0.1 はじめに

POCOP_TOP は、パイプライン化した POCO のコアのみをレイアウトしたものである。階層化されていて、まず、実体である POCOP をレイアウトし、これを POCOP_TOP のど真ん中に置いて全体をレイアウトする作戦を取っている。本ドキュメントは、この TOP 階層に関するものである。ディレクトリのセットアップ、POCOP マクロの作り方は、Astro、POCOP マクロのデザインフローを参照のこと。 hunga/verilog/f65/pr にレイアウト用のディレクトリがあるのでそこを参照のこと。

POCOP_TOP は、I/O を含む e-shuttle 65nm 半区画のチップ上に POCOP を載せるもので、このままテープアウト可能な設計を目指している。レイアウト講習の多くでは、マクロをレイアウトしただけで終わりにしてしまうが、実際にテープアウトしてみると、その設計時間の大半はマクロの組み上げ、I/O 配置、マクロ接合部のエラー等トップ階層に費やされる。したがって、このステップおよび最後の Calibre 四天王を体験しておくことは、テープアウトのために絶対に必要である。

1 POCOP_TOP の Verilog 記述

トップ階層の Verilog 記述には、I/O と信号の接続を定義する必要がある。ここでは、入力と出力にそれぞれ以下の I/O を用いている。実はこれ以外使ったことがないので、他はどうだか分からないのだが、CMOS 3.3V 入出力の標準的なものである。

```
IOCB2EITNNMXA02 PAD_IDATAIN_00 (
    .EA(IO_IDATAIN[0]              ),
    .X (IDATAIN[0]                 ));



IOCB2EOT2X2LA02 PAD_DDATAOUT_00 (
    .A (DDATAOUT[0]          ),
    .EX(IO_DDATAOUT[0]       ));
```

この場合、IDATAIN, DDATOUT がマクロへの接続、IO_が付く方が PAD つまりチップの外部への接続に相当する。最上位階層は、このまま配線のみなので、今回は sdc は使わない。使う場合は適当に作ってやれば良い。

2 POCOP_TOP.tcl の解説

Astro を立ち上げて、tcl の所をクリックし、source POCOP_TOP.tcl とやると以下が実行され、POCOP_TOP のレイアウトが生成される。これを順に解説する。

2.1 初期設定

以下、利用する変数を定義する。POCOP のサイズはマクロの所で決めたように 102um 四方にする。

チップのサイズを考えてコアの大きさを 3420 X 1320 に設定し、コアと電源リングのスペースを 28 取る。リファレンスライブラリとして、標準セルに合わせて I/O、POCOP を指定する。

```
set design_name POCOP_TOP
set is_combinational_circuit false
```

```
set pocop_width    102.0
set pocop_height   102.0

set core_width     3420.0
set core_height    1320.0
# 27 is space for power ring
set core_to_top    28
set core_to_bottom 28
set core_to_left   28
set core_to_right  28

set ref_libs {"CS202SZ" "CS202IO" "POCOP"}
```

2.2 verilog file の読み込み

最初に作ったトップ階層用の Verilog 記述を読み込む。これはマクロのケースとほぼ同様であるが、VDE を設定する所が違う。読み込み中 0 ／ 1 の変換、テクノロジファイルの指定等を行う。

```
POCOP_TOP_verilog_to_cell.tcl
```

レイアウトウインドウが生成される。

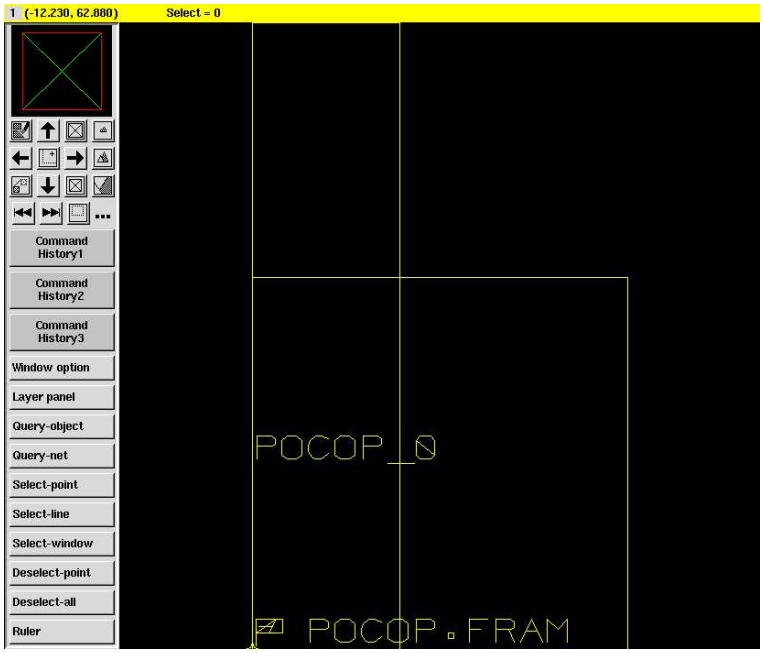

図 1: 読み込んだ後

```
source ./scripts/apply_tlu_plus.tcl
```

これはマクロと同じである。

2.3 フロアプラン

以下がフロアプランである。この辺の tcl は社長の作で、サイズを計算している。

```
source ./scripts/POCOP_TOP_floorplan.tcl
source ./scripts/POCOP_TOP_get_coordinate.tcl

set pocop_left     [expr ($core_left + $core_right) / 2 - ($pocop_width / 2)]
set pocop_bottom   [expr ($core_bottom + $core_top) / 2 - ($pocop_height / 2)-1.8]
set pocop_right    [expr $pocop_left + $pocop_width]
set pocop_top      [expr $pocop_bottom + $pocop_height]
```

このうち、POCOP_TOP_floorplan.tcl は、フロアプランの指定だが、この最後にセル境界の設定を以下のように行っていて、これが必須。

```
dbCreateCellBoundary   [geGetEditCell] [list "0 0" "3816 0" "3816 1716" "0 1716" "0 0"]
```

POCOP_TOP_get_coordinate.tcl は tcl 内で、ruby を起動し、コアのそれぞれの諸元の計算を行っている。これは社長の作ですごいと思う。ふんがはあまり良く理解してない。

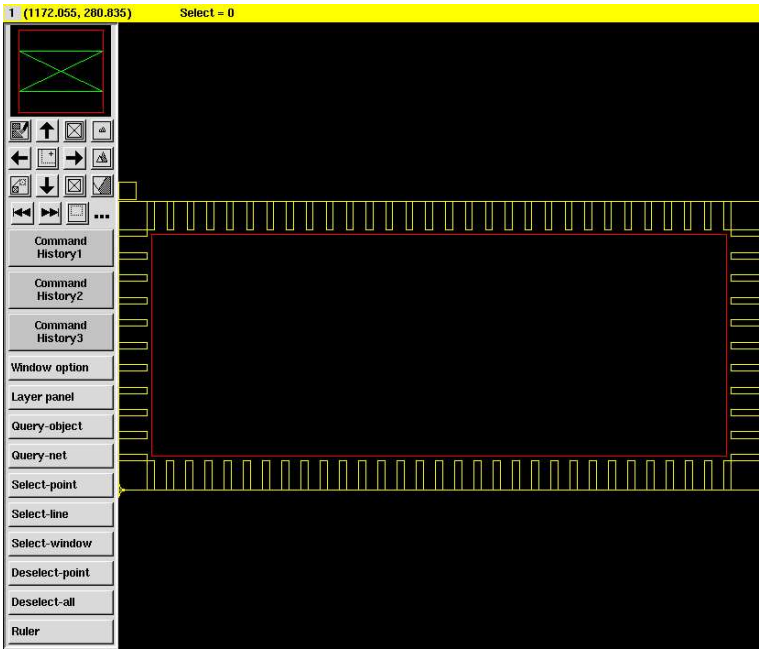

図 2: フロアプラン後

I/O は、コアの周辺とバウンダリの間に適当に並べてくれているが、これは間延びしているのがわかる通りあくまでいい加減に並べたもの。左上にマクロが確認できる。

次にマクロを配置する。チップのど真ん中に置いた。

```
source ./scripts/POCOP_TOP_place_macro.tcl
```

中身

```
dbSetCellInstPlacement [geGetEditCell] "POCOP_0" "0" "no" "origin" [list $pocop_left 805.4]
```

この場所は、適当ではあるが、周囲とピッチを揃えて DRC がでないように後で調整したもの。

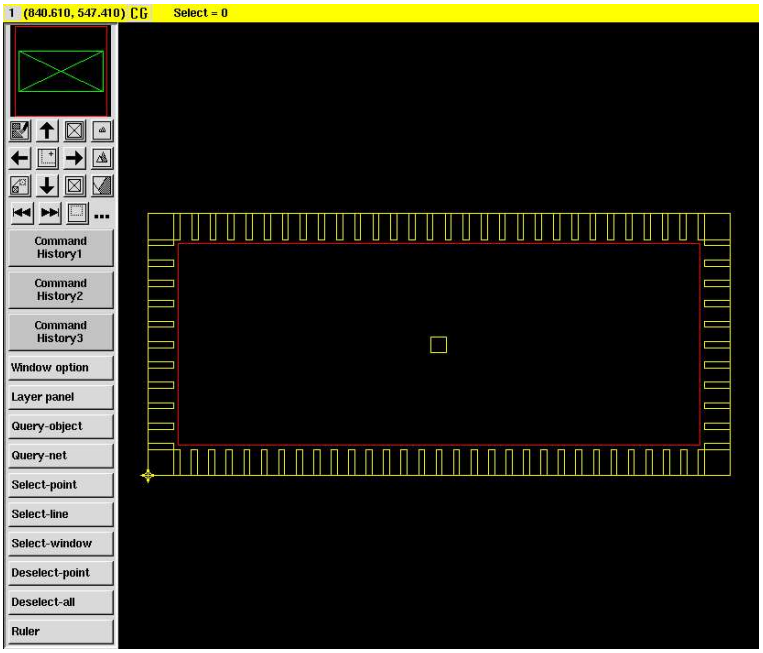

図 3: マクロの配置

2.4 I/O の配置

これは実行は一文で行うようになっているのだが、中身を作るのには大変時間が掛かる。マクロの I/O の配置には tdf を利用した。I/O の配置にも利用可能であるはずだが、なぜかあまり上手く行かないのと、WIRE_PAD を重ねる必要があるので、直接指定する方法を取っている。もっと格好良い方法があれば教えて欲しい。

```
source ./scripts/POCOP_TOP_place_pad.tcl
```

まず最初にバウンダリを設定する。

```
dbCreateCellBoundary [geGetEditCell] [list "0 0" "3816 0" "3816 1716" "0 1716" "0 0"]
```

これはさっきと同じである。(必要か?)

次に、Verilog 記述に表れないものは、定義する必要がある。まず、コーナーセル (ZCGCB2E4C0XXA1)、電源用 I/O(VDD 用:IOCB2EPD5PI11, VSS 用:IOCB2EPG5PB11)、VDE:つまり入出力 3.3V 用 (IOCB2EW3AOC0A1) を定義する。

さらに信号線を含むすべての I/O にワイヤパッドが必要である。ワイヤパッドを物理的なパッドにぴったり重ねてやらないと、配線をしてくれない。(最初にテープアウトした際に、この情報を誰も教えてくれず、泣いた。)

VDE 用:IOCB2EW3AOC0A1、VDD 用:IOCB2EWDAOC0A1、VSS 用:IOCB2EWGAOC0A1 信号線用:IOCB2EWSAOC0A1

```
dbCreateCellInst [geGetEditCell] "" "ZCGCB2E4C0XXA1.FRAM" "PAD_CORNER_B_L" "180" "" "0 0"
dbSetCellInstPlacement [geGetEditCell] "PAD_CORNER_B_L" "180" "no" "origin" "170 170"
```

まずコーナーセルを置く。上の記述は左下 (B-L) の例である。コーナーセルの原点は、左下で、これを 180 度回転して置く必要がある。他のコーナーセルも、回転角度を考えて右下は 270 度、右上は 0 度、左上は 90 度回転させて原点を考慮して置く。

次に電源と信号線の I/O を置くのだが、以下大変めんどくさい。

- PAD の位置を考えると、I/O は 60um ピッチで置く必要がある。
- 電源、GND 用の PAD は横幅が 45um だが、信号線用は 40um である。

- I/O の原点は左下。この場所はチップ側 (内側) である。このため、チップの下部に配置する場合は 180 度、右側は 270 度、上部は 0 度、左側は 90 度回転させる必要がある。
- すべての I/O にはワイヤパッドを重ねる。

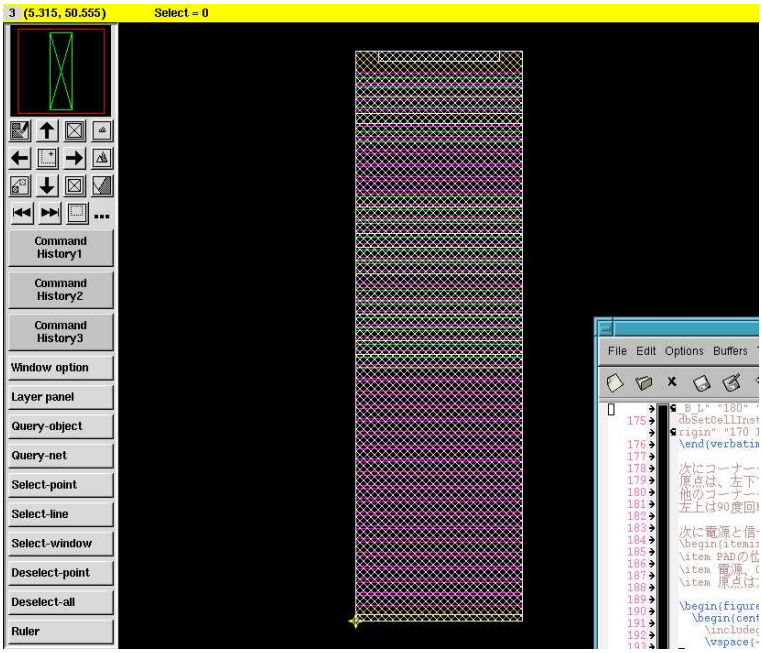

図 4: I/O

上が I/O の原点を示す図で、下がチップ左下の配置を示す。
したがって、各 I/O は、その幅、原点位置と回転を考えて配置していかなければならない。

```
dbCreateCellInst [geGetEditCell] "" "IOCB2EPE5PE11.FRAM" "PAD_VDE01" "180" "" "0 0"
dbSetCellInstPlacement [geGetEditCell] "PAD_VDE01" "180" "no" "origin" "223 170"
dbCreateCellInst [geGetEditCell] "" "IOCB2EW3AOC0A1.FRAM" "WIRE_PAD_VDE01" "180" "" "0 0"
dbSetCellInstPlacement [geGetEditCell] "WIRE_PAD_VDE01" "180" "no" "origin" "223 170"
dbCreateCellInst [geGetEditCell] "" "IOCB2EPG5PB11.FRAM" "PAD_VSS_01" "180" "" "0 0"
dbSetCellInstPlacement [geGetEditCell] "PAD_VSS_01" "180" "no" "origin" "283 170"
dbCreateCellInst [geGetEditCell] "" "IOCB2EWGAOC0A1.FRAM" "WIRE_PAD_VSS_01" "180" "" "0 0"
dbSetCellInstPlacement [geGetEditCell] "WIRE_PAD_VSS_01" "180" "no" "origin" "283 170"
```

これは、図中に示した 2 つの I/O (VDE, VSS) を示す。これは両方とも幅が 45um なので、間隔が同じだが、信号線用が混ざると調整する必要がある。
このため、配置後は、I/O 間のすき間が不均一だが、これが正しく、すき間が均一になるのは多分どこかがおかしい。
ピン配置図から自動的にこの配置を作るスクリプトは、みんな試したが結局 ad hoc なものになって残っていない。相当面倒だが誰かやっておくれ。

`POCOP_TOP_insert_pad_filler.tcl`

今回のフローでは、フィラーを並べることで、PAD 上に電源リングを形成する。Rhom のと違って、PAD 上の電源リングを経由して、すべての I/O に 3.3V 電源が配給される。したがって、このための配線は必要ない。しかし、この方法は、Geyser や SMA など電源を分離する場合には、フィラーを切らなければならない。また、どこかのフィラーが抜けていると致命的な問題を引き起こすことになる。このため十分なチェックが必要である。

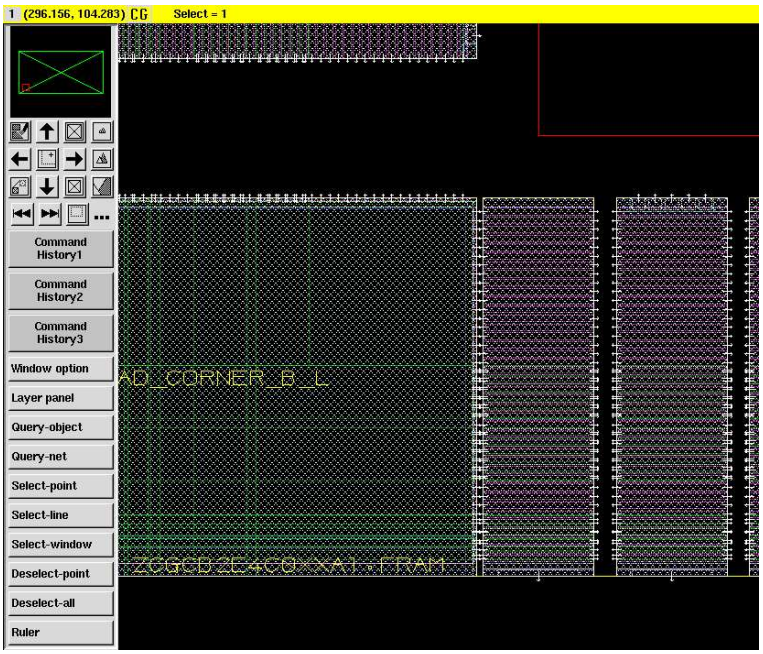

図 5: I/O とコーナーの配置

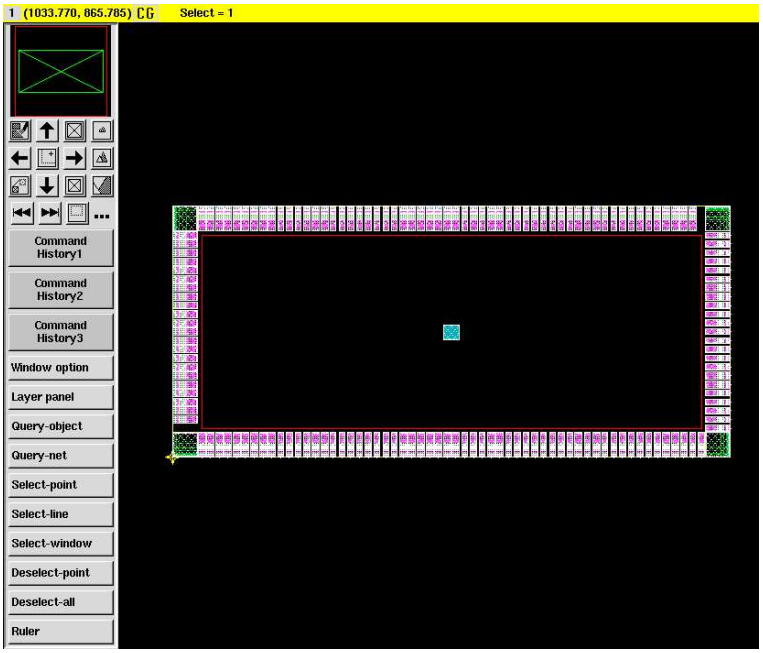

図 6: I/O の配置後

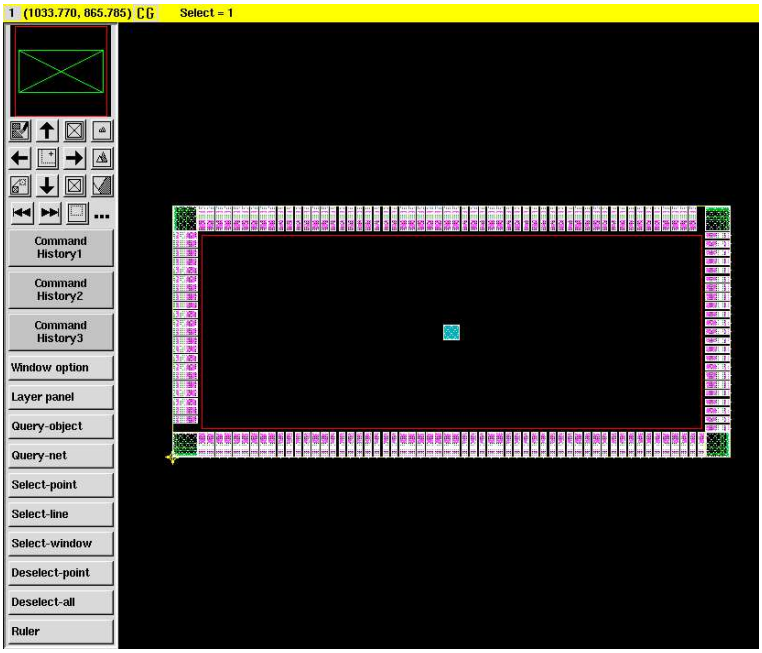

図 7: PAD filler

2.5 電源リング

以下で電源リングを作る。

```
source scripts/POCOP_TOP_create_ring.tcl
source scripts/POCOP_TOP_connect_pg.tcl
```

これは、社長の設定を使った所、二重巻になるのだが、本人そのつもりはなかったとのこと。この辺もっといじくって色々可能性を試してみないといけないがとりあえずいいことにする。さらに、connect_pg で電源に論理的な接続を行う。

次に、I/O と電源リングをコネクトする。

```
source ./scripts/POCOP_TOP_connect_pad_ring.tcl
source ./scripts/POCOP_TOP_connect_pg.tcl
```

次にマクロと電源リンクの間を接続する。

```
source ./scripts/connect_ring.tcl
```

中身

```
axgPrerouteInstances
formDefault preroute_instances
setToggleField preroute_instances instance_type(s) macro 1
setToggleField preroute_instances instance_type(s) pad 1
setToggleField preroute_instances instance_type(s) cover 0
formOK preroute_instances
```

これで、マクロの電源ストライプと電源リングの間に接続ができる。

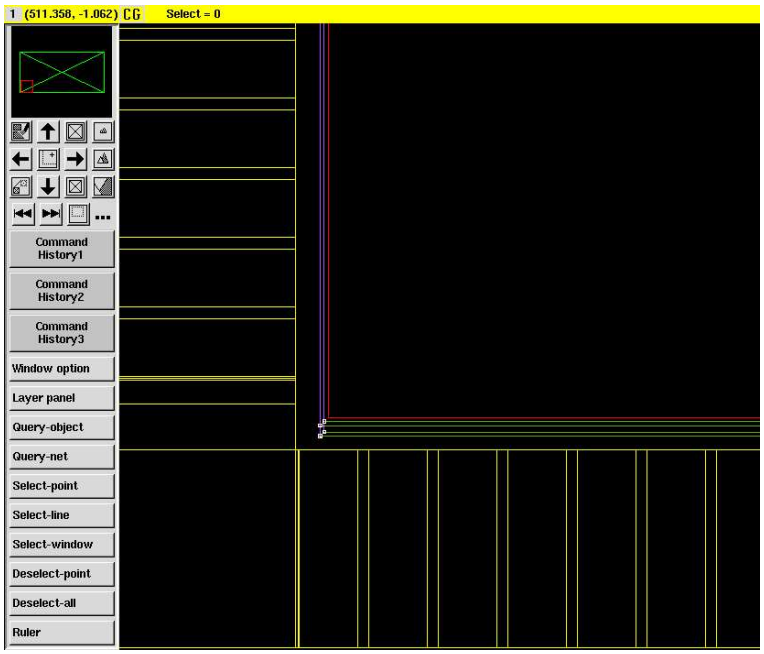

図 8: 電源リング

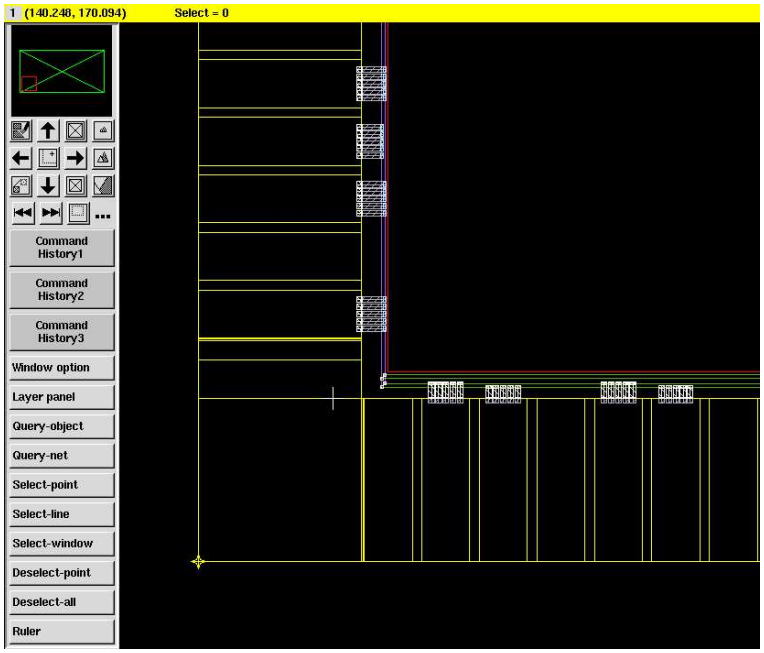

図 9: 電源リングと I/O との接続

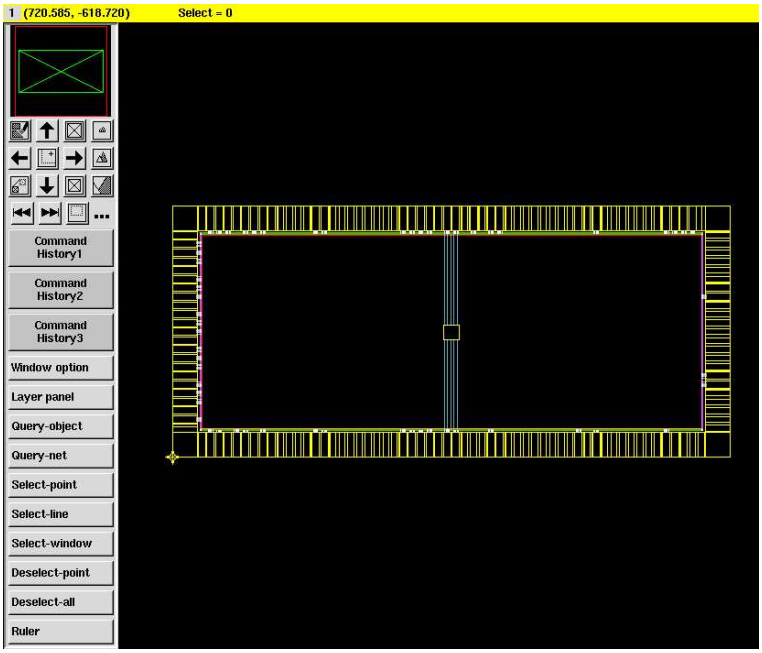

図 10: 電源リングとマクロとの接続

2.6 トップ階層の電源メッシュ

次に、トップ階層に電源ストラップ、タップセル挿入、フィラー用のレール入れを行うが、その前にマクロ周辺には入れないようにブロッケージを張っておく。

```
source ./scripts/macro_blockage.tcl
```

次にストラップ入れ、タップ挿入、レール挿入を行う。

```
source ./scripts/POCOP_TOP_strap.tcl
source ./scripts/POCOP_TOP_tap.tcl
source ./scripts/route_rail.tcl
```

間隔はルールに則って適当に決める。全体図は以下のようになる。
これだとレールが見えないが、ちゃんとできている。

2.7 配線

次に、一度ブロッケージを消去し、電源を接続、TLU を読ませて配線の準備をする。

```
source ./scripts/remove_blockage.tcl
source ./scripts/connect_pg.tcl
source ./scripts/apply_tlu_plus.tcl
```

今回は、マクロのターミナルと周辺を接続するだけで、CTS も不要なので、いきなりつないでしまう。

```
source ./scripts/auto_route.tcl
source ./scripts/post_route_opt.tcl
```

上記はマクロの時に使ったのと同じである。

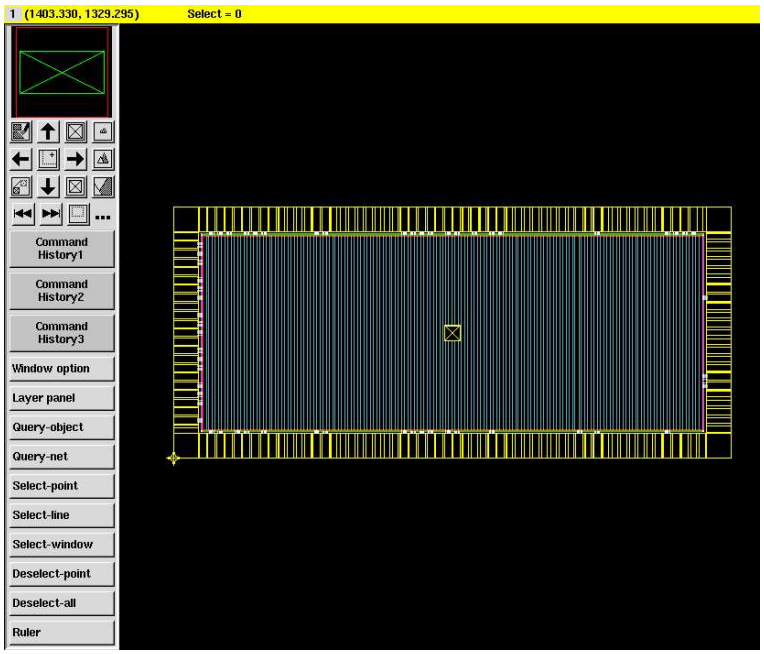

図 11: ストラップ、レールタップ挿入後

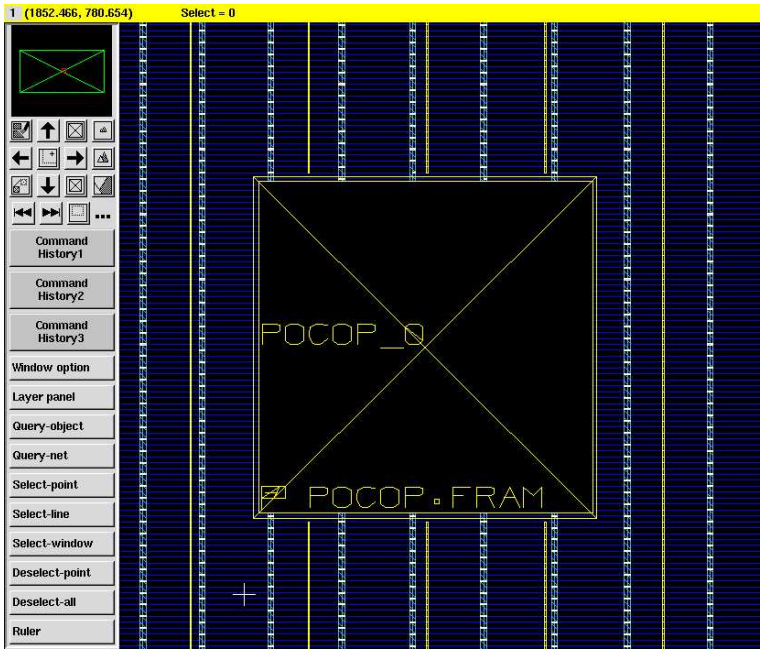

図 12: ストラップ、レールタップ挿入後 (拡大図)

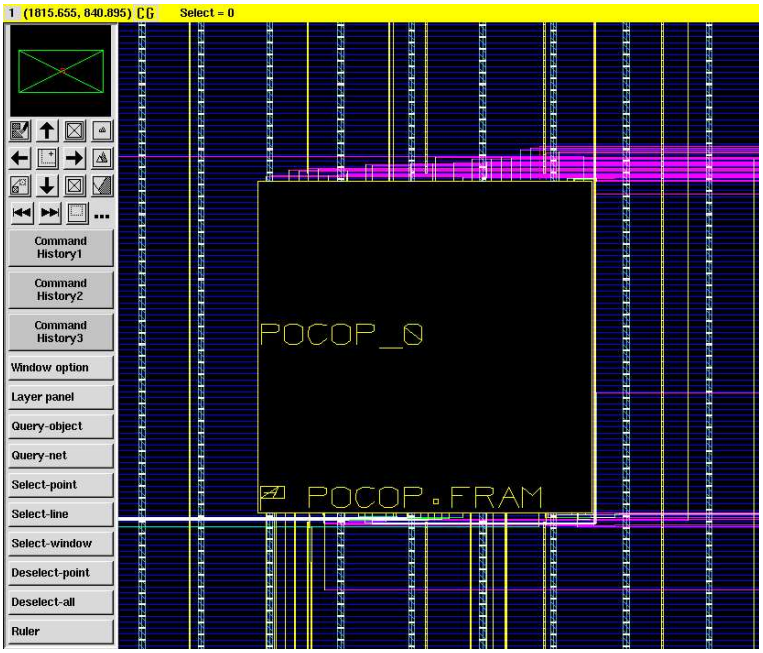

図 13: 配線後の結果

2.8 仕上げ

ERC を防ぐため、一定の間隔で CUBA を入れ、残った部分に普通のフィラー YUZS を入れる。

```
source ./scripts/macro_blockage.tcl
source ./scripts/POCOP_TOP_cuba.tcl
source ./scripts/POCOP_TOP_addfiller.tcl
source ./scripts/POCOP_TOP_connect_pg.tcl
```

これはマクロのとほぼ同じで、最初全面 Cuba で埋めておいて、一定の間隔を残してこれを削る。面積が広いので時間が掛かる。

最後に原点を移動する。これは、Calibre でフレームと重ねるためである。

```
source ./scripts/moveOrigin.tcl
```

最後に必ずセルを保存してから、ストリームアウトする。さらに LVS 用の Verilog 記述を保存する。

```
source ./scripts/save_cell.tcl
geSaveAs
setFormField "Save As" "Cell Name" "04_post_filler"
formOK "Save As"

source ./scripts/stream_out.tcl
source ./scripts/verilog_out_last.tcl
```

後は、Calibre 四天王を参照のこと。今回の設計は、初期段階で以下の問題が発生した。

- マクロとの境界付近に DRC が出た。マクロの位置を調節すると共に、ブロッケージを利用して、周辺からセルを遠ざけ、これらを消した。

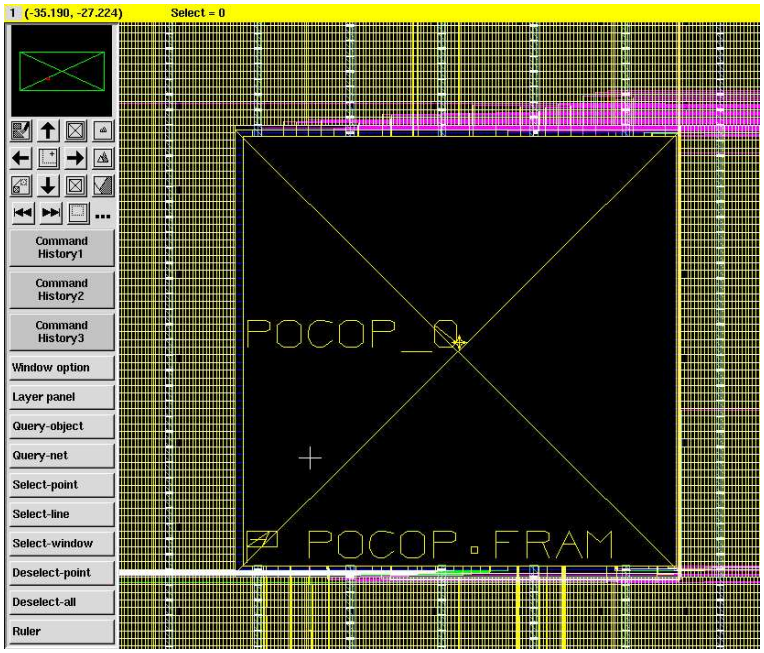

図 14: 最終レイアウト

- ANT はほとんどのマクロ入力で出た。これは、線が長いので当然かもしれない。元の POCOP の入力の Verilog 記述にダイオード SC23DS01 を入れ回避した。これはぶっちさんの一度 Verilog を吐き出すノウハウではなく、元に入れたが、マクロが単体なのでうまく行った。

```
SC23DS01 DS_RST (.A(RST_N));
SC23DS01 DS_ID0 (.A(IDATAIN_0_));
```

- LVS では、cdl 内で VDD と VDE をコネクトしている記述があり、これを消去した。
- ERC は、Cuba などの対策のおかげで問題は生じなかった。