Section 1:

CUBE の設計フロー

CUBE チップの設計フロー

松谷 *

平成 22 年 10 月 21 日

もくじ

1　チップ全体の説明

CUBE チップは、以下の４つのモジュールから構成されます（図 1)。

- cube コア： オンチップルータ、コア、バスコントローラなど
- ptp コア 0： point-to-point なワイヤレス通信用のインダクタ
- ptp コア 1： point-to-point なワイヤレス通信用のインダクタ
- sb4 コア ： broadcast bus なワイヤレス通信用のインダクタ

実際には、ptp コア、sb4 コアには、慶應大学・黒田研によるインダクタ回路を用いますが、本設計フローでは論理合成可能なダミーインダクタを使って説明をします。テープアウトの際は、本設計フローを最後まで実行したあとで、ダミーの ptp、sb4 を本物のインダクタマクロに置換しました。

なお、今回の設計では ptp コア、sb4 コアに VDD、VDDBR、VDDBC の 3 電源が必要であるため、電源系統を 3 種類持たせています。詳細は「Appendix A：CUBE_TOP のレイアウトの流れ」をご参照ください。

*matutani@hal.ipc.i.u-tokyo.ac.jp

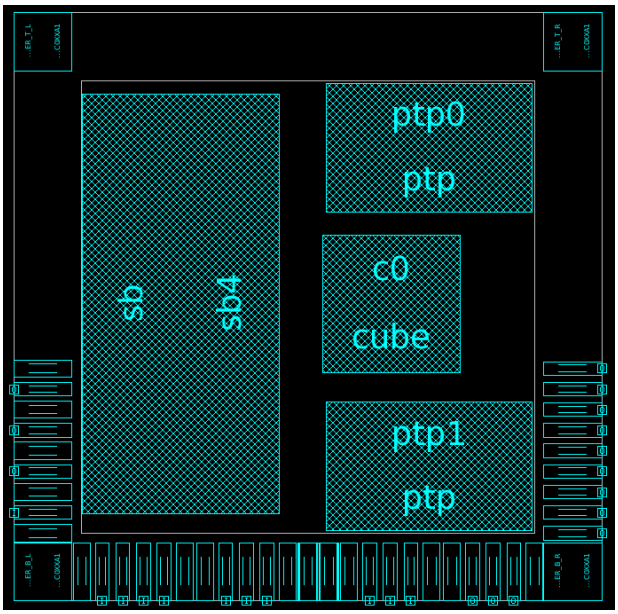


図 1: CUBE チップのフロアプラン。

2 cube コアの動作モードについて

テストモジュール test.v には、以下の４つのモードがあります。

表１：テストモジュールの動作モード

```
=====================================================================
モード０ Point-to-point（Single：毎回、ランダム宛先指定で連続注入）
モード２ Point-to-point（Burst ：同じ宛先にバースト注入）
モード１ Shared-bus     （Single：毎回、ランダム宛先指定で連続注入）
モード３ Shared-bus     （Burst ：同じ宛先にバースト注入）
---------------------------------------------------------------------
```

sim/rom.pl の $mode の値を変更して、event.hex を作ると、各モードに対応した Event ROM が生成されます。詳細は「Section 2：CUBE チップの仕様書」をご参照ください。

3 cube コアの RTL シミュレーション

シミュレーション内容の詳細は「Section 2：CUBE チップの仕様書」をご参照ください。

```
cd cube/sim

vi rom.pl（$mode = 0 にする）
./rom.pl >! event.hex
make sim

vi rom.pl（$mode = 1 にする）
./rom.pl >! event.hex
make sim

vi rom.pl（$mode = 2 にする）
./rom.pl >! event.hex
make sim
```

```
vi rom.pl（$mode = 3 にする）
./rom.pl >! event.hex
make sim
```

test.v の後ろのほうにある「Dump file」のコメントを外すと、test.vcd に値の変化がダンプされます。Simvision や GTKwave などの波形ビューワで test.vcd を開けば波形が見れます。

4 cube コアの論理合成

[要修正] cube/syn/scripts/compile_dc.tcl の LIB_DIR に、スタセルの db ファイルのパスをセットする。

```
cd cube/syn
make syn
```

syn.log に Design Compiler のログ、cube.rep に合成レポートが出ます。

5 cube コアの合成後シミュレーション

[要修正] cube/sim/Makefile の LIBV に、スタセルの Verilog モデルのディレクトリへのパスをセットする。

```
cd cube/sim

vi rom.pl（$mode = 0 にする）
./rom.pl >! event.hex
make ssim

vi rom.pl（$mode = 1 にする）
./rom.pl >! event.hex
make ssim

vi rom.pl（$mode = 2 にする）
./rom.pl >! event.hex
make ssim

vi rom.pl（$mode = 3 にする）
./rom.pl >! event.hex
make ssim
```

6 cube コアの配置配線

[要修正] cube/pr/Makefile の setup の部分に、スタセルの Milkyway ライブラリ、各種テクノロジファイルのパスをセットする。

[要修正] cube/pr/scripts/set_sz.tcl の LIB_DIR、IO_LIB_DIR、DW_DIR に、スタセルの db ファイルのパスをセットする。

```
cd cube/pr
make pr
```

make pr すると、テクノロジーファイルにリンクを張り（make setup)、IC Compiler を起動して配置配線を実行します。

7　cube コアの配置配線後シミュレーション

[要修正] cube/sim/scripts/convert_sdf.tcl の LIB_DIR に、スタセルの db ファイルのパスをセットする。

まず、ICC が生成した遅延（SDF）ファイルを、NC Verilog で読めるかたちに変換します。Synopsys PrimeTime を使用。

```
cd cube/sim
make sdf
```

遅延ファイル（cube_pt.sdf）を読み込ませて、シミュレーションを実行。

```
vi rom.pl（$mode = 0 にする）
./rom.pl >! event.hex
make psim

vi rom.pl（$mode = 1 にする）
./rom.pl >! event.hex
make psim

vi rom.pl（$mode = 2 にする）
./rom.pl >! event.hex
make psim

vi rom.pl（$mode = 3 にする）
./rom.pl >! event.hex
make psim
```

test.v の「Dump file」を有効にして波形ファイルを出す場合、波形ファイルが巨大になるので注意。

8　cube コアの等価性検証

[要修正] cube/verify/scripts_misc/verify_fm.tcl の LIB_DIR に、スタセルの db ファイルのパスをセットする。

cube コアの配置配線を済ませた状態で、

```
cd cube/verify
make verify
```

fm.log をみて、「Failing (not equivalent)」が全部 0 なら OK。

9　cube コアの DRC、LVS、ERC、ANT 検証

[要修正] cube/verify/Makefile の setup の部分に、スタセルの gds、各種テクノロジファイルのパスをセットする。
[要修正] cube/verify/Makefile の cdl の部分に、スタセルの cdl ファイルのパスをセットする。

cube コアの配置配線を済ませた状態で、

```
cd cube/verify
make setup
make gds
make cdl
make ed
```

make setup で、Fujitsu 65nm の検証用スクリプトにリンクを張る。make gds で、配置配線によって得られた cube の GDS ファイルにスタンダードセルをはめ込む。検証では、はめ込み済みの GDS を使う。make cdl、make ed で、LVS で使用する SPICE ネットリスト、ピン位置ファイル（ED TEXT）を生成する。

Design rule check（DRC）をやるには、

```
./cal_drccs2001
```
質問に答えて、
```
./cube_drc_run.csh
```

Layout versus schematic（LVS）と Electric rule check（ERC）をやるには、

```
./cal_lvscs2001
```
質問に答えて、
```
./cube_lvs_run.csh
```

Anntena rule check（ANT）をやるには、

```
./cal_antcs2001
```
質問に答えて、
```
./cube_ant_run.csh
```

.rsf.setup_ant、.rsf.setup_lvs、.rsf.setup_drc があるので、各種質問にはデフォルトのままで大丈夫。

10 ダミーインダクタコア（ptp、sb4）の論理合成、配置配線

[要修正] ptp/syn/scripts/compile_dc.tcl の LIB_DIR に、スタセルの db ファイルのパスをセットする。
[要修正] ptp/pr/Makefile の setup の部分に、スタセルの Milkyway ライブラリ、各種テクノロジファイルのパスをセットする。
[要修正] ptp/pr/scripts/set_sz.tcl の LIB_DIR、IO_LIB_DIR、DW_DIR に、スタセルの db ファイルのパスをセットする。
[要修正] sb4/syn/scripts/compile_dc.tcl の LIB_DIR に、スタセルの db ファイルのパスをセットする。
[要修正] sb4/pr/Makefile の setup の部分に、スタセルの Milkyway ライブラリ、各種テクノロジファイルのパスをセットする。
[要修正] sb4/pr/scripts/set_sz.tcl の LIB_DIR、IO_LIB_DIR、DW_DIR に、スタセルの db ファイルのパスをセットする。

ptp コアを合成する。

```
cd ptp/syn
make syn
```

ptp コアを配置配線する。

```
cd ptp/pr
make pr
```

sb4 コアを合成する。

```
cd sb4/syn
make syn
```

sb4 コアを配置配線する。

```
cd sb4/pr
make pr
```

11　トップ階層（CUBE_TOP）の配置配線

[要修正] CUBE_TOP/pr/Makefile の setup の部分に、スタセルの Milkyway ライブラリ、各種テクノロジファイルのパスをセットする。

[要修正] CUBE_TOP/pr/scripts/set_sz.tcl の LIB_DIR、IO_LIB_DIR、DW_DIR に、スタセルの db ファイルのパスをセットする。

レイアウトに関しては、CUBE_TOP/pr/scripts/CUBE_TOP.tcl および「Appendix A：CUBE_TOP のレイアウトの流れ」をご参照ください。

本チップでは、VDD、VDDBC、VDDBR の 3 電源設計となります。何も考えずに、VDD、VDDBC、VDDBR の IO パッドを配置するとショートするので VDDBC と VDDBR の IO パッドの左右にディバイダセルを入れて、VDD と VDDBC、および、VDD と VDDBR の間でショートが発生するのを防ぎます。詳細は、CUBE_TOP/pr/scripts/ioplace.tcl をご参照ください。

```
cd CUBE_TOP/pr
make setup
```

実は、CUBE_TOP の配置配線で DRC エラーが出てしまうので、最初から最後まで自動実行（make pr）することはできません。icc_shell -gui を立ち上げて、

```
icc_shell -gui
```

CUBE_TOP/pr/scripts/CUBE_TOP.tcl を、「STOP HERE. PLEASE RUN DRC.」まで 1 コマンドずつ実行してください。コマンドは、MainWindow の「icc_shell>」のところにコピーアンドペーストすれば良いです。

「STOP HERE. PLEASE RUN DRC.」まで実行したら、LayoutWindow の Verification → DRC を実行する。

DRC エラーのうち「MaxWdith 1」だけは手動で取る必要があります。エラーの取り方は「Appendix B：CUBE_TOP の DRC エラー修正方法」をご参照ください。

修正が終わったら、もう一度、DRC を実行して「MaxWdith 1」が消えていることを確認する。大丈夫そうなら「STOP HERE. PLEASE RUN DRC.」以降のコマンドを実行する。

12　フレームはめ込み

[要修正] CUBE_TOP/frame/Makefile の setup の部分に、各種テクノロジファイルのパスをセットする。

[要修正] CUBE_TOP/frame/Makefile の stdcell の部分に、スタセルの GDS（cs202sz_uc.gds）のパスをセットする。同様に、cs202_fm.gds、frames.str、MB8AW4203_FRAME.gds、cs202_io.gds のパスもセットする。

MB8AW4203_FRAME.gds はチップ試作番号に応じて VDEC から送られてきますので、それを使ってください。

```
cd CUBE_TOP/frame
make setup
```

これによって、cube コア、ダミーコンダクタコア（ptp、sb4）、CUBE_TOP の GDS へのシンボリックリンクが CUBE_TOP/frame/input/ 以下にできる。

スタセル、フレーム、cube コア、ダミーインダクタコア（ptp、sb4）、CUBE_TOP の GDS を読み込む。

```
make base
```

フレームに CUBE_TOP をはめ込む。「Appendix C：CUBE チップのフレームはめ込み」を見るとイメージがつかめると思います。

```
make CUBE_TOP.addframe
```

フレームはめ込み済みの GDS が CUBE_TOP/frame/CUBE_TOP.gds にできる。

13　チップ全体の DRC、LVS、ERC、ANT 検証

[要修正] CUBE_TOP/verify/Makefile の setup の部分に、各種テクノロジファイルのパスをセットする。
[要修正] CUBE_TOP/verify/Makefile の CUBE_TOP.cdl の部分に、スタセルの CDL ファイルのパスをセットする。

```
cd CUBE_TOP/verify
make setup
```

これによって、Fujitsu 65nm の検証用スクリプトにリンクが張られる。また、cube コア、ダミーコンダクタコア（ptp、sb4)、CUBE_TOP の LVS 用 Verilog ネットリストへのシンボリックリンクが CUBE_TOP/frame/input/ 以下にできる。なお、IC Compiler で、チップレベルの LVS 用に write_verilog するときは、「-no_physical_only_cells」オプションを付けてはいけない。

LVS 用 Verilog ネットリストから、LVS 検証用 SPICE ネットリストを作る。テキスト処理に ruby を使ってるので ruby が使える環境で実行してください。

```
make cube.cdl
make sb4.cdl
make ptp.cdl
make CUBE_TOP.cdl
```

LVS 用 SPICE ネットリストが CUBE_TOP/frame/cdl 以下にできる。

VDD や VSS を global 宣言したところ、LVS の際に、VDD と VDDBC/VDDBR が誤認識されたため、global 宣言しないようにしている。また、IO セルやコーナーセルから、VNW=VNW、VPW=VPW を取り除く。VDEC のシステム上で LVS をかけるときは、include している cs202sz_uc.cdl や cs202_io.cdl も含めて１つのファイルにする。詳細は、CUBE_TOP/verify/Makefile をご確認ください。

Design rule check（DRC）をやるには、

```
./cal_drccs2001
```
質問に答えて、
```
./CUBE_TOP_drc_run.csh
```

Layout versus schematic（LVS）と Electric rule check（ERC）をやるには、

```
./cal_lvscs2001
```
質問に答えて、
```
./CUBE_TOP_lvs_run.csh
```

Anntena rule check（ANT）をやるには、

```
./cal_antcs2001
```
質問に答えて、
```
./CUBE_TOP_ant_run.csh
```

.rsf.setup_ant、.rsf.setup_lvs、.rsf.setup_drc があるので、各種質問にはデフォルトのままで大丈夫。

Section 2:

CUBE チップの仕様書

CUBE チップの仕様書

松谷 *

平成 22 年 10 月 20 日

もくじ

1　チップの I/O、制御レジスタ

自由に使える I/O ピンは、clk および rst_ を含めて１２本だけ。１２本を以下のように使う。リセット信号は active-low である。

```
IN  クロック（clk）
IN  リセット（rst_） active-low
IN  入力データ４本（idata）
OUT 出力データ３本（odata）
IN  制御レジスタバンク指定２本（sel）
IN  制御レジスタ書き込み１本（wr）
```

設定の書き込みは、sel 信号で制御レジスタのバンクを切り替えて、wr をアサートして、idata で指定した値をセットする。データの読み込みは、sel 信号で指定した制御レジスタの中身が odata に出る。

表１：制御レジスタのフォーマット。X はがんばれば省略可能なビット。

```
=====================================================================
制御レジスタ０（sel == 0）:
  idata[0]     コアの選択（コア０かコア１）
  idata[1]     設定済みの dest と vch でパケットを発射する（inject）
  idata[2]     ダミーモード（絶対に１にしないでください）
X idata[3]     選択中コアのパケットカウンタをクリアする
  odata[0]     選択中コアの VC0 にパケット注入してよいか？
  odata[1]     選択中コアの VC1 にパケット注入してよいか？
X odata[2]     選択中のコア（コア０かコア１）
---------------------------------------------------------------------
制御レジスタ１（sel == 1）:
  idata[3:0]    パケットの宛先アドレス dest（４ビット）
```

*matutani@hal.ipc.i.u-tokyo.ac.jp

```
X odata[2:0]    パケットカウンタの値（０～３ビット）
-----------------------------------------------------------------
制御レジスタ２（sel == 2)：
  idata[0]      パケットが使う仮想チャネル番号 vch（１ビット）
  idata[1]      バーストモード ON
  idata[2]      NoC モードの切り替え（０：Point-to-point、１：Sbus）
  idata[3]       選択中コアのパケットカウンタの値を読み出す（cread）
  odata[2:0]    パケットカウンタの値（４５ビット、シリアル出力）
-----------------------------------------------------------------
制御レジスタ３（sel == 3)：
  idata[2:0]    チップ番号（０～７）
  idata[3]      自チップがトップか？
X odata[2:0]    パケットカウンタの値（４～７ビット）
-----------------------------------------------------------------
```

cread がアサートとされると、次のサイクルから１５サイクルかけてパケットカウンタの値が３ビットずつ出力される。

バーストモード ON の状態で、inject がアサートされると、バーストモードが OFF になるまで dest と vch で発射し続けます。

チップ初期化（チップ番号設定)、パケット送信、受信フリット数の集計シーケンスの実例は、sim/test.v を参照してください。

2　全体アーキテクチャ、トポロジ、ルーティング

１チップに２コアを実装する。コアごとにルータが１個付く。

このチップを最大８枚（１６コア）積層できるようにする。ただし、シミュレータ（sim/test.v）ではとりあえず４枚（８コア）にしてある。

Chip-3 がトップで、Chip-0 がボトムとする。各チップにおいて、左が偶数コア（ルータ)、右が奇数コア（ルータ）とする。

```
Chip-3 [Router_6] ←→ [Router_7]
           ↓              ↑
Chip-2 [Router_4] ←→ [Router_5]
           ↓              ↑
Chip-1 [Router_2] ←→ [Router_3]
           ↓              ↑
Chip-0 [Router_0] ←→ [Router_1]
```

同一チップ上の２コアは、水平、双方向リンクで接続される。偶数ルータは下向きリンクを持ち、奇数ルータは上向きリンクを持つ。

仮想チャネル０を使うパケットは、水平リンクを使ってはいけない（ただし、ボトムの右向き、トップの左向きリンクを除く)。ルータ１からルータ２へのパケットは、ルータ１→ルータ３→ルータ５→ルータ７→ルータ６→ルータ４→ルータ２を流れる。

仮想チャネル１を使うパケットは、仮想チャネル０の制約に加え、宛先が同一チップ上の反対側にあるときだけ、水平リンクを使う。ルータ１からルータ２へのパケットは、ルータ１→ルータ３→ルータ２を流れる。

3　パケットフォーマット、フリットフォーマット

１パケットは、１個のヘッダフリット、３個のデータフリット、１個のテイルフリットから構成される。

ヘッダには宛先アドレスが格納される。データとテイルは実データを運ぶが、テイルがルータ上を通過したら、クロスバを開放する。

表２：フリットのフォーマット。X は本来あってはいけないフィールド。

```
=====================================================================
ヘッダフリット：
  flit[33:32]   フリットタイプ。ヘッダは 2'b01
  flit[31:8]    M 系列によるランダム値
X flit[7:4]     パケット内のフリット番号。ヘッダなので常に０
  flit[3:0]     宛先コアアドレス
---------------------------------------------------------------------
データフリット：
  flit[33:32]   フリットタイプ。データは 2'b10
  flit[31:8]    M 系列によるランダム値
X flit[7:4]     パケット内のフリット番号。１～３のどれか
X flit[3:0]     宛先コアアドレス
---------------------------------------------------------------------
テイルフリット：
  flit[33:32]   フリットタイプ。テイルは 2'b11
  flit[31:8]    M 系列によるランダム値
X flit[7:4]     パケット内のフリット番号。テイルなので常に４
X flit[3:0]     宛先コアアドレス
---------------------------------------------------------------------
```

sim/test.v を実行すると、パケットが流れる様子が分かると思います。

4　イベント ROM フォーマット

sim/test.v では、イベント ROM（sim/event.hex）にしたがって、パケットを生成、送信する。

表３：イベント ROM のフォーマット。１６進数で書く。

```
=====================================================================
event[43:12]    パケット送信クロック
event[11:8]     使用する仮想チャネル（VC0 もしくは VC1）
event[7:4]      送信コア（０～７）
event[3:0]      宛先コア（０～７）
---------------------------------------------------------------------
```

パケット送信クロックになったら、パケットを生成（コアの制御レジスタを設定する）し始めるので、実際に送信するのはその数サイクル後になる。

イベント ROM の例を以下に示す。１行目には「NoC モード」、２行目には「総イベント数」を入れる。

```
0                   // NoC モード（０～３。詳しくは flow.txt を参照）
3 // イベント数は３個
00000010_0_3_2 // １６サイクル目に、VC0 を使って、コア３→コア２
00000020_1_7_6 // ３２サイクル目に、VC1 を使って、コア７→コア６
00000030_1_2_3 // ４８サイクル目に、VC1 を使って、コア２→コア３
```

5　プローブ信号

上記の１２本の I/O ピンに加え、最上位のチップから内部を観測するためにプローブ信号９本（probe）を持たせる。

表４：プローブ信号（９本）の割り当て。

probe[1:0]	コア０のチップ ID（id_0[2:1]）
probe[2]	コア０のタイムスロット（ts_control）
probe[4:3]	コア０ --> ルータ０間のフリットタイプ
probe[6:5]	ルータ０ --> ルータ１間のフリットタイプ
probe[8:7]	ルータ１ --> コア１間のフリットタイプ

以下の検証が可能である。

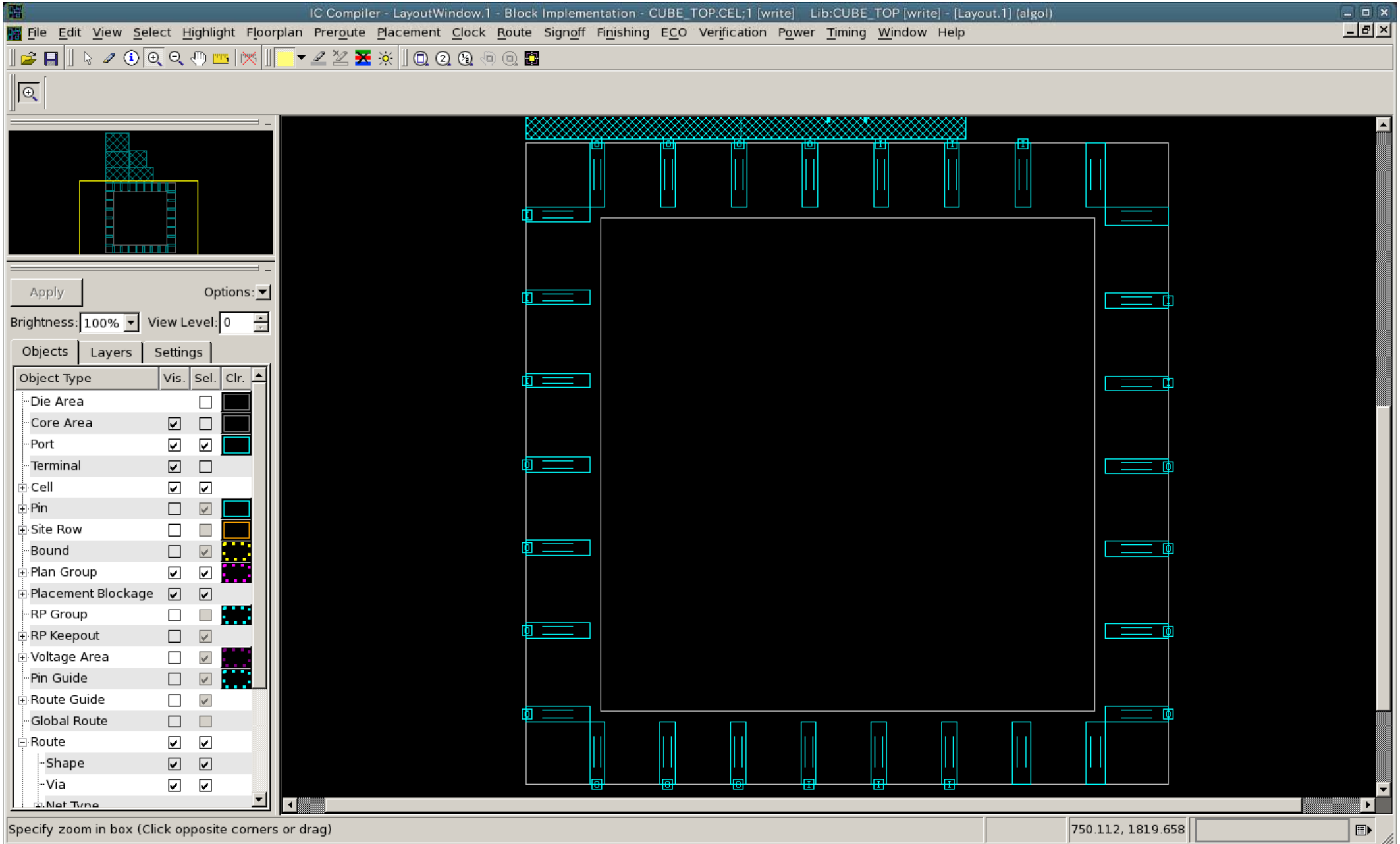
- probe[1:0] で、制御レジスタに値を書き込めることを確認。
- probe[2] で、Shared bus のタイムスロットが動作しているか確認。
- コア０からコア１にパケットを送り、probe[8:3] でパケットが各リンクを流れる様子を確認できる。

Appendix A:

CUBE_TOP のレイアウトの流れ

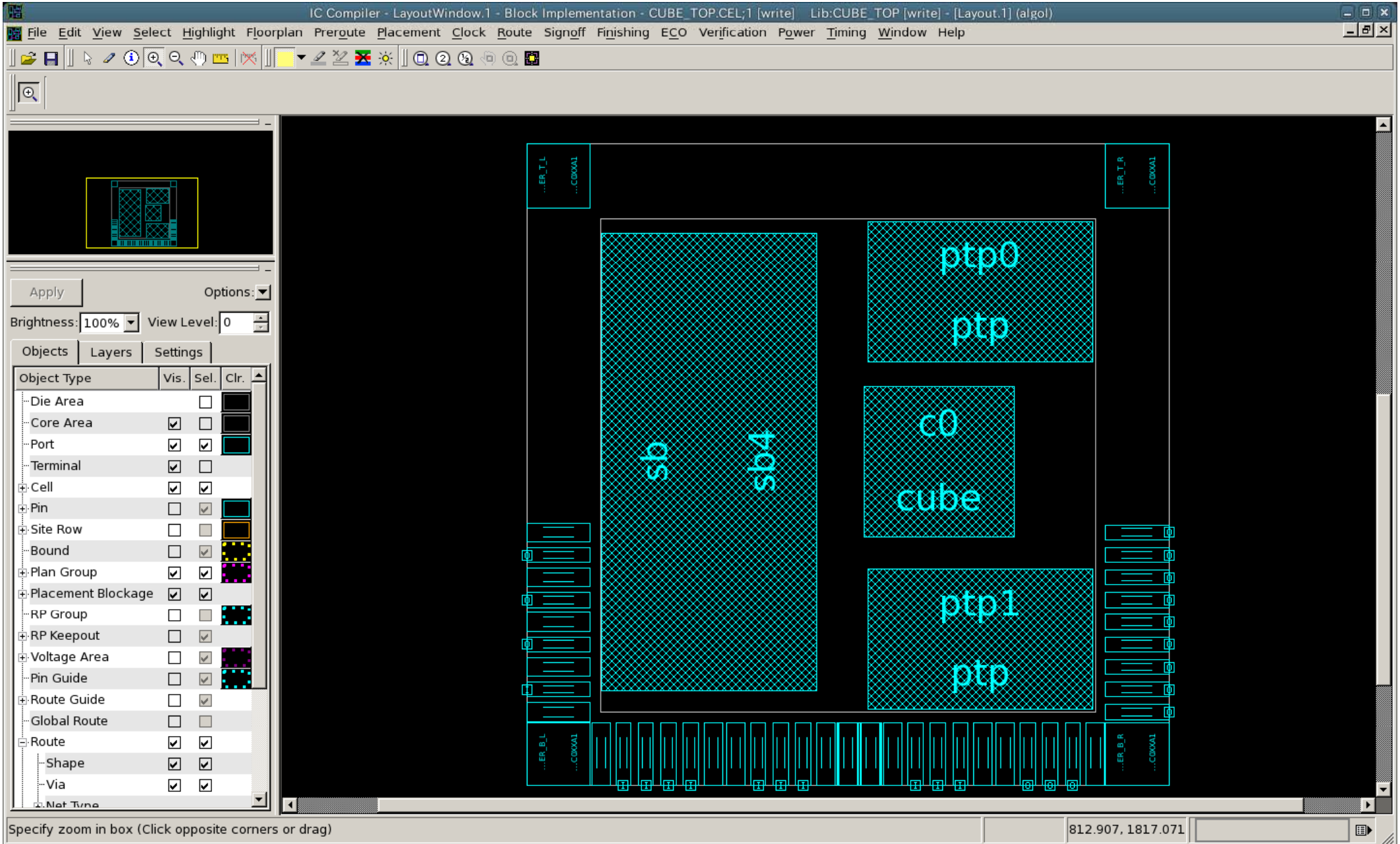
CUBE_TOPのレイアウトの流れ 1/8

initialize_floorplanで、枠 (1320um x 1320um) を決める。



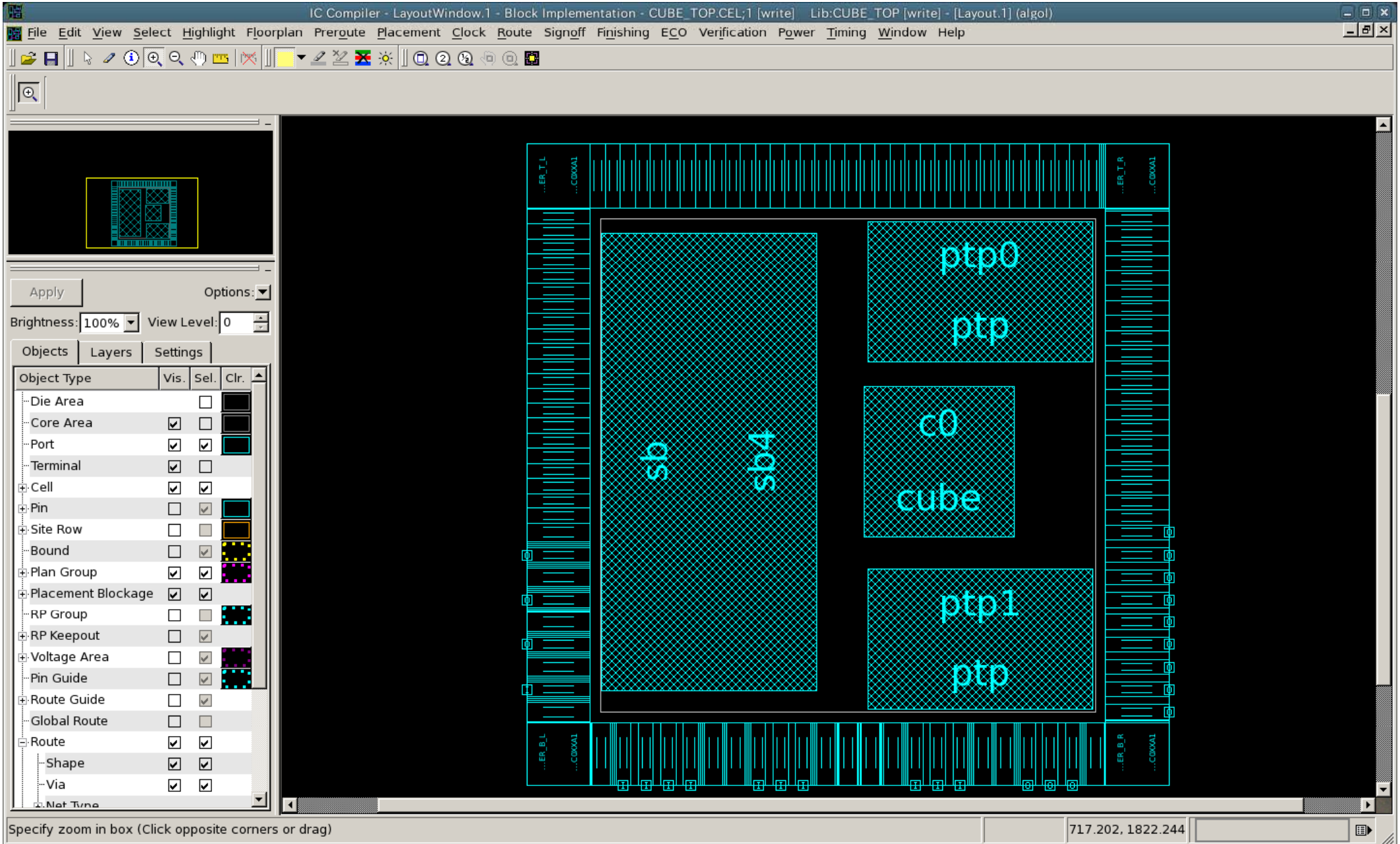
CUBE_TOPのレイアウトの流れ 2/8

iogen.tclとioplace.tclを実行し、マクロ (cube、ptp0、ptp1、sb）やI/Oパッドを配置。



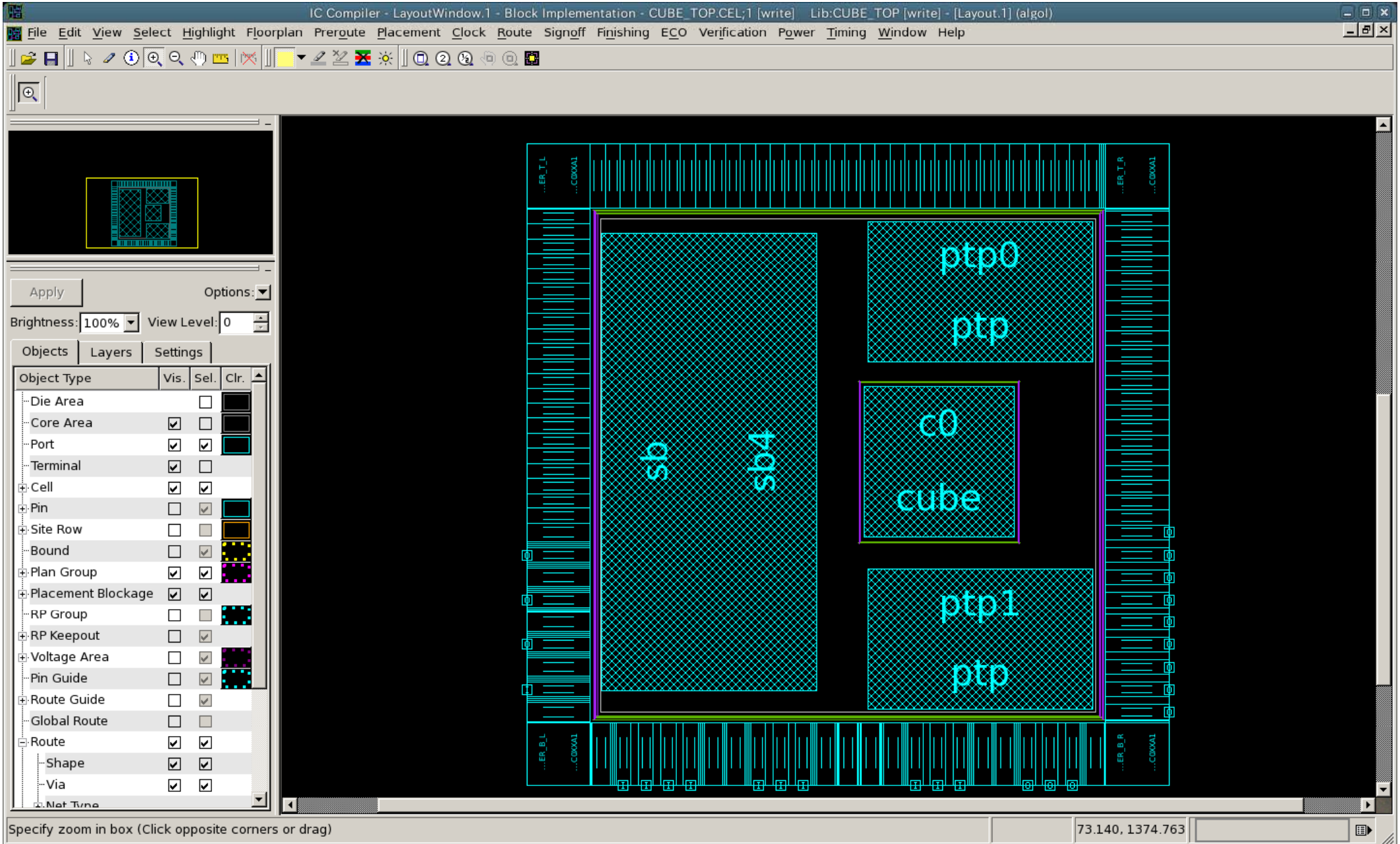
CUBE_TOPのレイアウトの流れ 3/8

insert_pad_filler を実行して、パッドとして使われてない部分にフィラーを詰める。



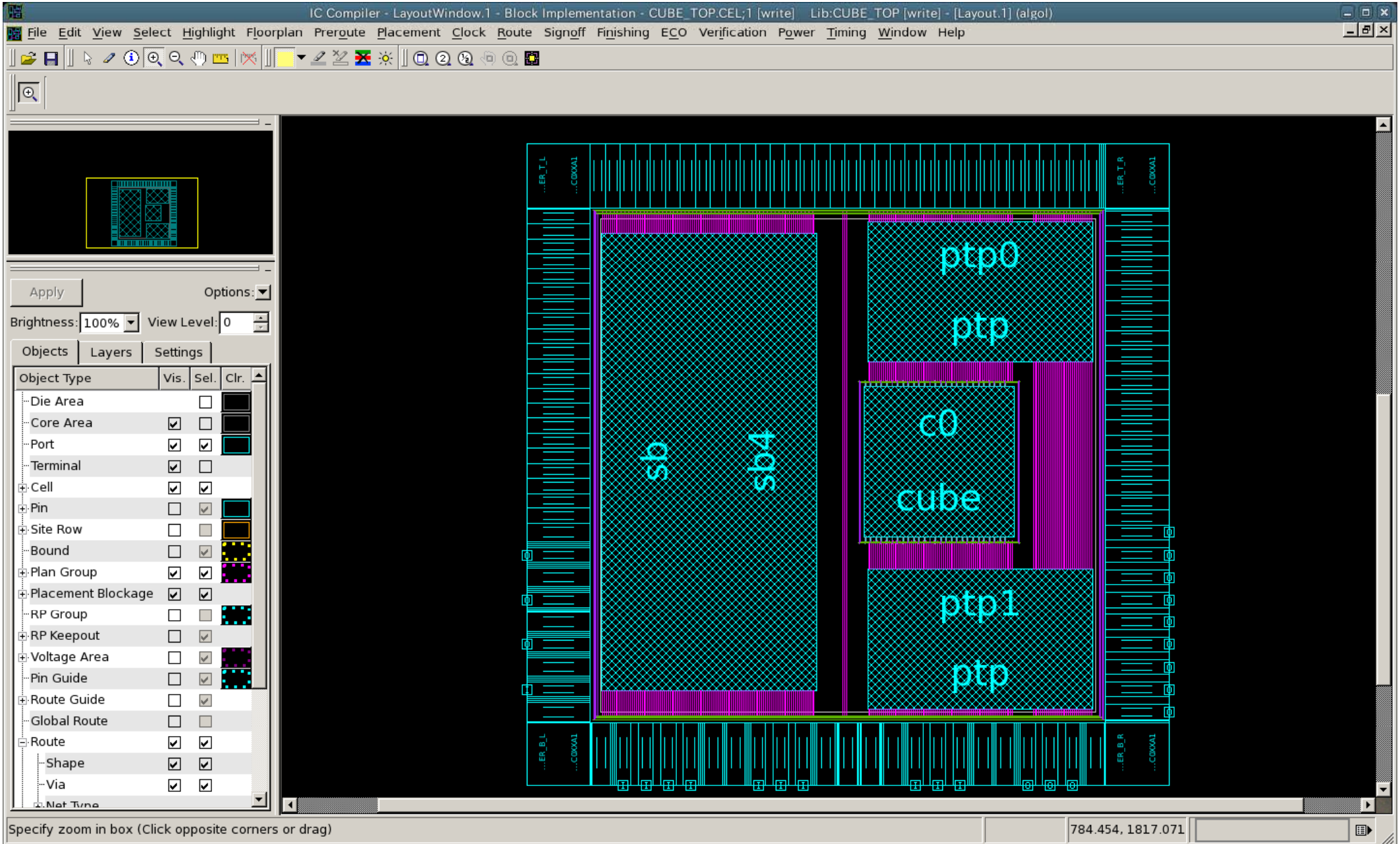
CUBE_TOPのレイアウトの流れ 4/8

create_rectangular_ringsを実行して電源リングを巻く。電源はVDD、VDDBR、VDDBCの3種類なので、VSSを含めて電源リングは4本になる。



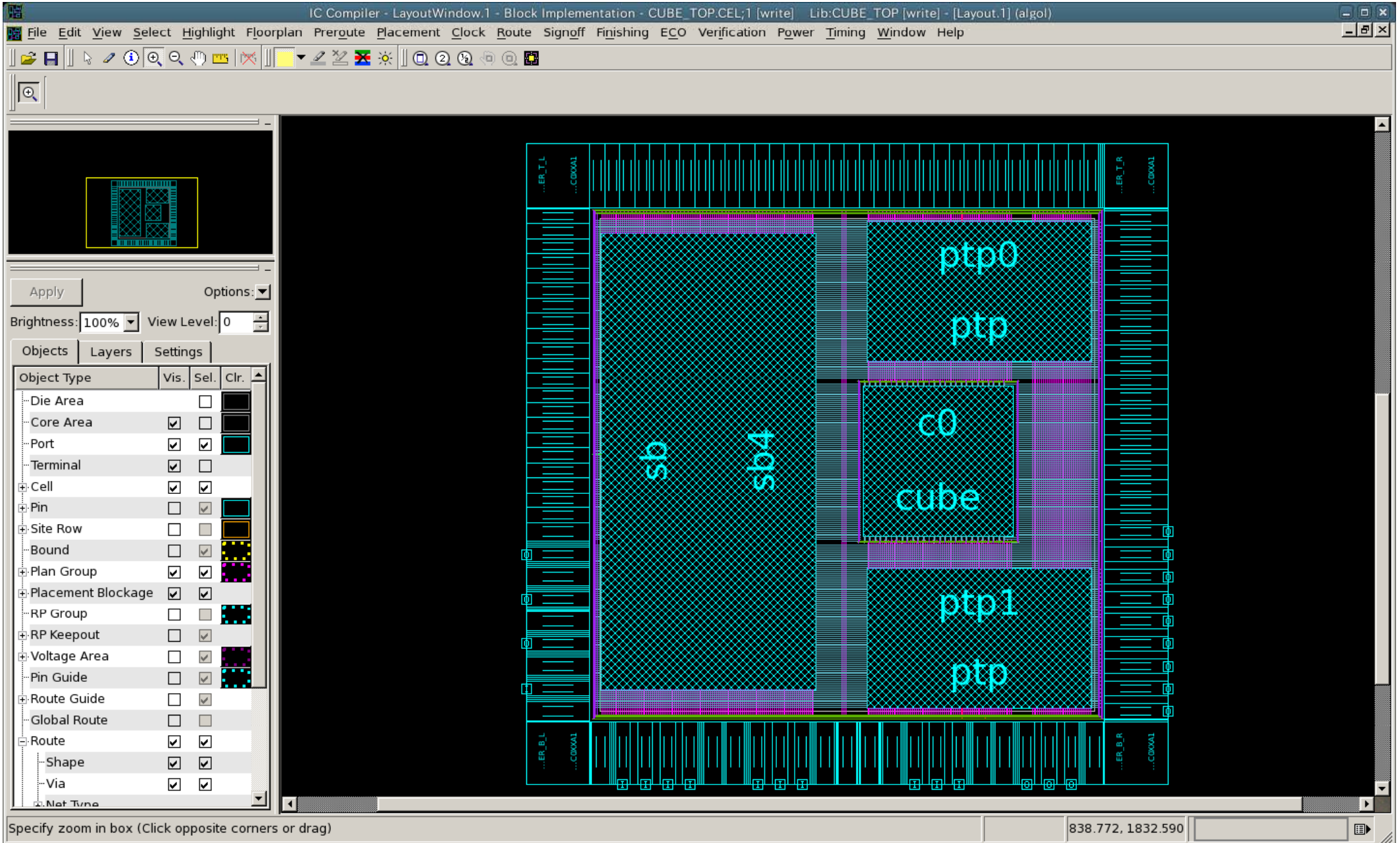
CUBE_TOPのレイアウトの流れ 5/8

create_power_strapsを実行して、垂直方向の電源ストラップを張る。



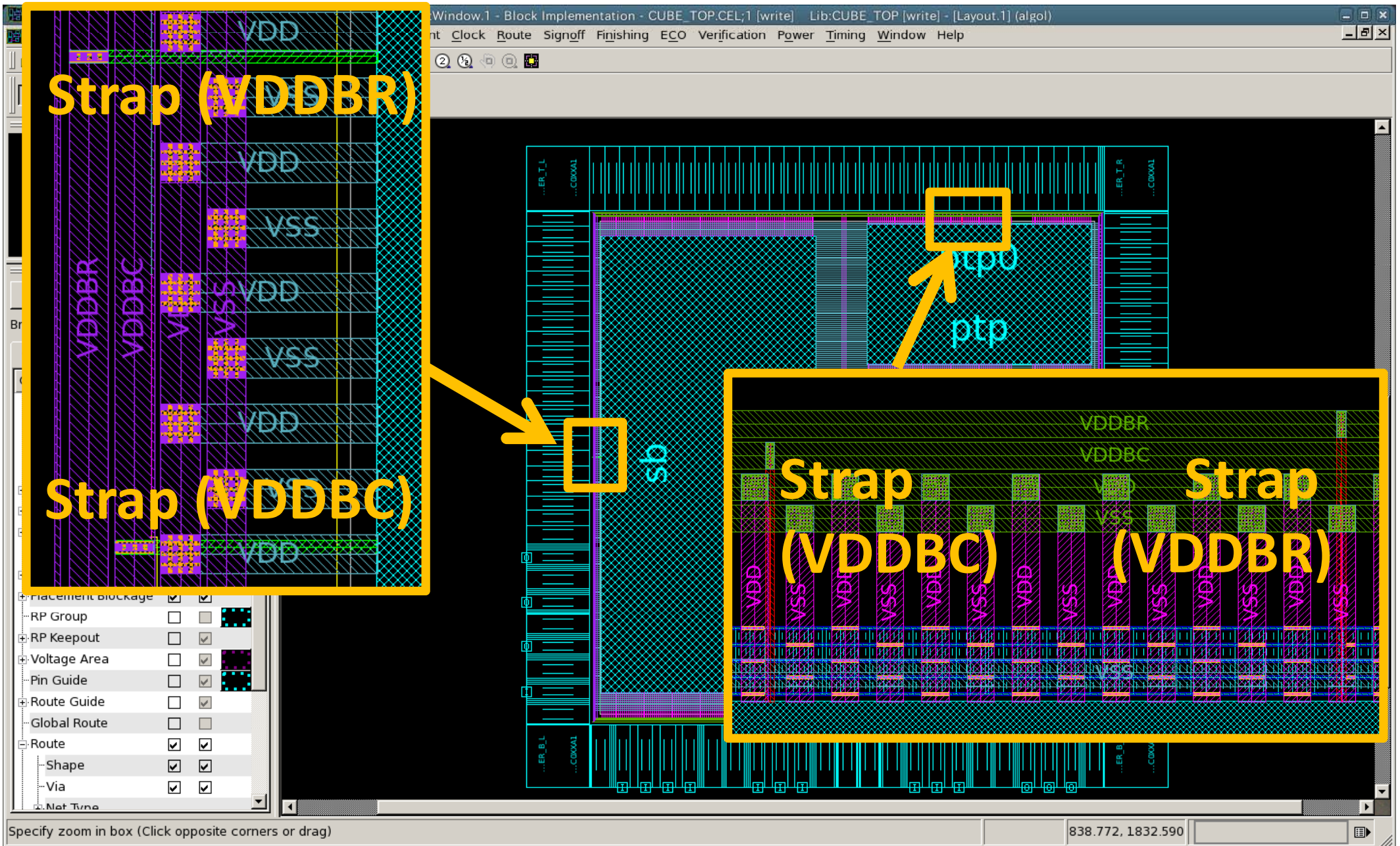
CUBE_TOPのレイアウトの流れ 6/8

今回のインダクタ(ptp0、ptp1、sb)には、水平方向からも電源ストラップを張る。



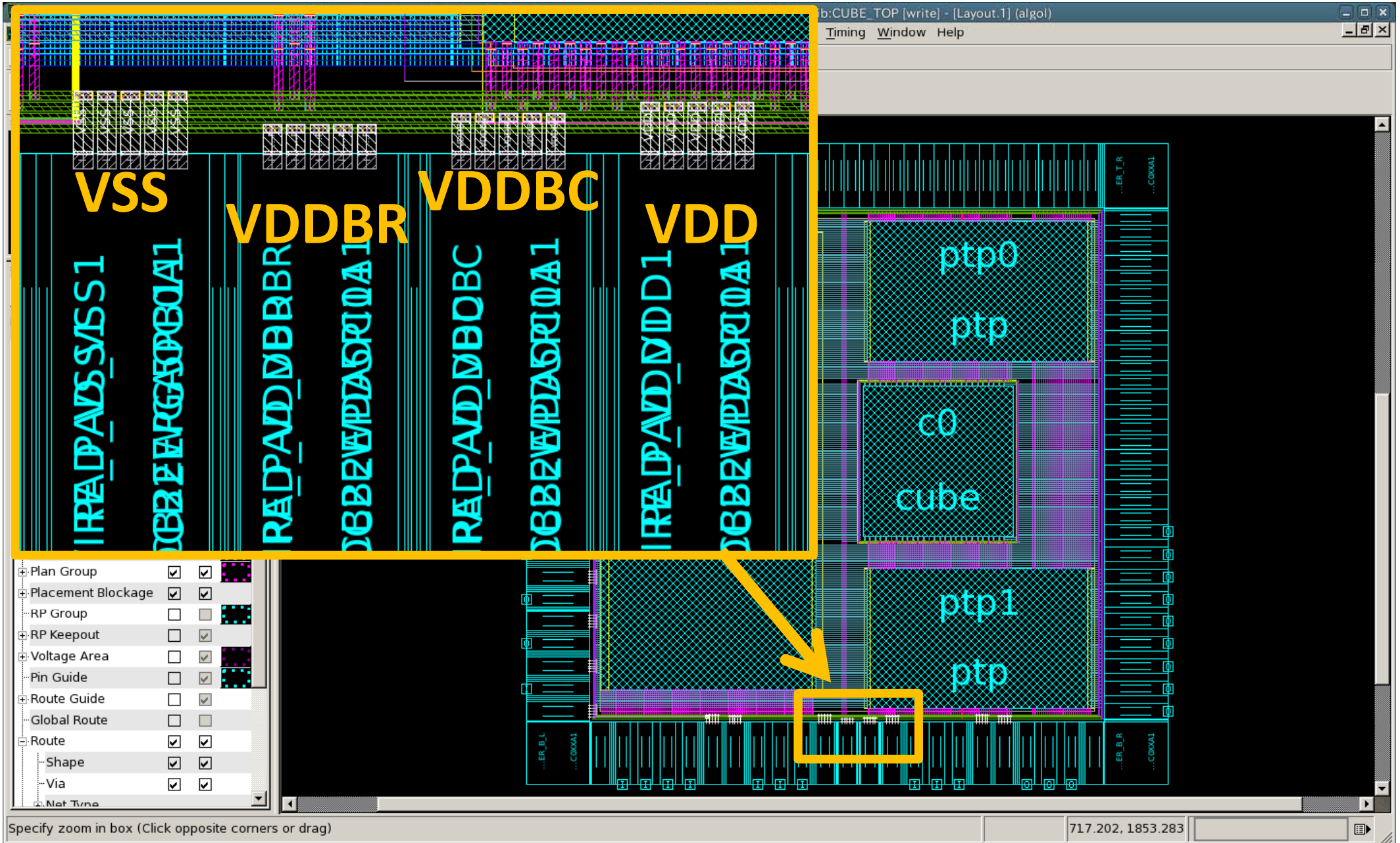
CUBE_TOPのレイアウトの流れ 7/8

VDDBRおよびVDDBCの電源リングから、インダクタ(ptp0、ptp1、sb）のVDDBRおよびVDDBCポートへストラップを張る。



CUBE_TOPのレイアウトの流れ 8/8

preroute_instancesを実行して、電源パッド (VDD、VSS、VDDBC、VDDBR) とそれに対応する電源リングとをつなぐ。



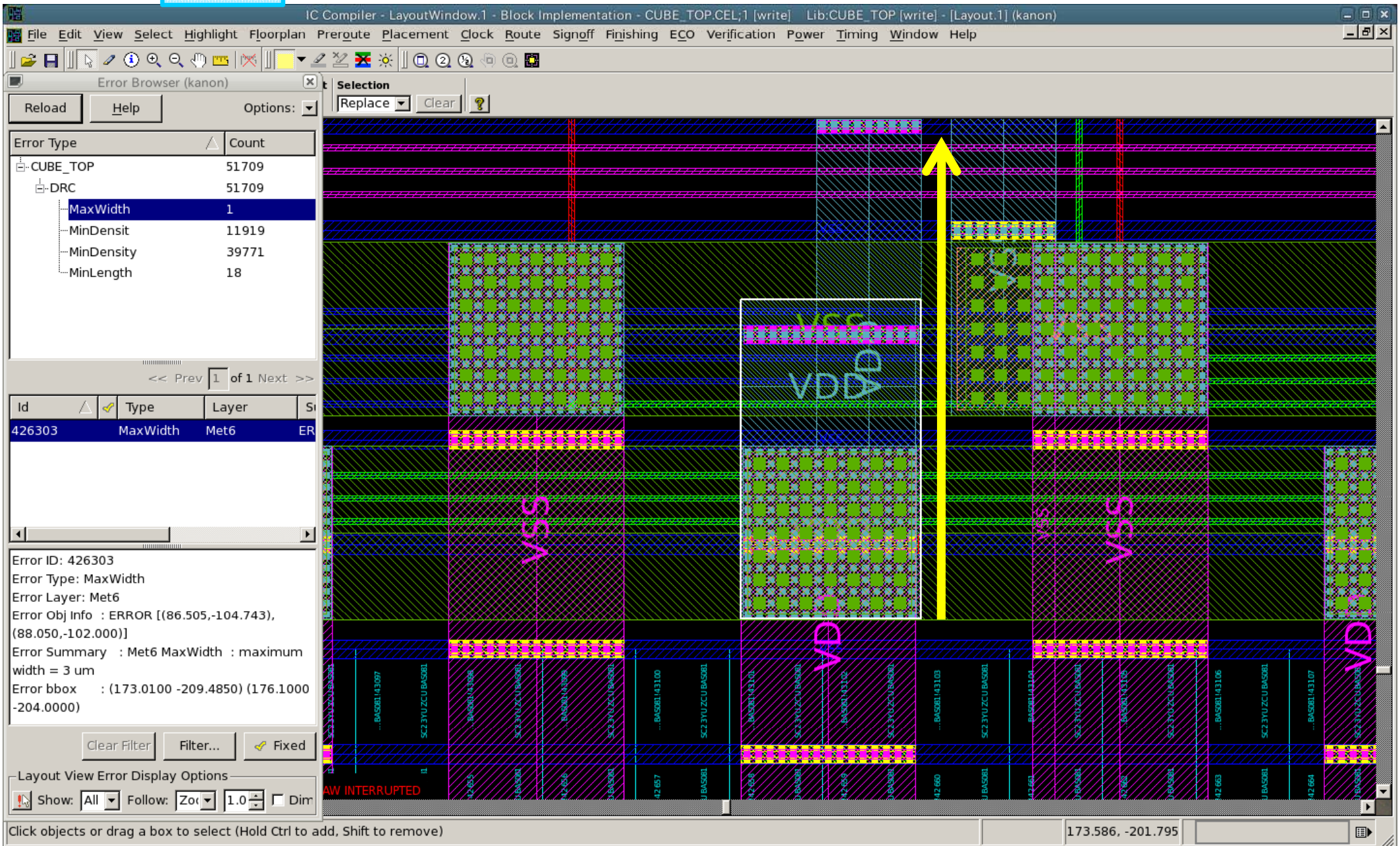
Appendix B:

CUBE_TOP の DRC エラー修正方法

CUBE_TOPのDRC回避方法 1/4

白で囲まれた VDD の部分が横にちょっとだけ広いので、ひっこめる必要あり。

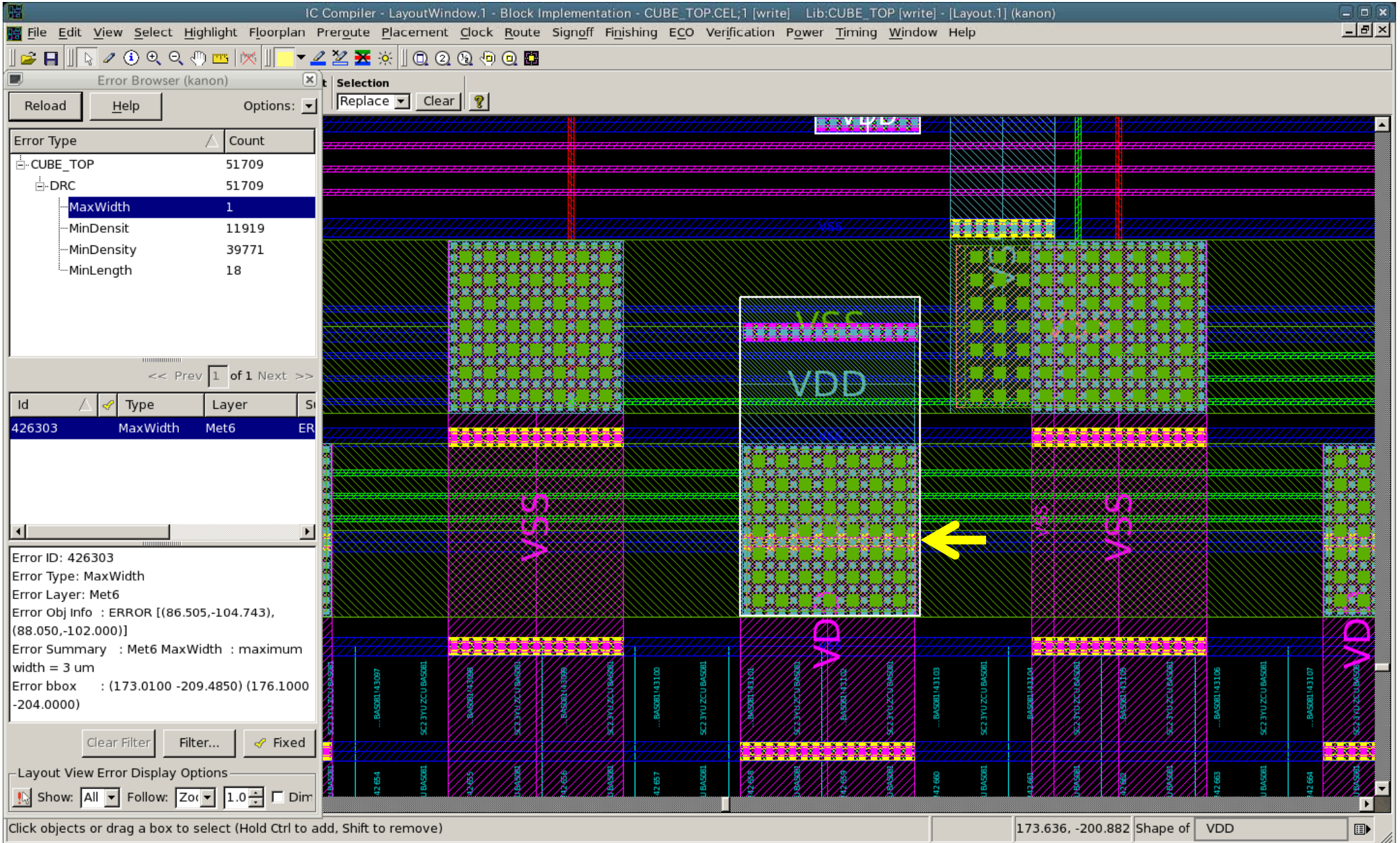
まず、VDD の縦線を上のほうまで引きずり上げる (黄色矢印に従い Resize)。



CUBE_TOPのDRC回避方法 2/4

白で囲まれた VDD の部分が横にちょっとだけ広いので、僅か左にひっこめる。

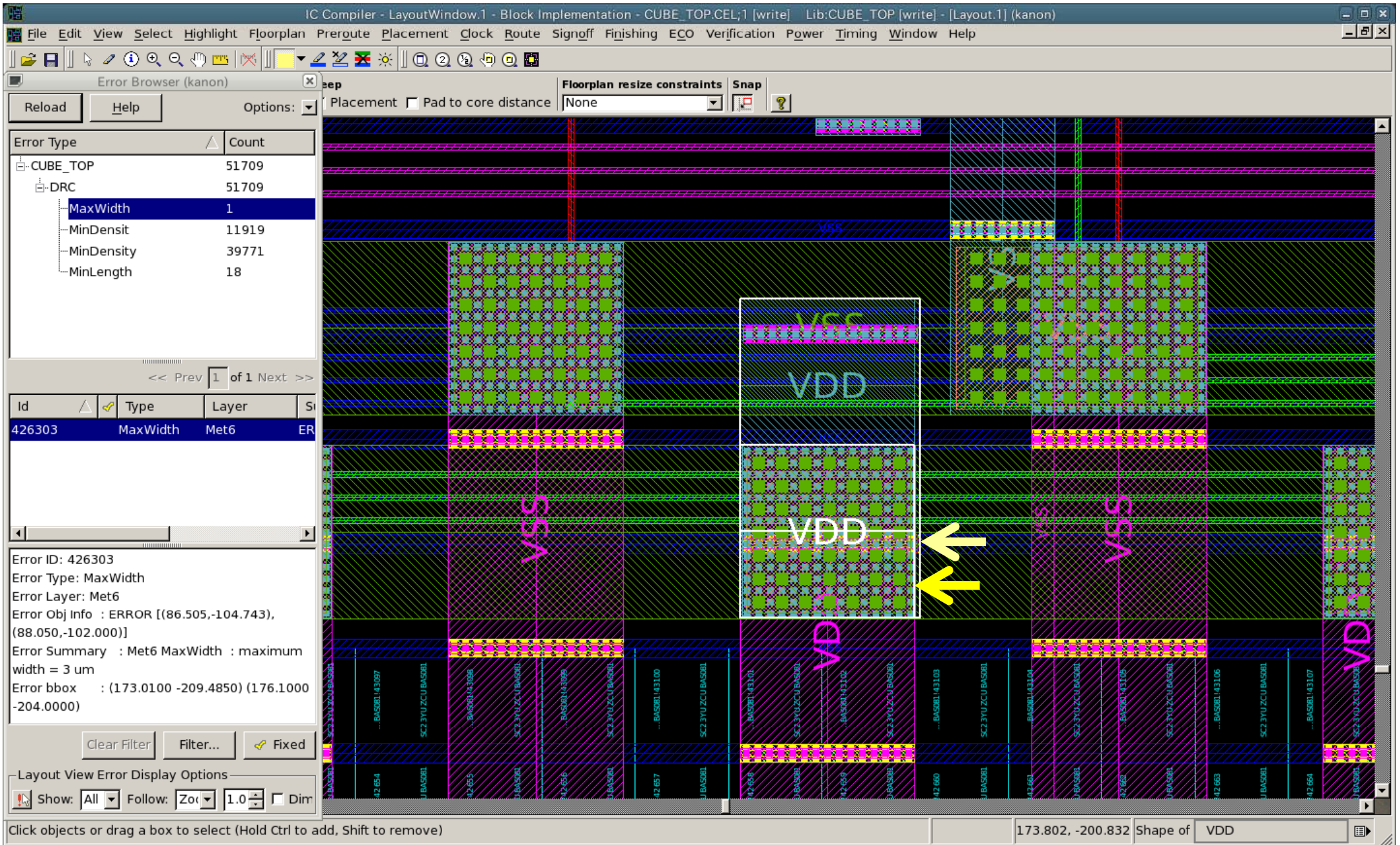
(黄色矢印の位置に注目)



CUBE_TOPのDRC回避方法 3/4

白で囲まれた VDD の部分が、僅か左にひっこんだ!

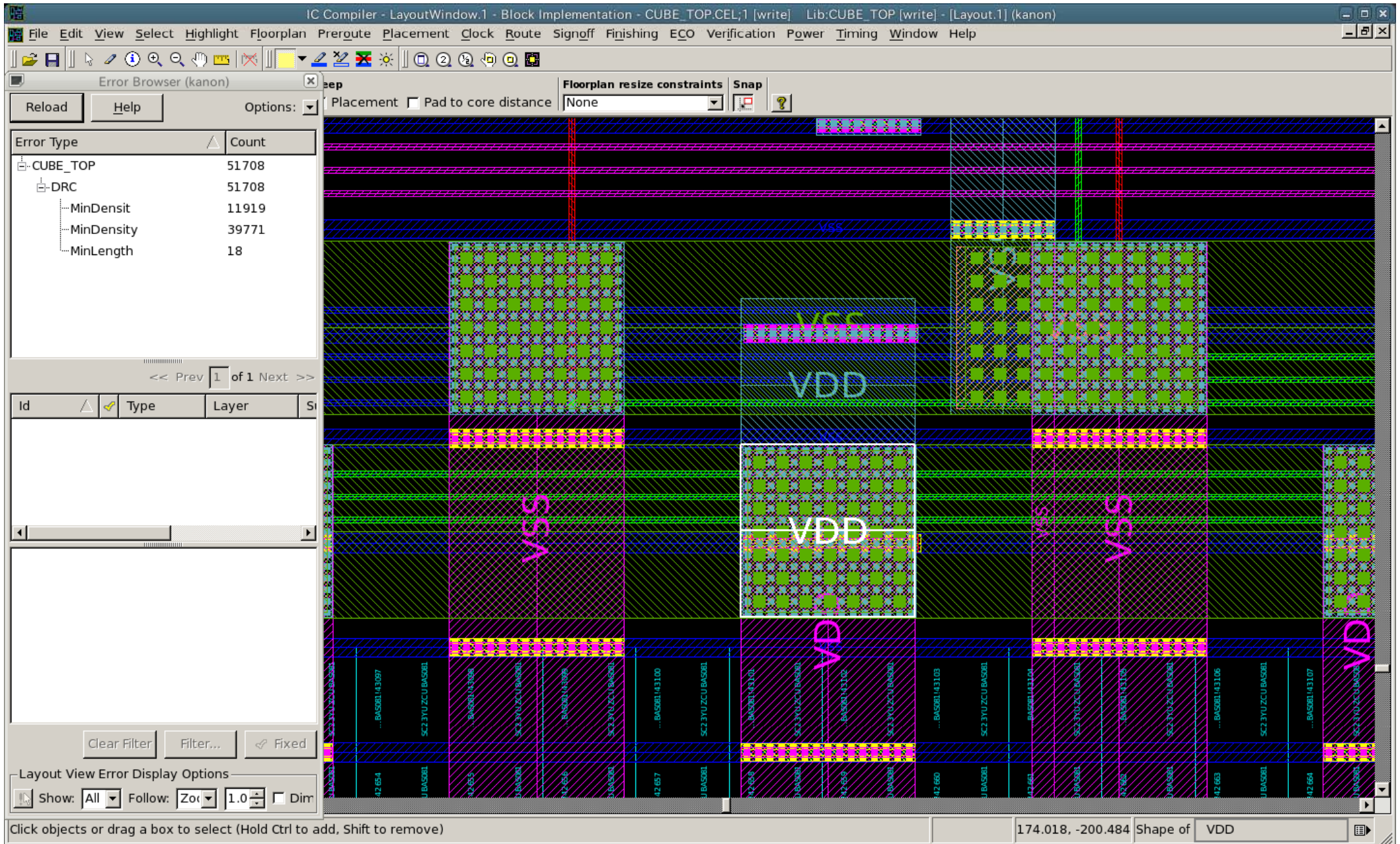
(黄色矢印の位置に注目)



CUBE_TOPのDRC回避方法 4/4

白で囲まれた VDD の部分が、僅か左にひっこんだので、

DRCをかけ直したら、MaxWidth x1 が消えた。他のエラーはここでは無視。



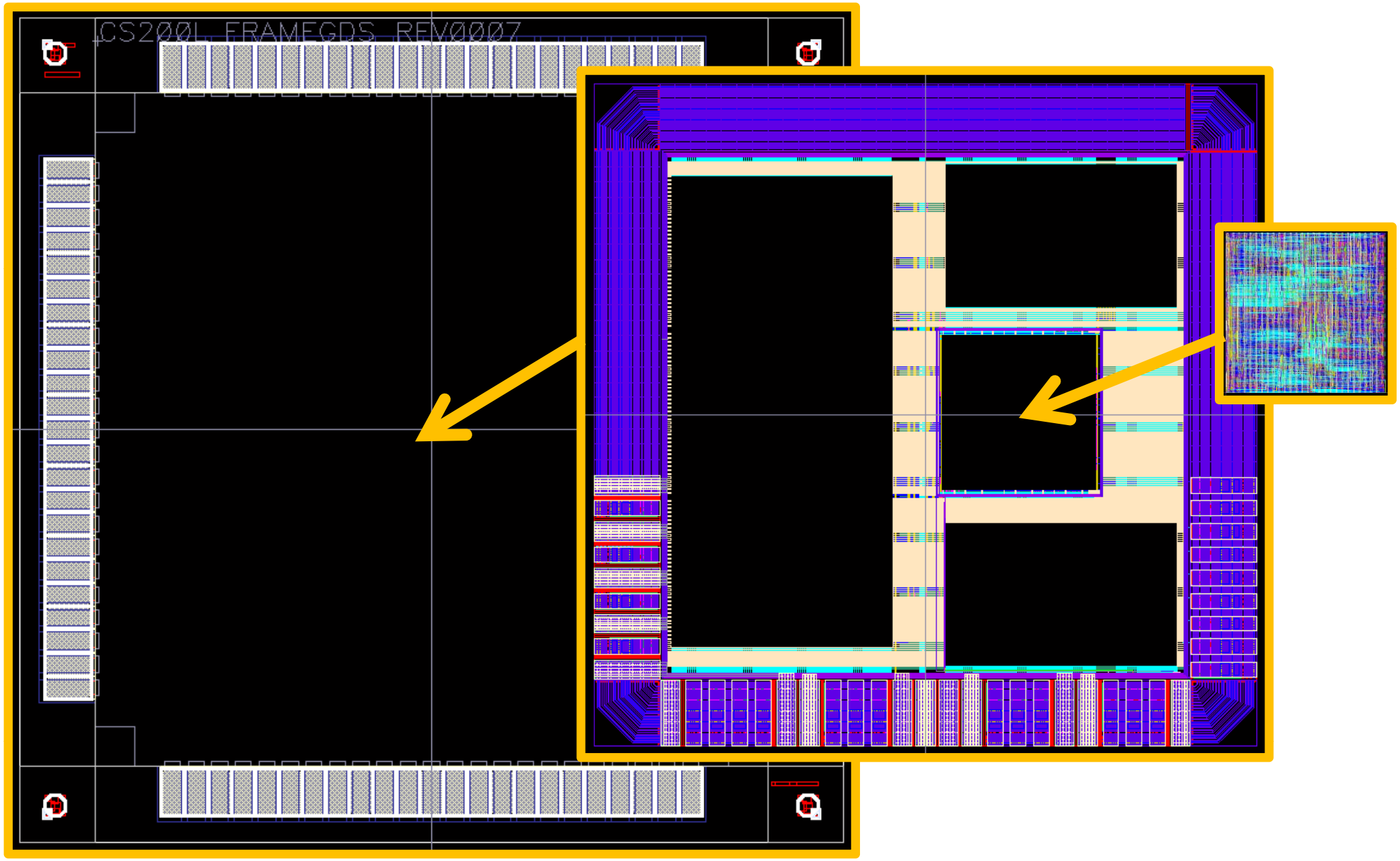
Appendix C:

CUBE チップのフレームはめ込み

CUBEチップのフレームはめ込み 1/2

E5FRAM、CUBE_TOP、cubeコア、ダミーインダクタ (ptp0、ptp1、sb) をはめ込む。

(注意: 実際のテープアウトでは、本物のインダクタを使用)



CUBEチップのフレームはめ込み 2/2

E5FRAM、CUBE_TOP、cubeコア、ダミーインダクタ (ptp0、ptp1、sb) をはめ込む。

(注意: 実際のテープアウトでは、本物のインダクタを使用)