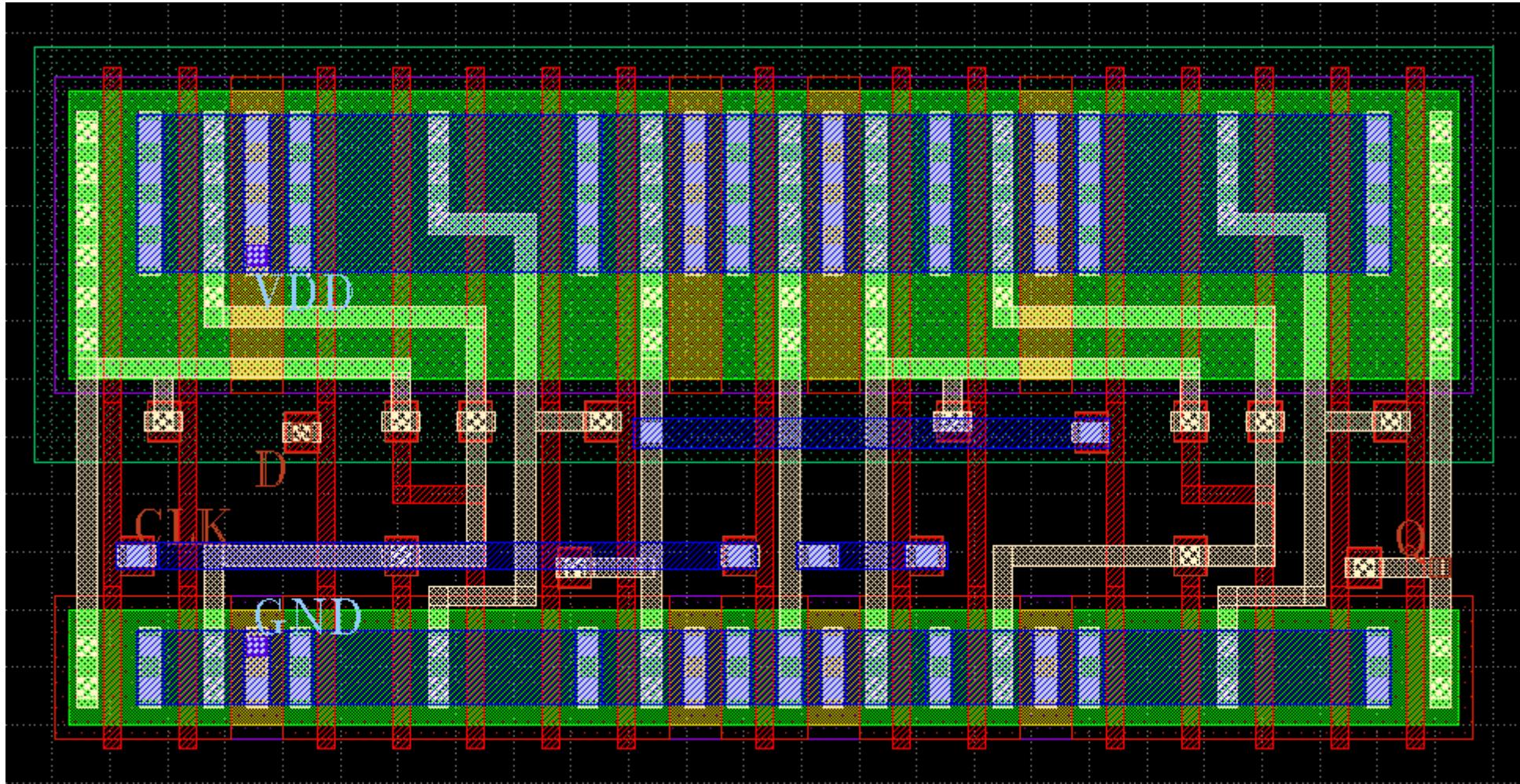
TC4013BP(左)とTC74HC74(右)でのDフリップフロップ構成例



Dラッチを2個使用したDフリップフロップ構成例

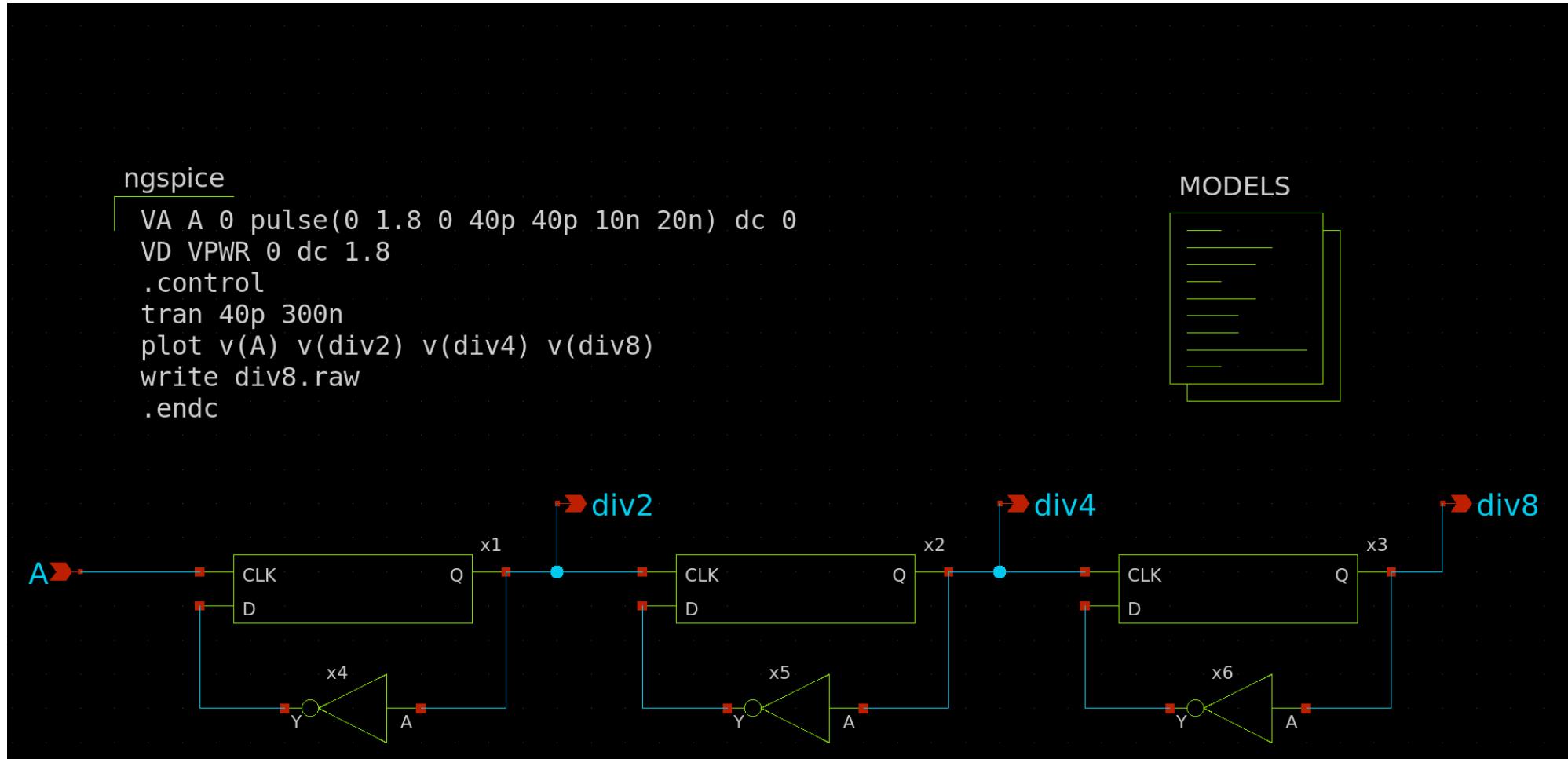


Dラッチを2個使用したDフリップフロップ構成例

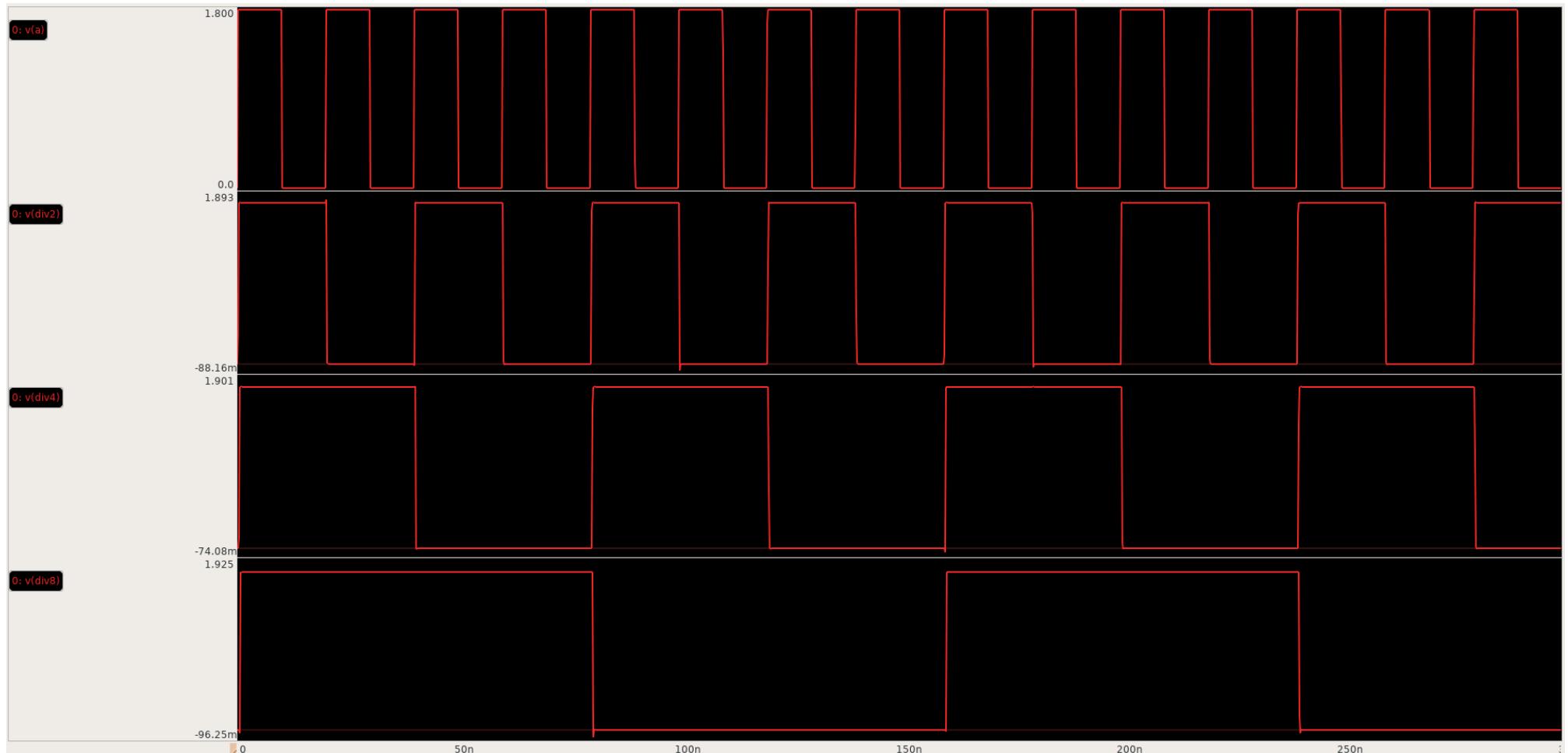


f/2, f/4, f/8 Clock divider 構成例

fdiv.sch

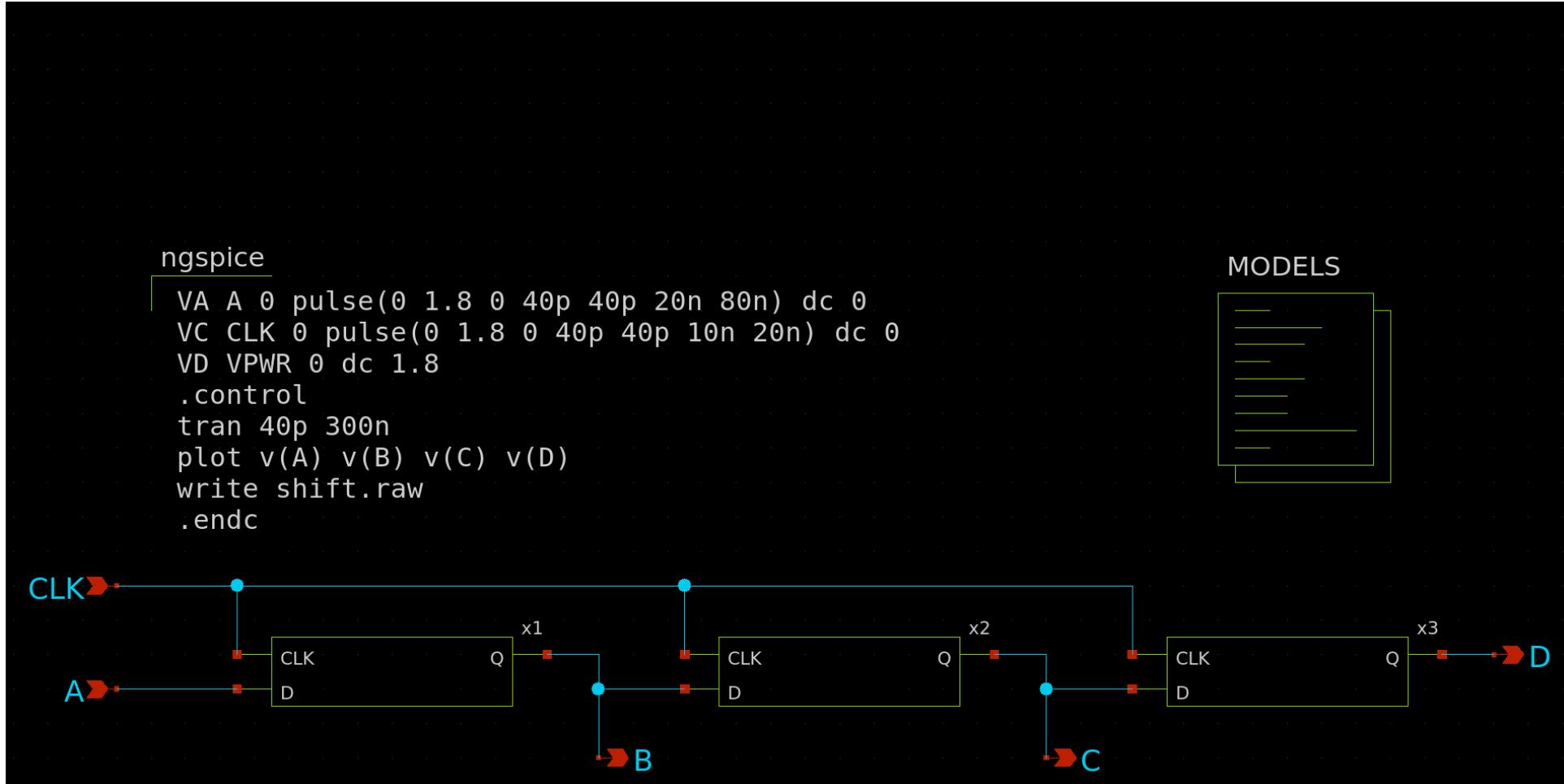


f/2, f/4, f/8 Clock divider 構成例

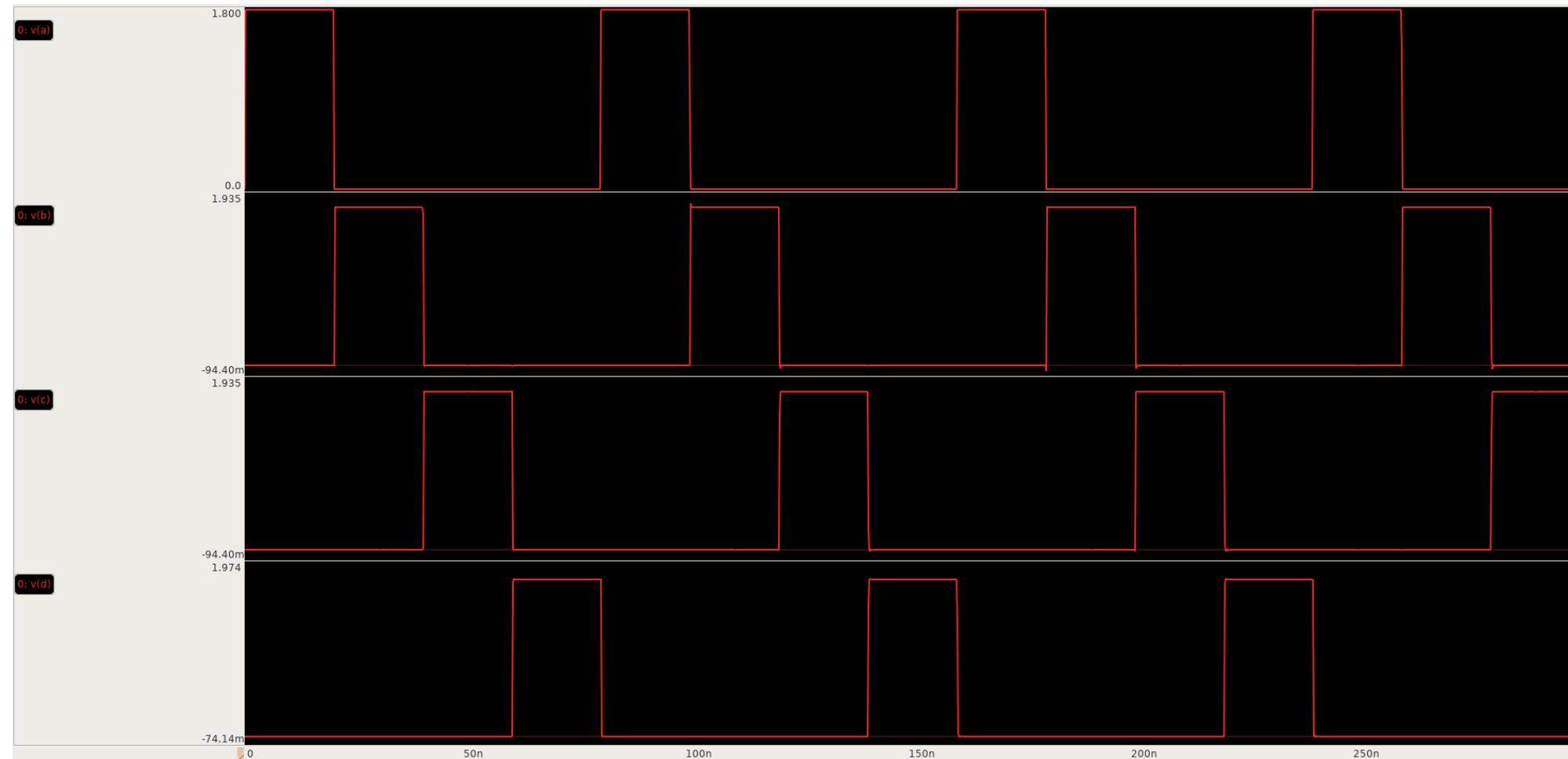


3-bit Shift register 構成例

shift.sch

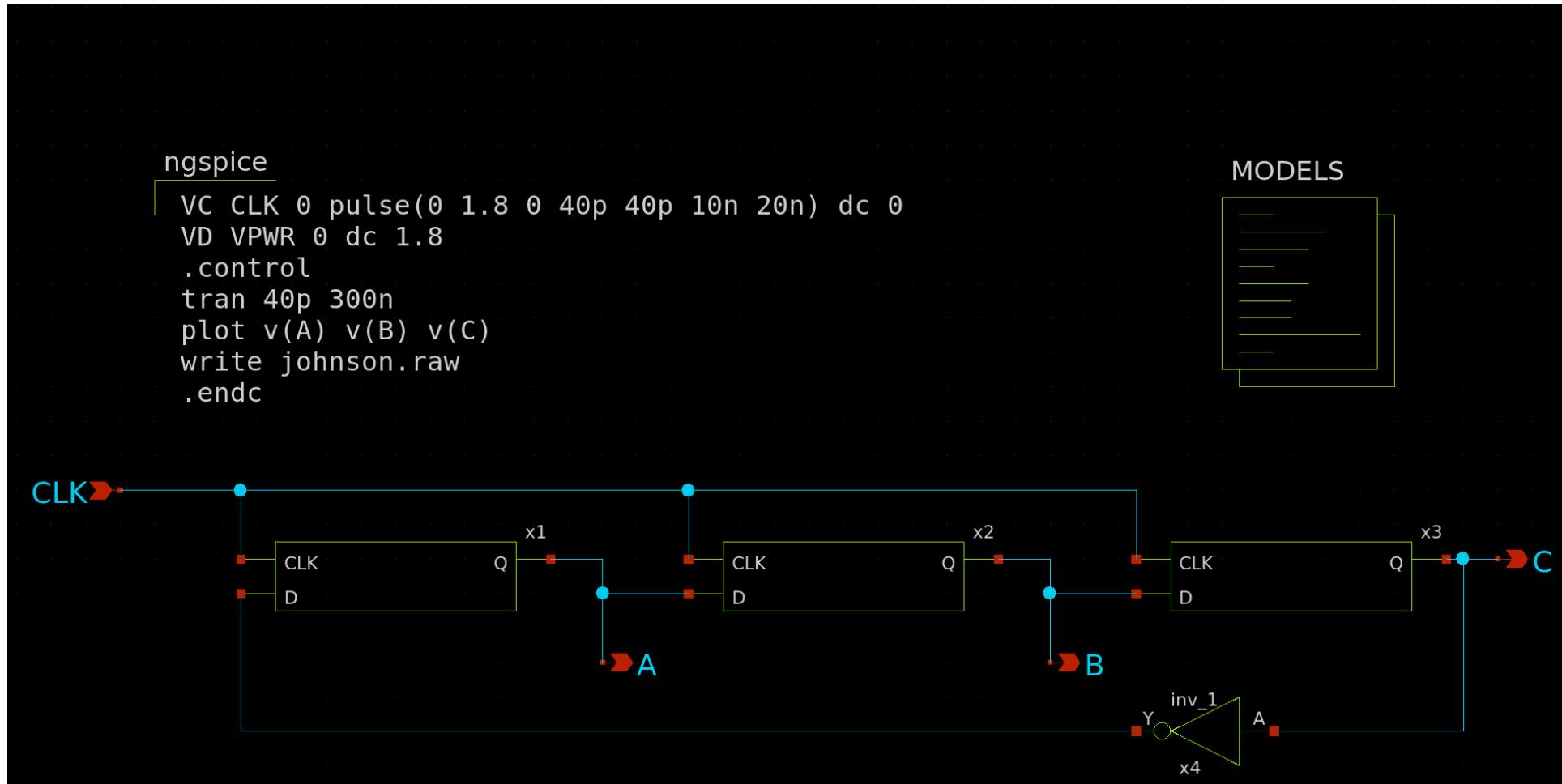


3-bit Shift register 構成例



3-bit Johnson ring counter 構成例

johnson.sch



3-bit Johnson ring counter 構成例

