LibreLane によるデジタル回路設計に挑戦してみたよ！

広島大学 西澤真一 nishizawa@hiroshima-u.ac.jp

広 島 大 学

ISHI-Kai (Japan)

Open-Source EDA supporters (Japan)

自己紹介

- デジタル回路の物理設計に関する技術を開発しています
  - スタンダードセルライブラリの設計
    - 低電圧，低消費電力，耐ばらつきライブラリ
  - コンピュータ援用設計技術 (Computer-Aided Design)
    - 集積回路設計用ツールの開発
    - レイアウトジェネレータ
      - スタンダードセルレイアウトを自動生成
    - キャラクタライザ
      - スタンダードセルの遅延・電力を自動抽出

アジェンダ

- デジタル集積回路の簡単な導入

- LibreLane でオレオレデジタル回路を作成しよう
  - 簡単な RISC-V コアを実装してみよう

- LibreLane でオレオレライブラリを利用したい(願望)
  - 必要なファイルは？
  - オレオレライブラリの作成

- コンピュータの中身，見たことあります？

コンピュータの中身

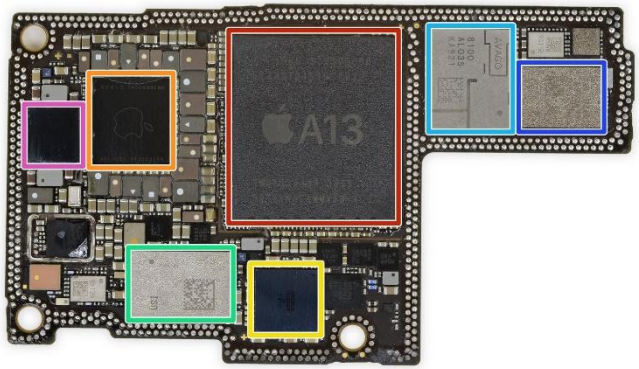
- iPhone 11 Pro Max の分解写真 [1]

ディスプレイとカメラ
(裏返し)

バッテリー

基板に複数のハードウエア (集積回路)
が実装されている

- A13プロセッサ(メモリも同時に集積)
- パワーマネージメントIC, RF, U1 UWBチップなど



脳に相当するプロセッサがデジタル回路

[1] "iPhone 11 Pro Max 分解", iFixit,
https://jp.ifixit.com/Guide/iPhone+11+Pro+Max+の分解 20/11/24閲覧

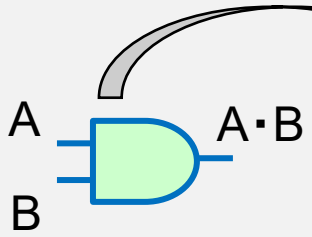
デジタル回路と論理ゲート

- デジタル回路 (論理回路) は入力を論理演算して出力する
  - 入力出力は共に電圧で，高電圧を”1”，低電圧を”0”とみなす
  - 2進数表現，ブール代数と相性がよい
- デジタル回路を構成する基本部品が論理ゲート
  - 論理ゲートを組み合わせて大規模なデジタル回路を作る
  - 参考：nvidia RTX 40シリーズは約170億ゲート (NAND2換算)

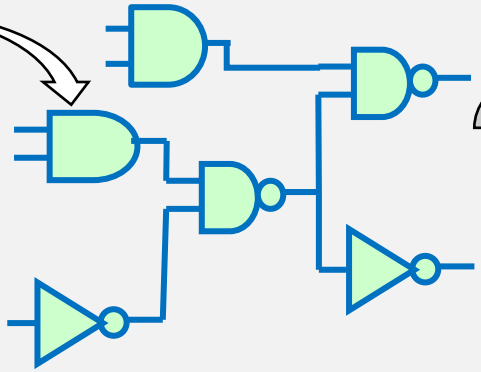
ANDゲート

(A) 真理値表　(B) 回路記号

A	B	A･B
0	0	0
1	0	0
0	1	0
1	1	1



論理回路

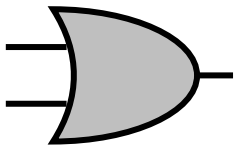


大規模集積回路 (intel Core-i7)

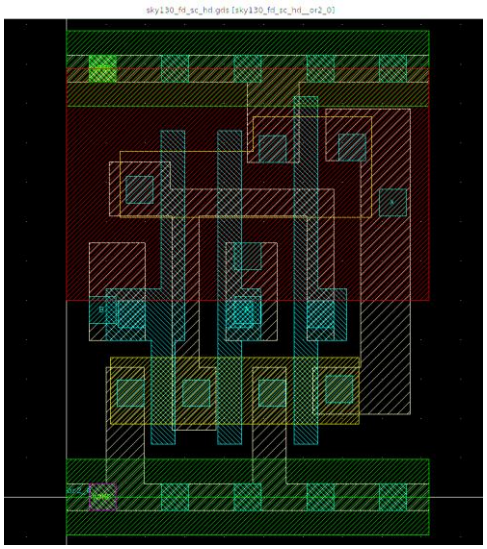
スタンダードセル

*Basic Cellとも

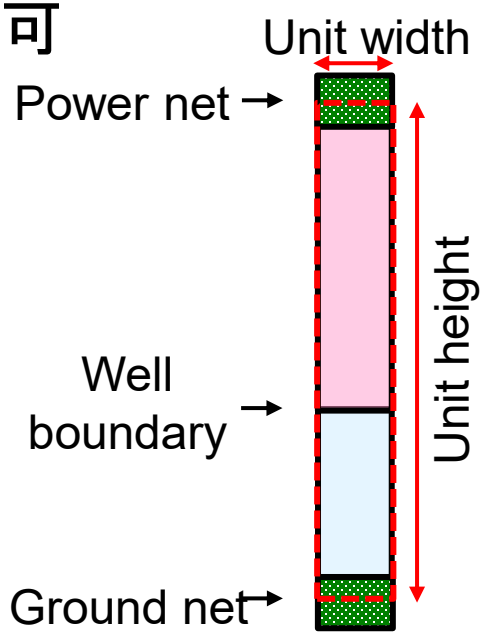
- 特定プロセスで実装された論理素子
  - 論理記号：製造プロセス非依存
  - スタンダードセル：特定の製造プロセスでの論理の実装結果
- ファブが提供するものを用いて回路設計
  - 提供されない場合，自社で作るか設計会社から購入
- スタンダード：形状を規格化，並べて配置可



OR2: 論理記号



OR2: Sky130Aのレイアウト



ライブラリのUnit Cell*

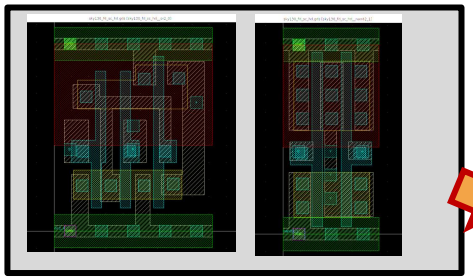
デジタル回路回路のレイアウト設計

- 回路を最終的にレイアウトに変換する
  - レイアウトがシリコンチップに転写されると考える
- 回路を回路記述言語 (HDL) で記述
- HDLをスタンダードセル集合へツールで変換
- セル集合を配置・配線し，性能検証後，データを工場へ
  - セルはトランジスタで構成されている

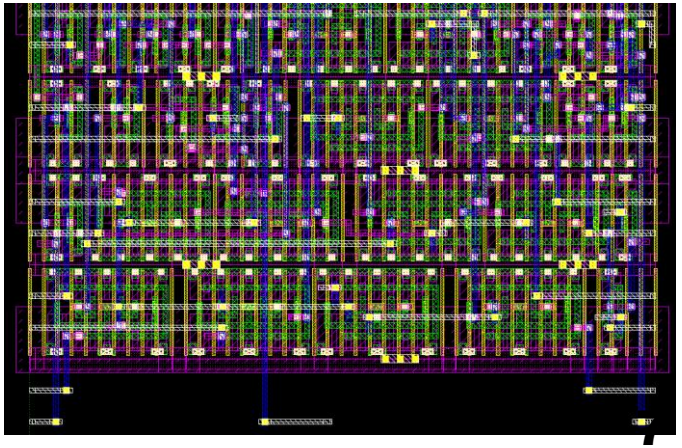
HDL記述

```
always@(A, B, Cin)begin
  {Cout, S} = A+B+Cin;
end
```

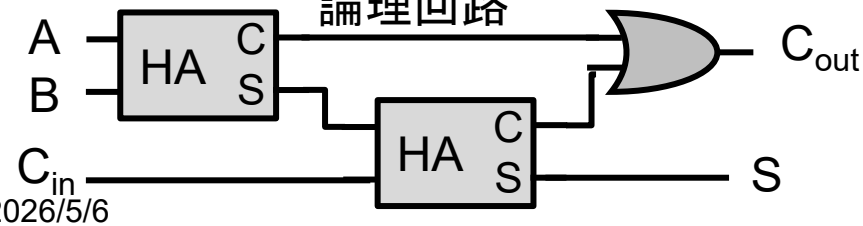
セルライブラリ



回路レイアウト

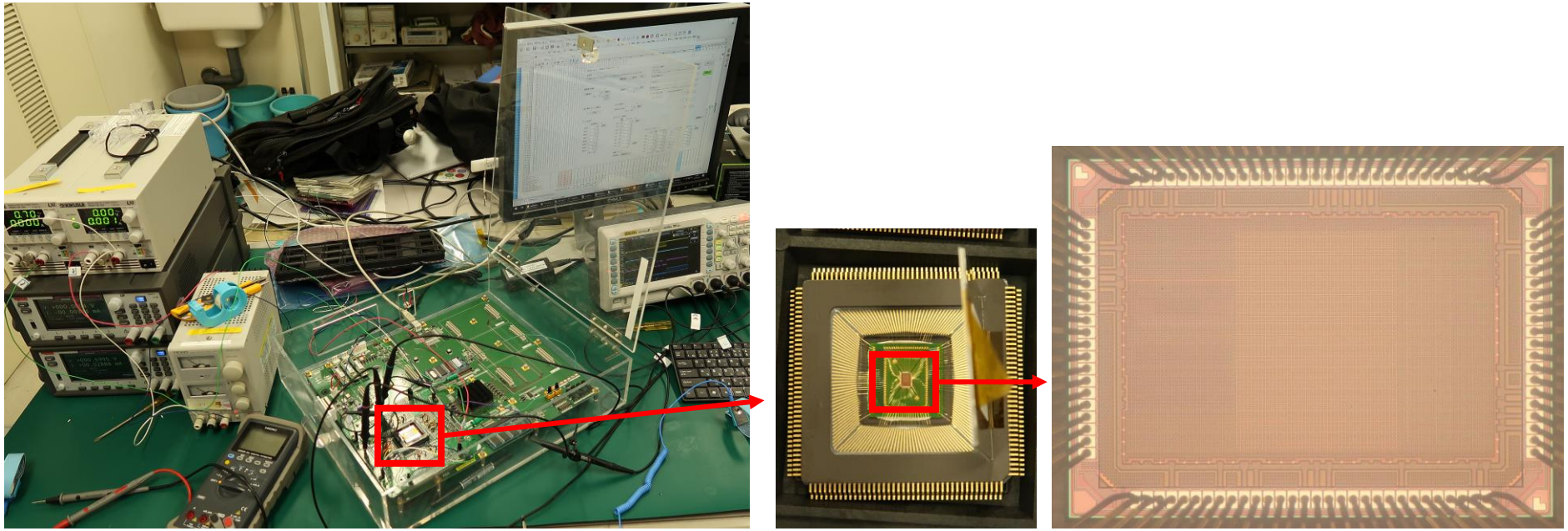


論理回路



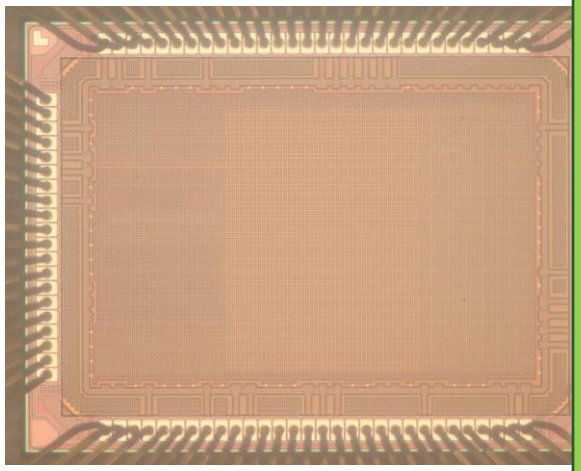
デジタル回路の中身を追ってみよう

- 実装 (正確にはテスト) されているデジタル回路の中身を見る
  - パッケージの中央の豆粒がデジタル回路を含むLSI
  - 顕微鏡を使わないと中身が見えない
  - 65 nm プロセス，2mm x 1.5 mm



デジタル回路の中身を追ってみよう:レイアウト編

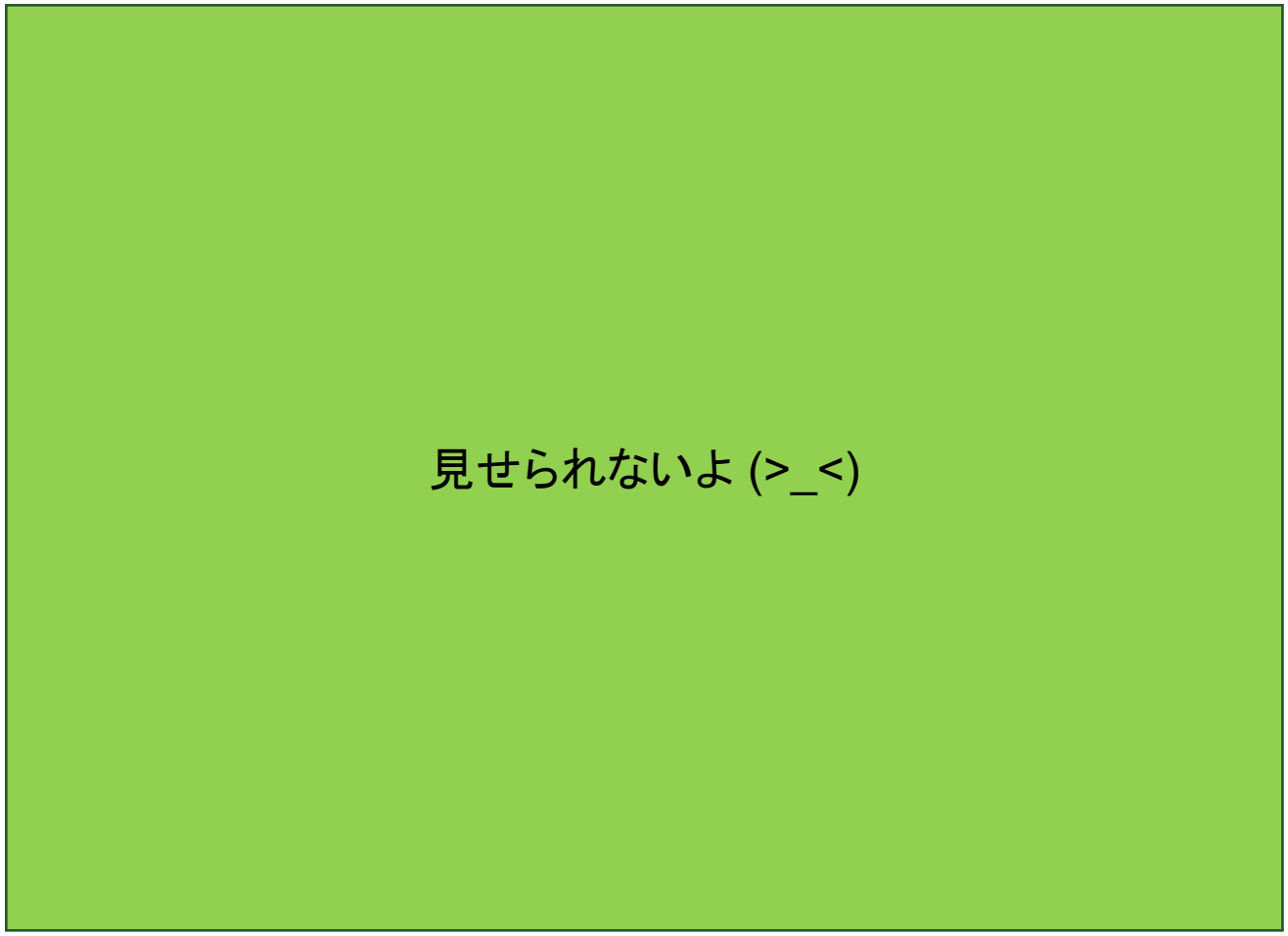
- 設計データ (レイアウト) を見てみる
  - 長方形のブロックが6つ見える
    - 真ん中4つがAES暗号処理のデジタル回路



見せられないよ (>_<)

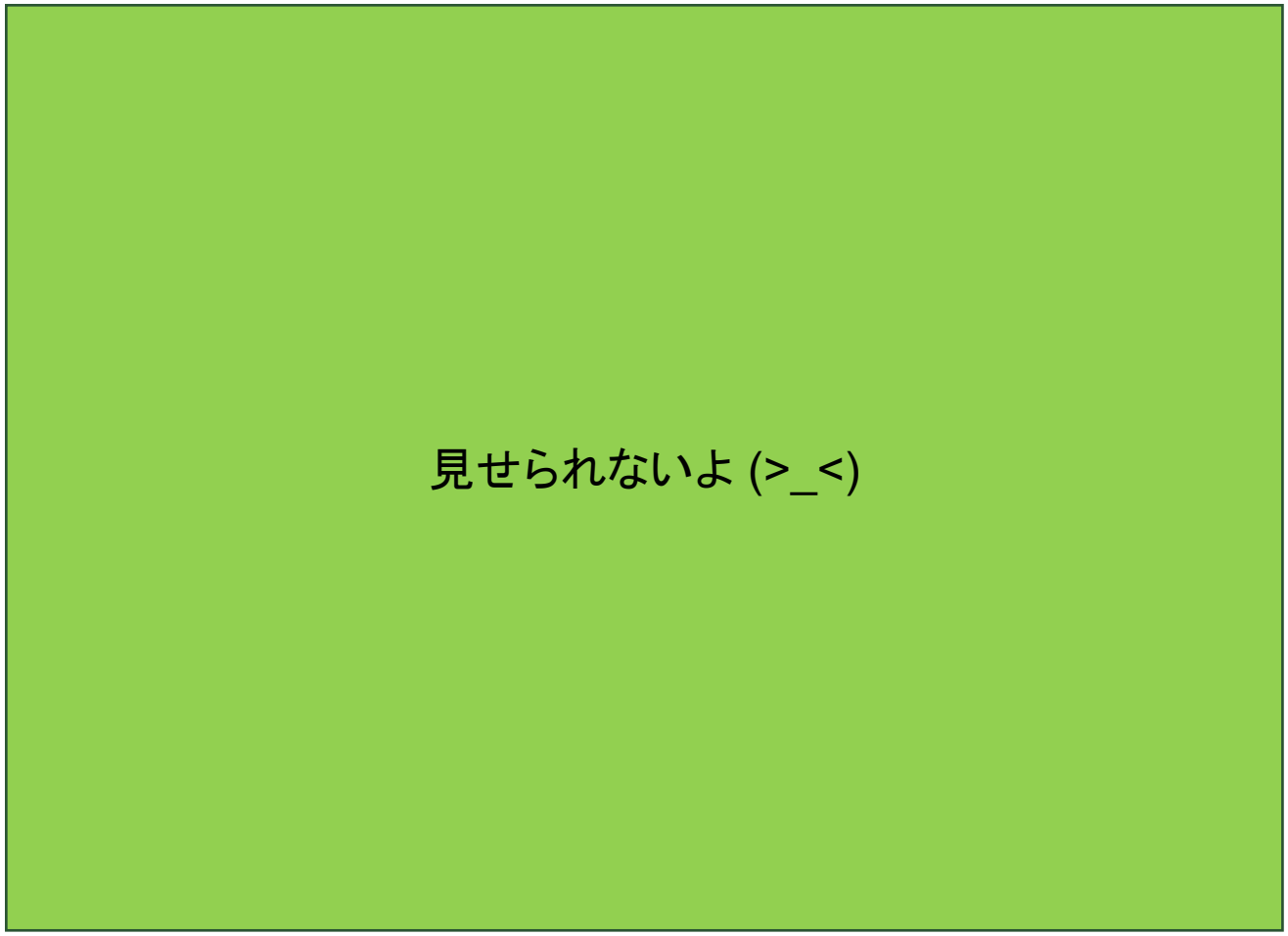
デジタル回路の中身を追ってみよう:レイアウト編

- AES暗号処理のデジタル回路に注目してみる
  - 細かい模様状



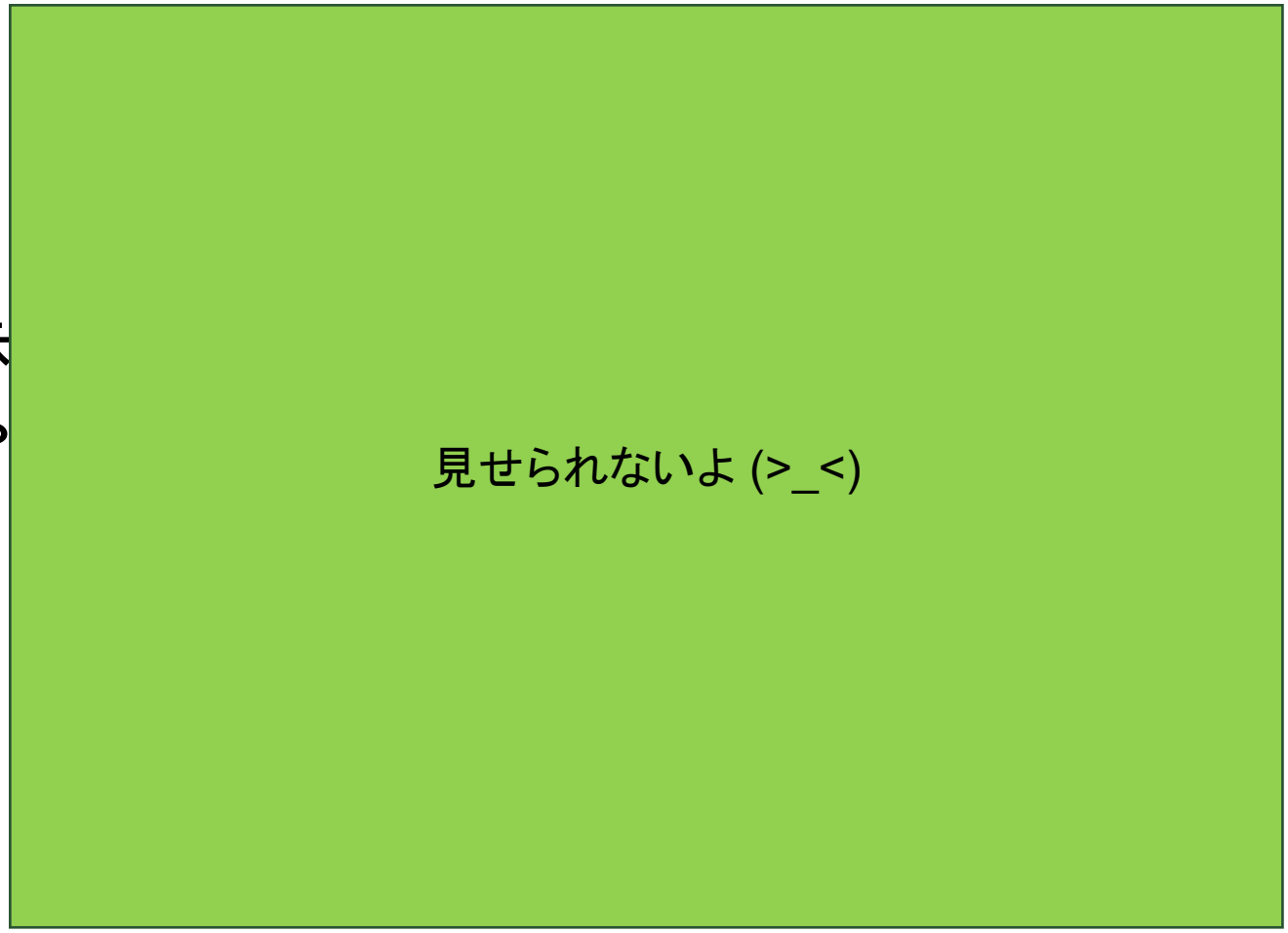
デジタル回路の中身を追ってみよう:レイアウト編

- AES暗号処理のデジタル回路に注目してみる
  - 細かい模様状
  - 拡大してみる
    - よくわからん



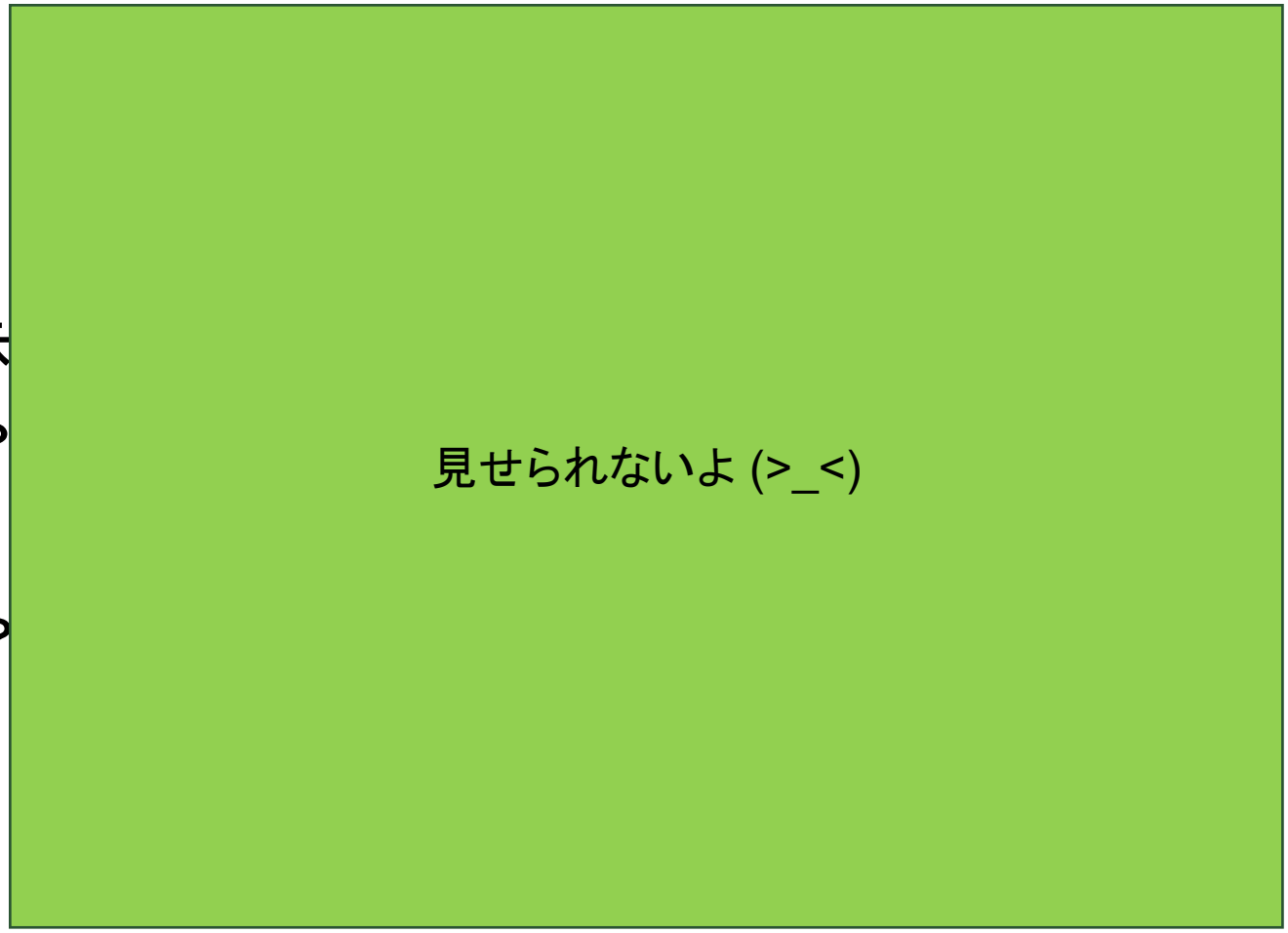
デジタル回路の中身を追ってみよう:レイアウト編

- AES暗号処理のデジタル回路に注目してみる
  - 細かい模様状
  - 拡大してみる
    - よくわからん
  - 一部配線非表示
    - 細かい模様？



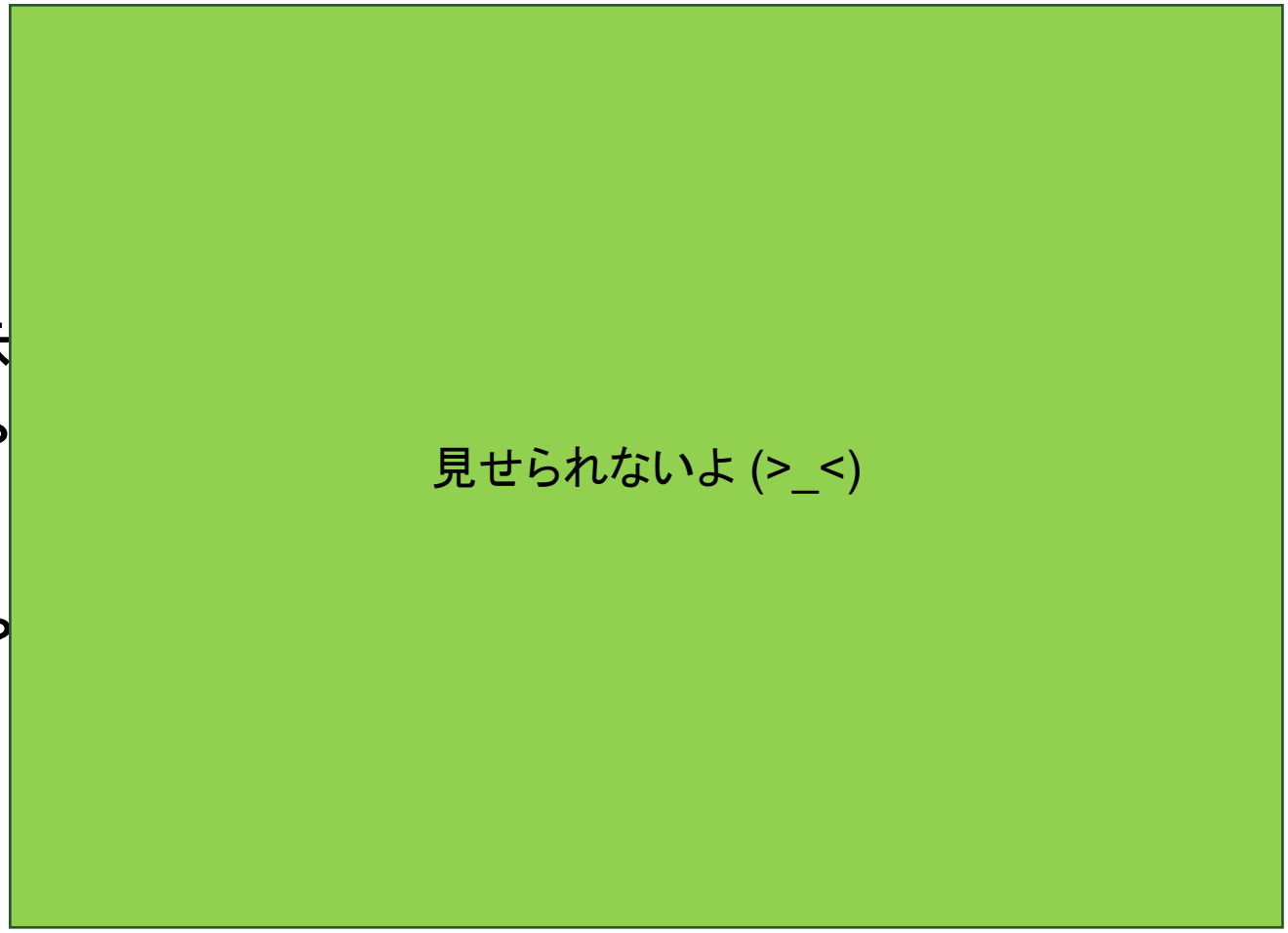
デジタル回路の中身を追ってみよう:レイアウト編

- AES暗号処理のデジタル回路に注目してみる
  - 細かい模様状
  - 拡大してみる
    - よくわからん
  - 一部配線非表示
    - 細かい模様？
  - もっと拡大
    - 青赤は配線？
    - その下は？



デジタル回路の中身を追ってみよう:レイアウト編

- AES暗号処理のデジタル回路に注目してみる
  - 細かい模様状
  - 拡大してみる
    - よくわからん
  - 一部配線非表示
    - 細かい模様？
  - もっと拡大
    - 青赤は配線？
    - その下は？
    - NOR2だ！



デジタル回路とCAD

- デジタル回路は物量が勝負
  - 何十何百億のトランジスタ・セルを組み合わせて設計
- 大規模すぎて人手ではとても設計できない
  - コンピュータによるコンピュータ援用設計技術 (CAD) を用いる
  - そのためには CAD ツールの習熟が必須

LibreLane [2]

OSEDA: Open-Source Electric Design Automation
*1: OpenLane 2.0 と当初呼称していた

- (旧) eFabress のデジタル回路設計ツールチェーン*1
  - RTL を入力し回路レイアウトを出力 (RTL-to-GDS)
  - 各ステップは既存の OSEDA を利用
    - OpenDB を用いて OSEDA をシームレスに接続
  - 130 nm 世代のプロセステクノロジ向けにチューンされている

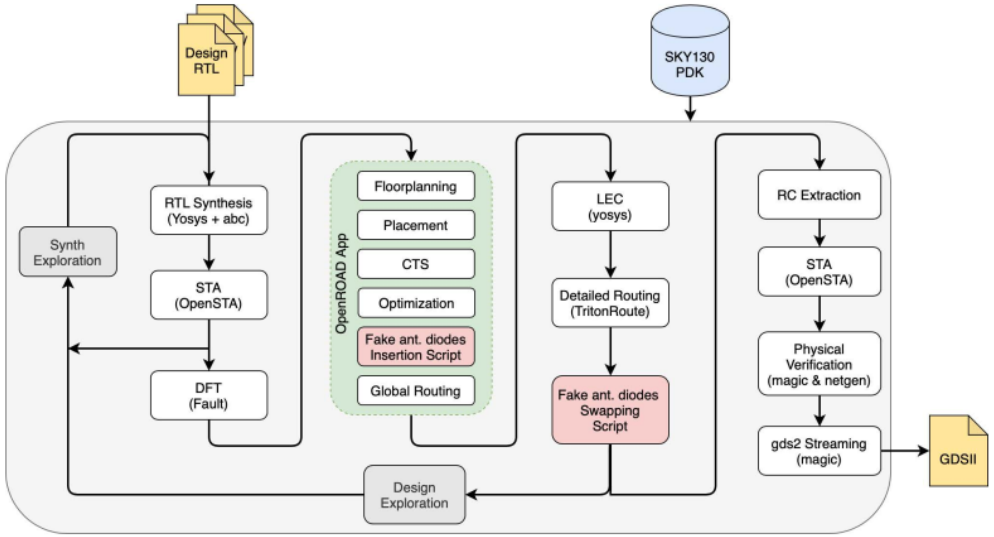


Figure 2. The *OpenLANE* Flow.

Table 2. *OpenLANE* Tools.

DESIGN STEP	EDA Tool
Logic Synthesis	Yosys and ABC
DFT Scan Insertion	Fault
DFT ATPG	Fault
Formal Verification	Yosys
Placement	RePlAce and OpenDP
Routing	FastRoute and TritonRoute
CTS	TritonCTS
Extraction	Magic
Timing Analysis	OpenSTA
Chip Floorplanning	PADGen
LVS	Netgen
DRC	Magic
GDS Streaming Out	Magic

[2] M. Shalan and T. Edwards, Tim, “Building OpenLANE: A 130nm OpenROAD-based Tapeout- Proven Flow : Invited Paper,” in *ICCAD*, 2020, pp.1-6

LibreLaneフロー

- 論理合成：Yosys
- テクノロジマッピング：ABC
- DFT：Fault
- フロアプラン：内製
- セル配置：RePlAce, OpenDP
- クロック合成：TritonCTS
- グローバル配線：FastRoute
- 詳細配線：FastRoute
- 等価性検証：Yosys
- アンテナ検証：Magic
- DRC検証：Magic
- LVS検証：nertgen
- タイミング検証：OpenSTA
- 設計空間探査 (16種以上)

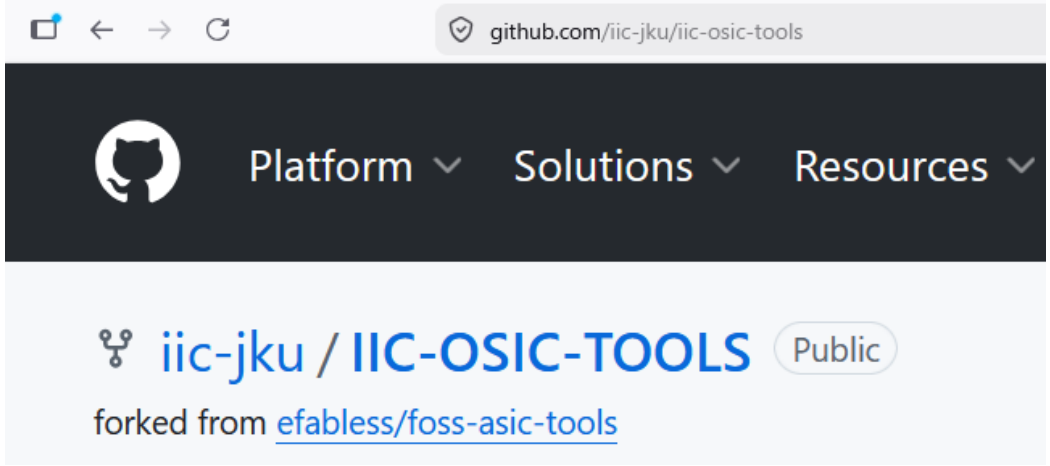
- 概ね商用と同じ機能を持つ
- Synopsys Design Constraint (SDC) 非対応
  - タイミングクロージャが苦手

LibreLane をお手軽に試してみる

- IIC-OSIC を使うとお手軽
  - Institute for Integrated Circuits (IIC), Jophannes Kepler University Linz (JKU) の提供するツールセット
  - Docker のコンテナを利用できる
  - 導入方法は土谷先生の Note にまとまっています
    - https://note.com/akira_tsuchiya/n/nf770aa77b785
    - (ありがとうございます)

IIC-OSIC ツールの実行

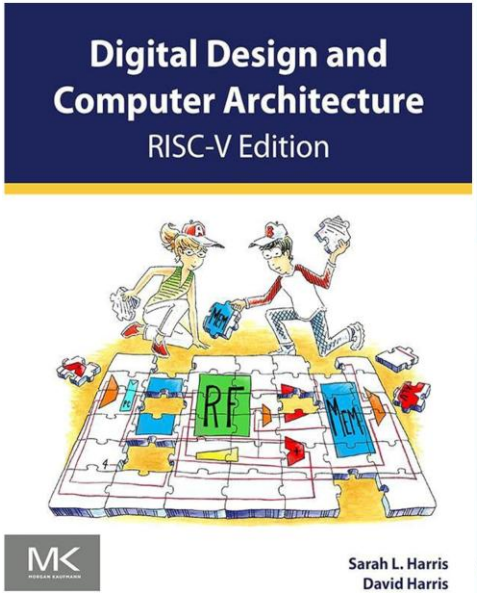
- 公式ビデオを参考にする
  - https://www.youtube.com/watch?v=AiFTwKdS2V4
- 動画スライド p17 に Tutorial ファイルがある
  - https://github.com/iic-jku/kvic_336007_ws25



- Docker 内の設計ファイルは以下のパスでアクセス
  - C:¥Users¥[ユーザー名]¥eda¥designs

設計練習：RISC-V

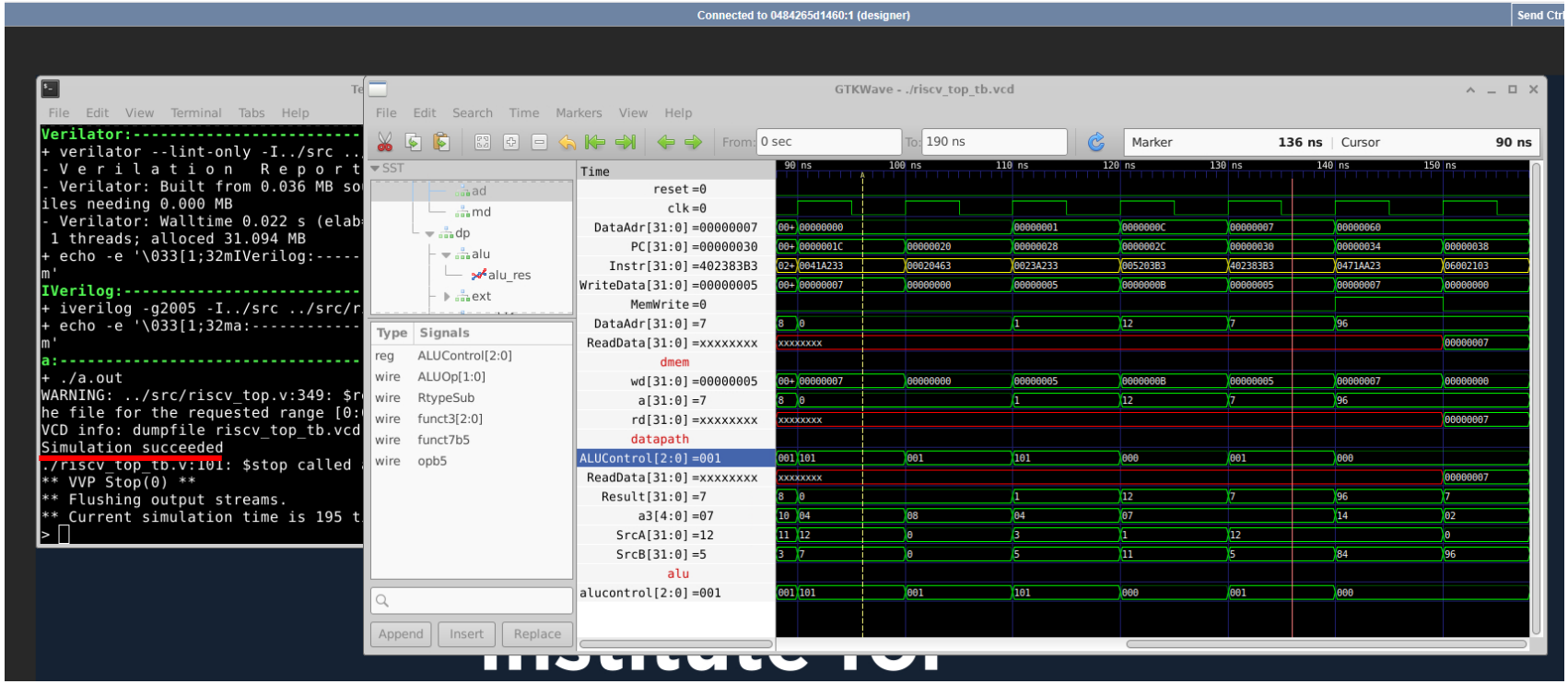
- Harris&Harris[3,4] の RISC-V コアを動かしてみよう
- 主な変更点
  - RISC-V コアとテストを分離，命名ルールを simulate.sh に合わせる(回路.v，回路?_tb.v)
  - 記述を SystemVerilog から Verilog に
  - メモリアラインメントを明示的に
    - rd = RAM[{2'b0,a[31:2]}];



[3] S. Harris and D. Harris, "Digital Design and Computer Architecture, RISC-V Edition", Morgan Kaufmann, 2021.
[4] サラ・L・ハリス，デイビッド・ハリス，"ハリス＆ハリスの『ディジタル回路設計とコンピュータアーキテクチャ【RISC―V版】", 株式会社エスアイビー・アクセス，2022年.

シミュレーション検証

- 付属テストベンチにて検証
  - DataAdr=100 の時に WriteData=25 であるか判定
  - “Simulation succeeded” または "Simulation failed"と表示



LibreLane による実装

- 回路に応じて config.json と pin_order.cfg を変更
  - config.json : RTL の名前と場所，面積
    - 面積は160 150 が初期値，おそらく整数倍がよい

```
"DESIGN_NAME": "riscv_top",
"VERILOG_FILES": "dir::../verilog/src/riscv_top.v",
"DIE_AREA": "0 0 320 300",
…(以下略)
```

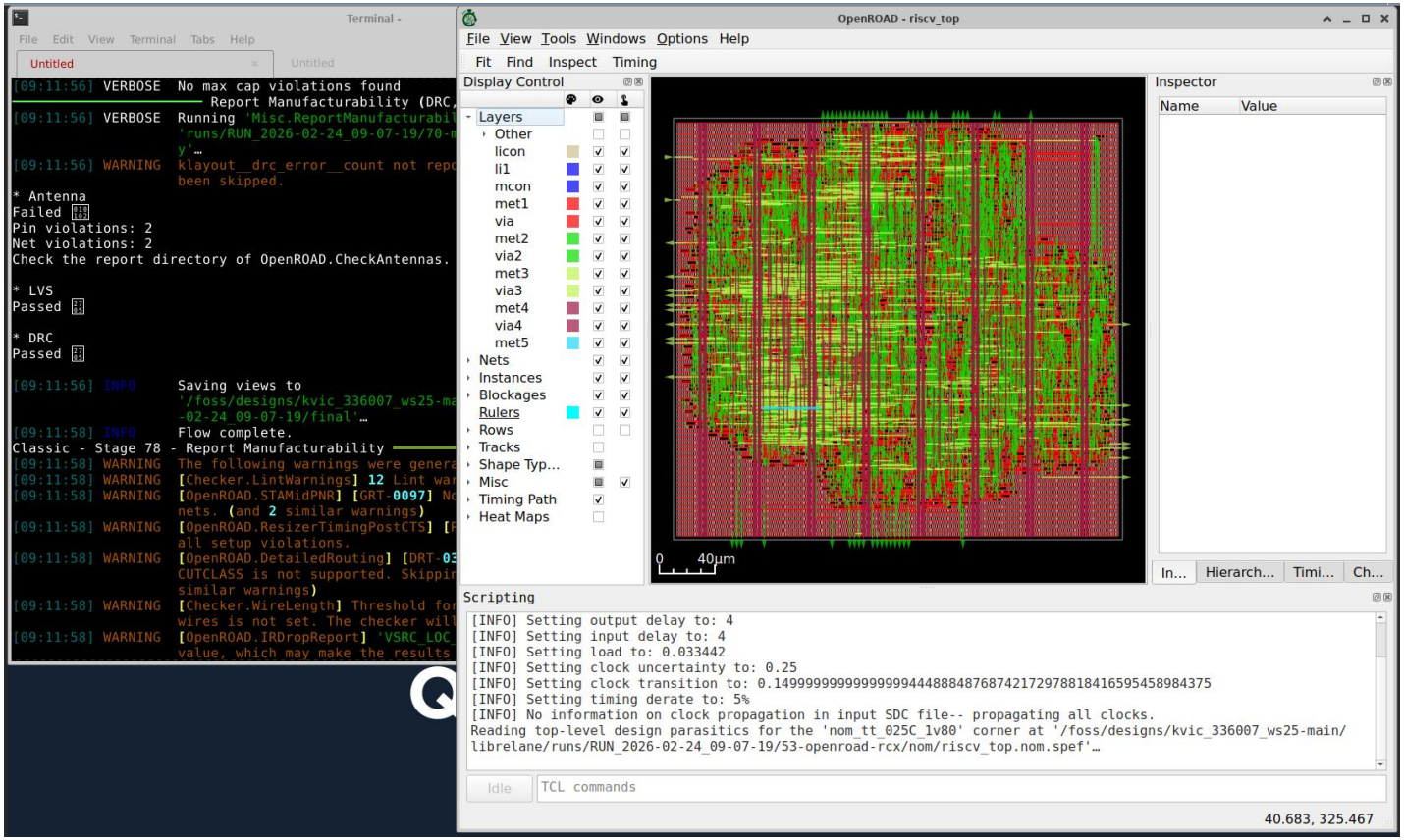
  - pin_order.cfg: ピンの名前と場所の定義

```
#E
WriteData[0]
WriteData[1]
…(以下略)
```

  - ./run_librelane.sh より実行

レイアウト例

- それっぽいレイアウトが生成された
  - DRC/LVS はパス，Antenna 違反がいくつか
    - (おそらく命令メモリが ROM になり電源に接続されたため)



LibreLane でオレオレライブラリを使いたい (願望)

- オレオレライブラリを利用するには PDK の変更が必要
  - PDK: Process Design Kit
  - 設計上必要な情報・データファイル一式
- ここではセルライブラリの情報が必要
  - レイアウト情報：LEF (Library Exchange Format)
  - タイミング情報：.lib (Liberty Format)

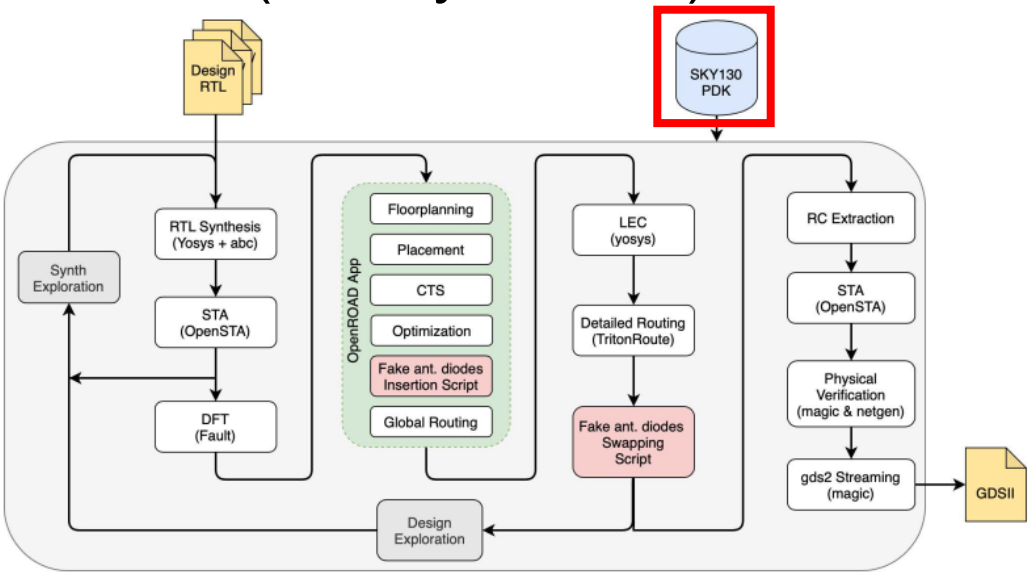


Figure 2. The *OpenLANE* Flow.

格納場所

- IIC-OSIC では以下に存在
  - ライブラリ

```
/foss/designs/kvic_336007_ws25-main/librelane > vi /foss/pdks/sky130A/libs.ref/sky130_fd_sc_hd/
cdl/     gds/     lef/     lib/     mag/     maglef/  spice/   techlef/ verilog/
```

  - ライブラリを LibreLane で読む設定 (config.tcl)

```
> vi /foss/pdks/sky130A/libs.tech/openlane/sky130_fd_sc_hd/config.tcl
```

  - config.tcl の記述

```
# Technology lib

set ::env(LIB_SYNTH) "$::env(PDK_ROOT)/$::env(PDK)/libs.ref/$::env(STD_CELL_LIBRARY)/lib/sky130_fd_sc_hd__tt_025C_1v80.lib"
set ::env(LIB_FASTEST) "$::env(PDK_ROOT)/$::env(PDK)/libs.ref/$::env(STD_CELL_LIBRARY)/lib/sky130_fd_sc_hd__ff_n40C_1v95.lib"
set ::env(LIB_SLOWEST) "$::env(PDK_ROOT)/$::env(PDK)/libs.ref/$::env(STD_CELL_LIBRARY)/lib/sky130_fd_sc_hd__ss_100C_1v60.lib"

set ::env(LIB_TYPICAL) $::env(LIB_SYNTH)
```

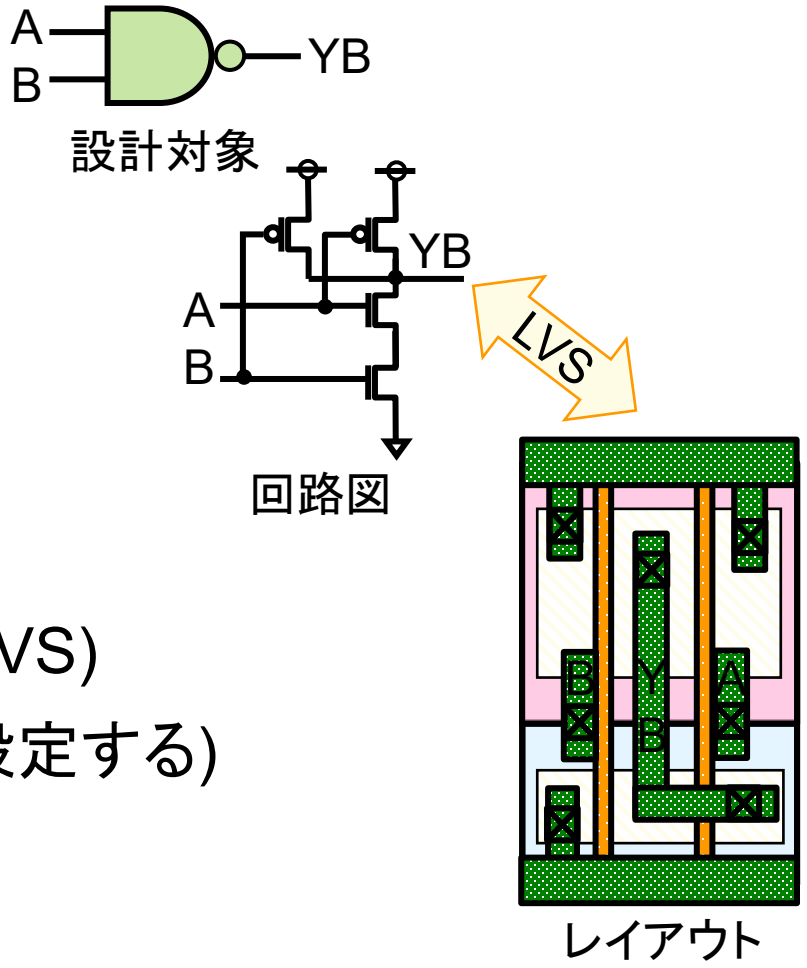
- これらを書き換えればオレオレライブラリ・セルを使えそう

オレオレセルを作ろう

- どうやってセルを作ろう？
  - レイアウト：手で頑張って描く
  - タイミング：SPICE シミュレーション
    - フリー SPICE (ngspice) がある

スタンダードセルを手で設計しよう

- オレオレスタンダードセルを作る
  - 対象を決める
  - 回路図 (Schematic) を描く
  - 回路図を満たすレイアウトを描く
  - 検証
    - Design Rule Check (DRC)
    - Layout Versus Schematic (LVS)
  - (本当は電気特性の目標なども設定する)

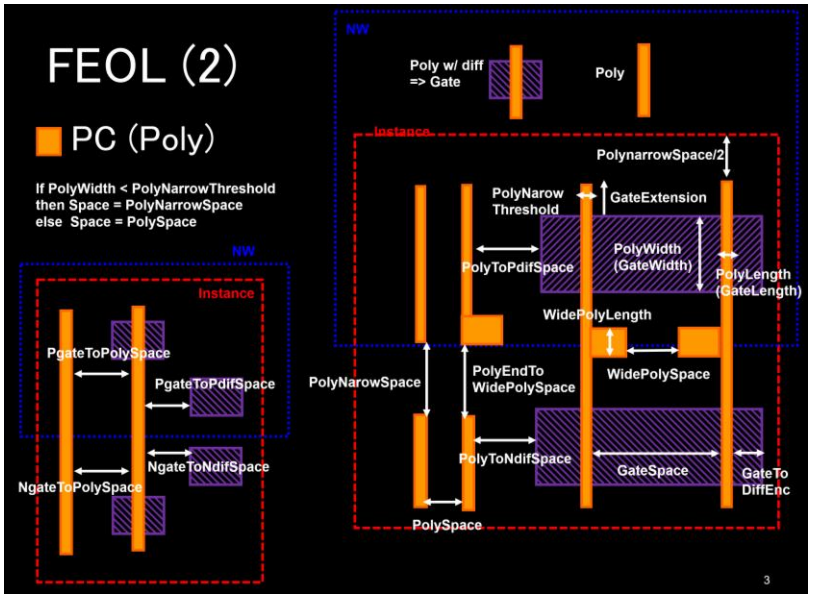


レイアウト設計

- トランジスタおよび配線を適切にレイアウトして機能を実現
  - 複数の(マスク) レイヤーを重ね合わせて実現
    - Sky130AのNAND2で約30レイヤー
  - すべてのレイヤーが設計ルールを満たす必要がある
    - 幅，形状，スペース，重ね合わせなど…(Design Rule)
  - その上でコンパクトなレイアウトの実現が求められる

Sky130A NAND2 セル

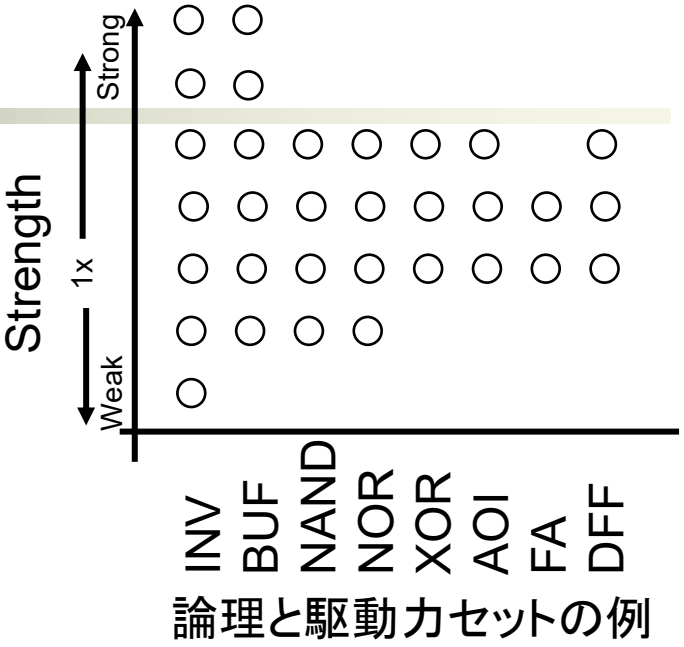
使用するレイヤー



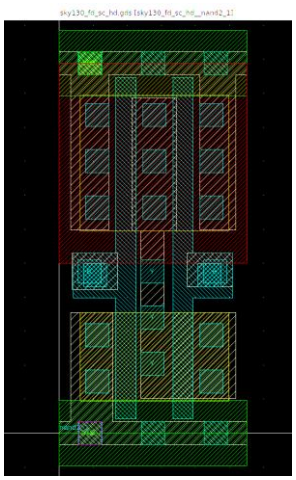
デザインルール例

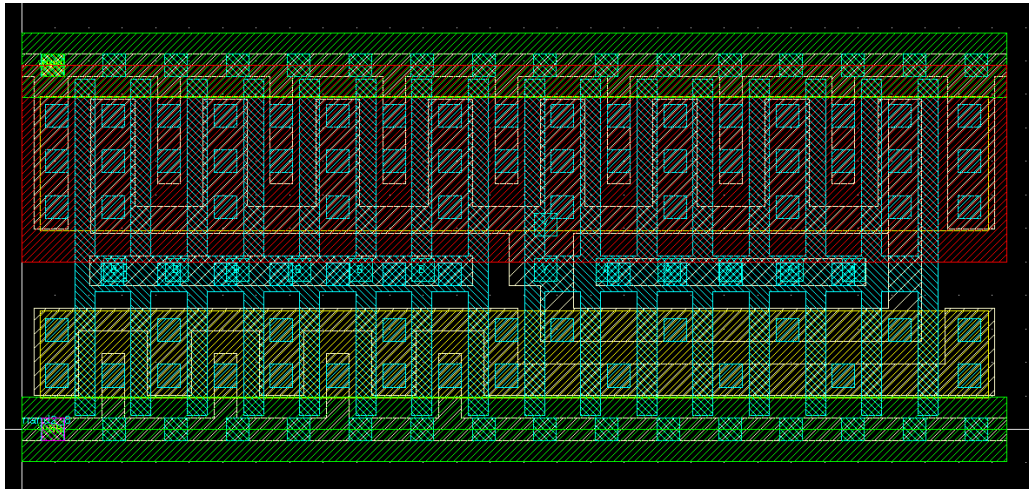
スタンダードセルの種類

- 現実の回路では複数のセル品種が必要
  - 論理の多様性
    - Inv., AND, OR, NAND, NOR, XOR
  - 駆動力の多様性
    - 速度・電力・面積のトレードオフ
  - 閾値電圧の多様性
    - 高速・標準・低リーク



論理と駆動力セットの例





NAND2_1X (baseline)　　NAND2_8X (8-parallel)

100~300 cells in one library

x

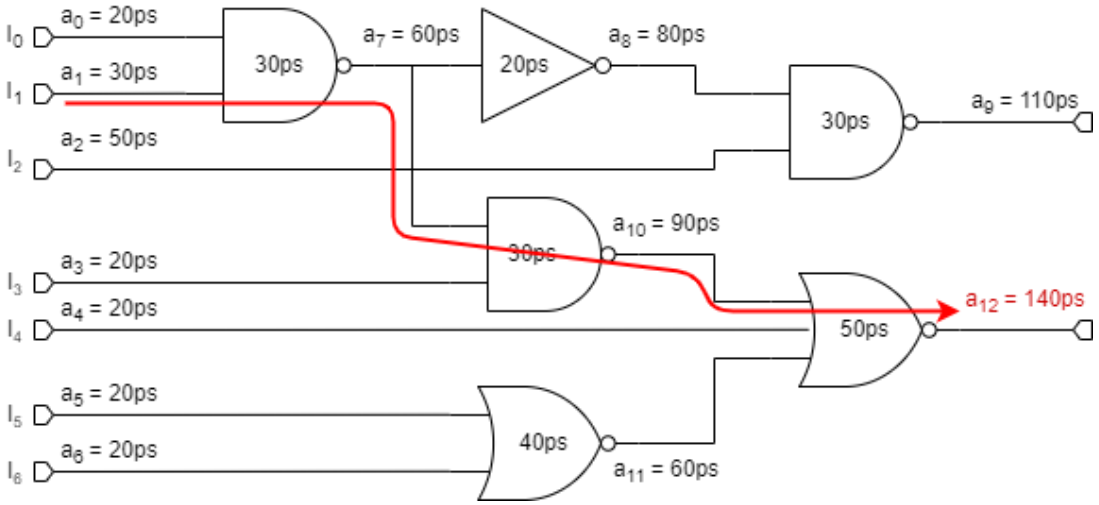
Vth options

=

人手設計は辛い ☹

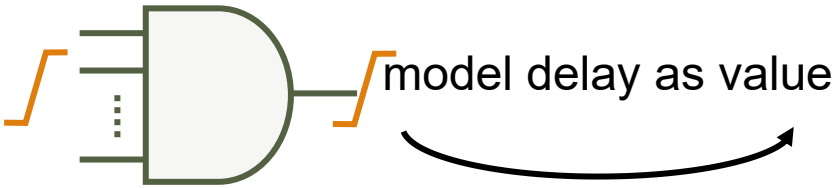
タイミング解析とセルのタイミングモデル

- 回路遅延の評価：Static Timing Analysis (STA)
  - パス中のセル遅延の和からパスの最大・最小遅延を計算
    - パスのi-番目のノードの信号到着時刻
      - $a_i = \max_{j \in \mathrm{fanin}(i)}\{a_j\} + t_{\mathrm{pd},i}$
  - STAのエンジンとセル遅延情報 (.lib) が必要
    - STA: OpenSTA (Open-ROAD), sta (yosis)
    - .lib (Liberty): SPICE シミュレーションで求める



セルのタイミングライブラリ

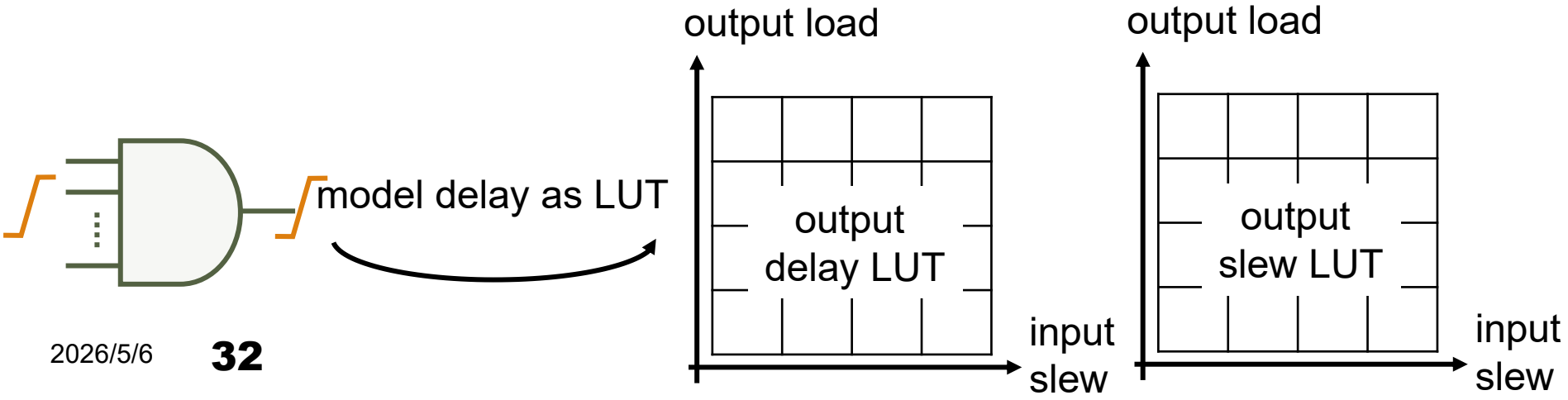
- セルの遅延と電力をSPICEで評価しライブラリ化
  - 伝搬遅延: t_{pd} = xx ps. 遷移遅延: t_{td} = xx ps
    - 信号到着時刻: $a_i = \max_{j \in \mathrm{fanin}(i)}\{a_j\} + t_{\mathrm{pd},i}$



Prop delay: t_{pd} = xx ps.
Trans delay: t_{td} = xx ps

セルのタイミングライブラリ

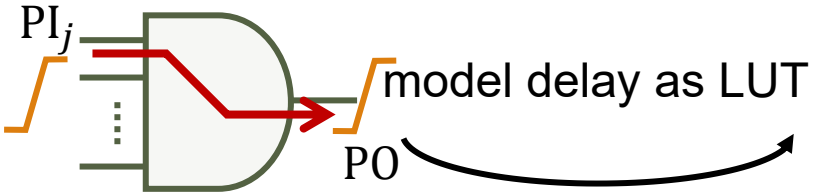
- セルの遅延と電力をSPICEで評価しライブラリ化
  - 伝搬遅延: t_{pd} = xx ps. 遷移遅延: t_{td} = xx ps
    - ✖信号到着時刻: $a_i = \max_{j \in \mathrm{fanin}(i)}\{a_j\} + t_{\mathrm{pd},i}$
- 遅延と電力はセルの入力遷移遅延と出力負荷容量に依存
  - 信号到着時刻: $a_i = \max_{j \in \mathrm{fanin}(i)}\{a_j\} + t_{\mathrm{pd},i}(t_{\mathrm{pd,PI}_j}, C_{\mathrm{load},i})$

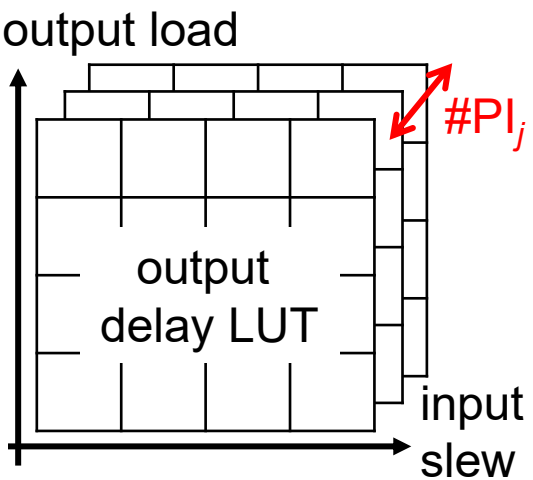


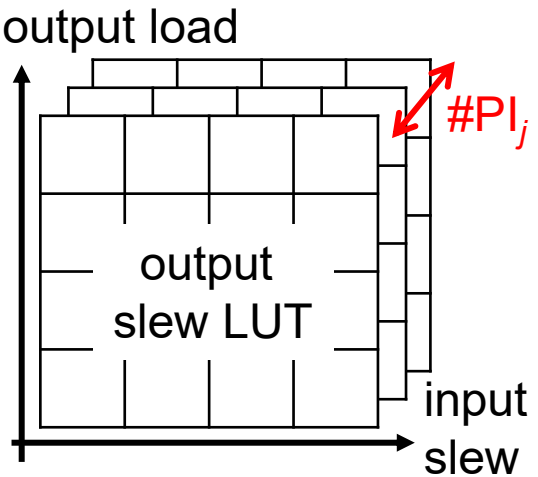
セルのタイミングライブラリ

- セルの遅延と電力をSPICEで評価しライブラリ化
  - 伝搬遅延: t_{pd} = xx ps. 遷移遅延: t_{td} = xx ps
    - ✗信号到着時刻: $a_i = \max_{j \in \mathrm{fanin}(i)}\{a_j\} + t_{\mathrm{pd},i}$
- 遅延と電力はセルの入力遷移遅延と出力負荷容量に依存
  - ✗信号到着時刻: $a_i = \max_{j \in \mathrm{fanin}(i)}\{a_j\} + t_{\mathrm{pd},i}(t_{\mathrm{pd,PI}_j}, C_{\mathrm{load},i})$
- セルの入力 (PI) は異なる遅延・電力特性を持つ
  - 信号到着時刻: $a_i = \max_{j \in \mathrm{fanin}(i)}\{a_j\} + t_{\mathrm{pd},i,\mathrm{PI}_j}(t_{\mathrm{pd,PI}_j}, C_{\mathrm{load},i})$

複数セルに対して SPICE シミュレーション手動実行は辛い☹





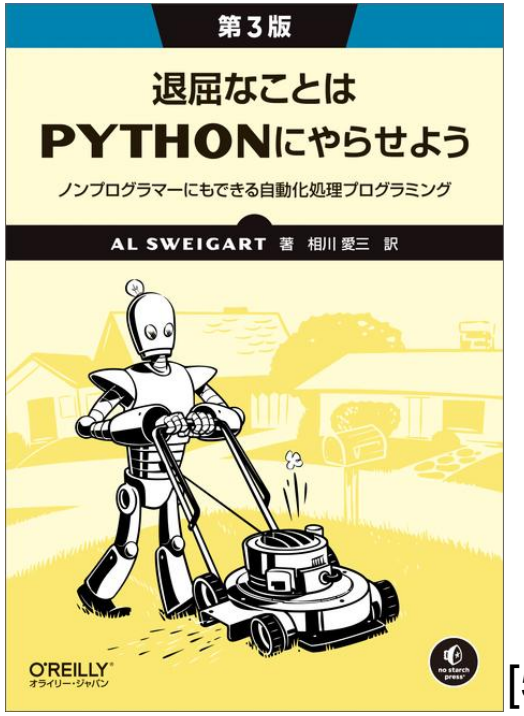


オレオレセルを作ろう…(弱気)

- どうやってセルを作ろう？
  - レイアウト：手で頑張って描く→フルセット手描きはキツい
  - タイミング：SPICE シミュレーション→人手は非現実的

オレオレセルを作ろう…(弱気)

- どうやってセルを作ろう？
  - レイアウト：手で頑張って描く→フルセット手描きはキツい
  - タイミング：SPICE シミュレーション→人手は非現実的
- 退屈なことはプログラムにやらせよう
  - OSEDA を作ろう・使おう
  - レイアウトジェネレータ (レイアウト生成)
  - キャラクタライザ (タイミング抽出)

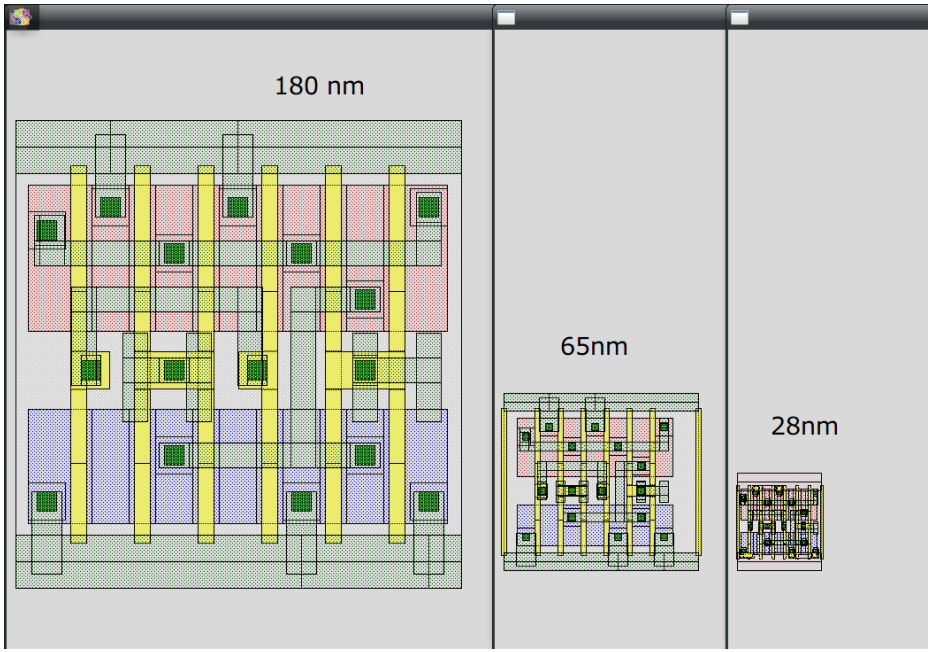


[5]

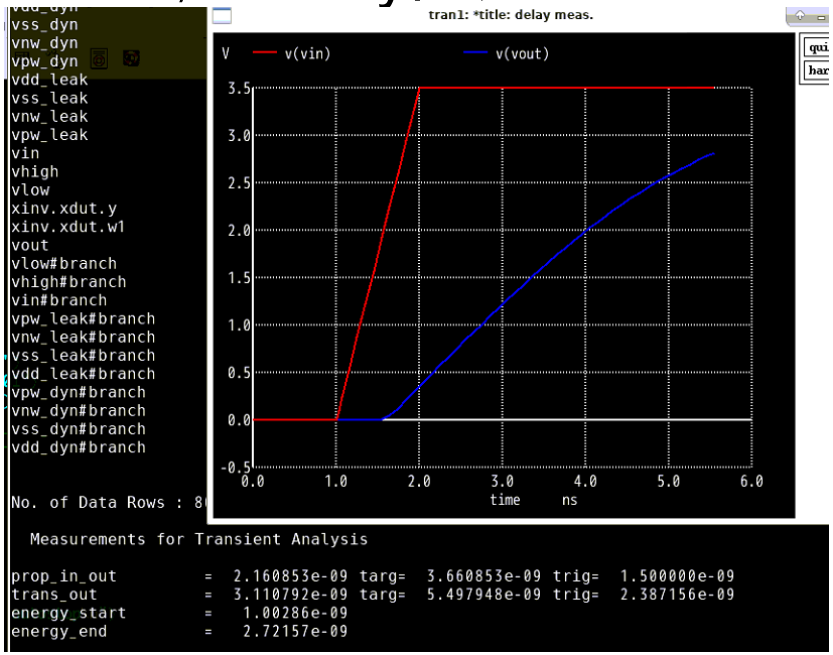
[5] Al Sweigart 著，相川 愛三 訳，“退屈なことはPythonにやらせよう”，オライリージャパン

PR: Open-Source EDAs for library design

- レイアウトジェネレータ[6] (OSEDAにしてません)
  - プロセス非依存の構造から特定プロセス向けレイアウト生成
- キャラクタライザ[7] (タイミング抽出)
  - SPICE ファイル生成，シミュレーション，Liberty出力



異なるプロセステクノロジ向けレイアウト生成



キャラクタライザによる自動sim

[6] S. Nishizawa et al., “Layout Generator with Flexible Grid Assignment for Area Efficient Standard Cell,” *IPSJ TSLDM.*, vol. 8, pp. 131–135, 2015.

[7] S. Nishizawa and T. Nakura, “libretto: An Open Cell Timing Characterizer for Open Source VLSI Design,” *IEICE Trans. Fundam*, vol. E106-A, no. No.3, pp. 551–559, 2023.

まとめ

- アナログ回路も重要だけれど，デジタルもおもしろいよ
  - LibreLane を簡単に紹介
  - オレオレライブラリの導入もしていきたい
- OSEDA 作るのもおもしろいよ
  - 使ってもらえたり，感謝されたり？ (論文化が大きな課題)
  - キャラクタライザ：https://github.com/snishizawa/libretto
    - 使っている場合教えていただけると励みになります
- オープンソースEDA研究会・フォーラムもあります
  - https://www.oseda-silicon.org