Section 1:

CUBE chip design flow

CUBE chip design flow (English version)

Hiroki Matsutani *

October 21st, 2010

Table of contents

1 Chip overview

CUBE chip consists of the following four modules (Figure 1).

- cube core: On-chip routers, cores, bus controller, etc
- ptp core0: Inductors for point-to-point downlink communications
- ptp core1: Inductors for point-to-point uplink communications
- sb4 core: Inductors for broadcast bus communications

We used the inductive-coupling transceiver circuits for ptp core0, ptp core1, and sb4 core in a real chip. In this design flow, however, we use Dummy inductor cores which can be fully synthesized for ease of explanation. Thus, for a real chip design, we have to replace these Dummy inductor cores with real inductive-coupling transceiver circuits.

Notice that three power lines are required for this chip since ptp cores and sb4 core require VDD, VDDBR, and VDDBC. For more detail, refer to "Appendix A: CUBE_TOP layout flow".

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

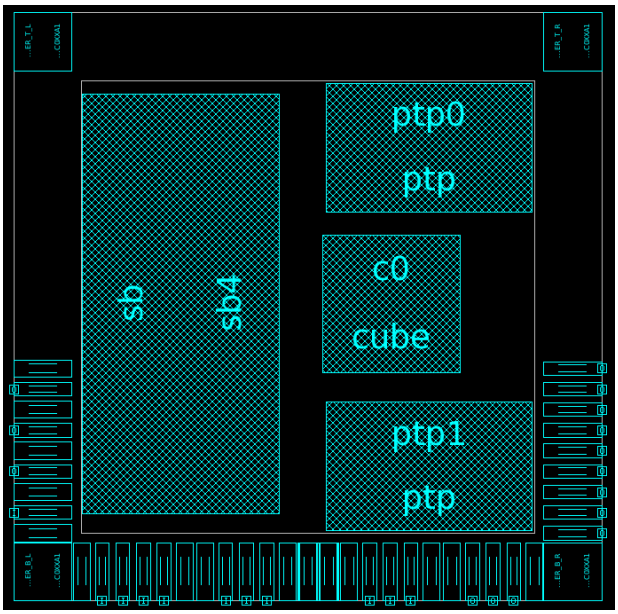


Figure 1: CUBE chip floorplan.

2 cube core: operation modes

The test module test.v supports the following four operation modes.

Table 1: Test module's operation modes

Mode 0	Point-to-point	(Single: packet injections to random destination)
Mode 2	Point-to-point	(Burst: packet streams to specified destination)
Mode 1	Shared-bus	(Single: packet injections to random destination)
Mode 3	Shared-bus	(Burst: packet streams to specified destination)

Change $mode value in sim/rom.pl. Then sim/rom.pl generates Event ROM file according to the mode specified. For more detail, refer to "Section 2: CUBE chip specification".

3 cube core: RTL simulation

Refer to "Section 2: CUBE chip specification" for more details on the RTL simulation.

```
cd cube/sim

vi rom.pl (Set $mode to 0)
./rom.pl >! event.hex
make sim

vi rom.pl (Set $mode to 1)
./rom.pl >! event.hex
make sim

vi rom.pl (Set $mode to 2)
./rom.pl >! event.hex
make sim
```

```
vi rom.pl (Set $mode to 3)
./rom.pl >! event.hex
make sim
```

You can find "Dump file" statement in the end of test.v. If you comment out this description, a value dump file test.vcd is generated after the simulation. Open test.vcd with a wave viewer (e.g., Simvision or GTKwave) and check the waveform.

4 cube core: Synthesis

[Modification for cube/syn/scripts/compile_dc.tcl] Set LIB_DIR to a directory path of standard cell db file.

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

See syn.log for Design Compiler's log and cube.rep for the synthesis report.

5 cube core: Post-synthesis simulation

[Modification for cube/sim/Makefile] Set LIBV to a directory path of standard cell Verilog model.

```
cd cube/sim

vi rom.pl (Set $mode to 0)
./rom.pl >! event.hex
make ssim

vi rom.pl (Set $mode to 1)
./rom.pl >! event.hex
make ssim

vi rom.pl (Set $mode to 2)
./rom.pl >! event.hex
make ssim

vi rom.pl (Set $mode to 3)
./rom.pl >! event.hex
make ssim
```

6 cube core: Place and route (layout)

[Modification for cube/pr/Makefile] For "setup" rule, set a directory path of standard cell Milkyway library and set file paths of required technology files.

[Modification for cube/pr/scripts/set_sz.tcl] Set LIB_DIR to a directory path of standard cell db file. Set IO_LIB_DIR to a directory path of I/O cell db file. Set DW_DIR if needed.

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

"make pr" first generates symbolic links to necessary technology files (see "setup" rule). Then it performs the place and route of the design using IC Compiler.

7 cube core: Post-layout simulation

[Modification for cube/sim/scripts/convert_sdf.tcl] Set LIB_DIR to a directory path of standard cell db file.

First, the delay information (SDF file) generated by ICC has to be converted to another form which is capable of NC Verilog simulation, by using Synopsys PrimeTime.

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

Perform a post-layout simulation by using the converted delay file (cube_pt.sdf).

```
vi rom.pl (Set $mode to 0)
./rom.pl >! event.hex
make psim

vi rom.pl (Set $mode to 1)
./rom.pl >! event.hex
make psim

vi rom.pl (Set $mode to 2)
./rom.pl >! event.hex
make psim

vi rom.pl (Set $mode to 3)
./rom.pl >! event.hex
make psim
```

Be careful of size of the value dump file when "Dump file" statement in the end of test.v is enabled. It is quite large.

8 cube core: Formal verification

[Modification for cube/verify/scripts_misc/verify_fm.tcl: Set LIB_DIR to a directory path of standard cell db file.

Perform the following verification after the place and route of cube core.

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

Check fm.log. If all items of "Failing (not equivalent)" are zero, the placed and routed netlist is logically correct.

9 cube core: DRC, LVS, ERC, and ANT verifications

[Modification for cube/verify/Makefile] For "setup" rule, set a file path of standard cell gds file and set file paths of required technology files.

[Modification for cube/verify/Makefile] For "cdl" fule, set a file path of standard cell cdl file.

Perform the following verification setup after the place and route of cube core.

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

"make setup" generates symbolic links to Fujitsu 65nm verification scripts. The gds file generated by ICC does not contain the gds image of standard cells. "make gds" embed the gds image of standard cells in the cube core gds to generate a complete gds that contains everything for the verification. "make cdl" generates SPICE netlist of cube core for LVS. "make ed" generates pin location file (ED TEXT) of cube core for LVS.

Perform Design rule check (DRC).

```
./cal_drccs2001
Answer the questions.
./cube_drc_run.csh
```

Perform Layout versus schematic (LVS) and Electric rule check (ERC).

```
./cal_lvscs2001
Answer the questions.
./cube_lvs_run.csh
```

Perform Antenna rule check (ANT).

```
./cal_antcs2001
Answer the questions.
./cube_ant_run.csh
```

"make setup" has generated the default answer files (.rsf.setup_ant, .rsf.setup_lvs, and .rsf.setup_drc). For the questions from the verification scripts, you can just select the default values.

10 Dummy inductors: Synthesis, place, and route

[Modification for ptp/syn/scripts/compile_dc.tcl] Set LIB_DIR to a directory path of standard cell db file.

[Modification for ptp/pr/Makefile] For "setup" rule, set a directory path of standard cell Milkyway library and set file paths of required technology files.

[Modification for ptp/pr/scripts/set_sz.tcl] Set LIB_DIR to a directory path of standard cell db file. Set IO_LIB_DIR to a directory path of I/O cell db file. Set DW_DIR if needed.

[Modification for sb4/syn/scripts/compile_dc.tcl] Set LIB_DIR to a directory path of standard cell db file.

[Modification for sb4/pr/Makefile] For "setup" rule, set a directory path of standard cell Milkyway library and set file paths of required technology files.

[Modification for sb4/pr/scripts/set_sz.tcl] Set LIB_DIR to a directory path of standard cell db file. Set IO_LIB_DIR to a directory path of I/O cell db file. Set DW_DIR if needed.

Synthesize the ptp core.

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

Place and route the ptp core.

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

Synthesize the sb4 core.

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

Place and route the sb4 core.

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

11 CUBE_TOP: Place and route

[Modification for CUBE_TOP/pr/Makefile] For "setup" rule, set a directory path of standard cell Milkyway library and set file paths of required technology files.

[Modification for CUBE_TOP/pr/scripts/set_sz.tcl] Set LIB_DIR to a directory path of standard cell db file. Set IO_LIB_DIR to a directory path of I/O cell db file. Set DW_DIR if needed.

For more detail on the layout, refer to CUBE_TOP/pr/scripts/CUBE_TOP.tcl and "Appendix A: CUBE_TOP layout flow". This chip uses three power lines: VDD, VDDBC, and VDDBR. Divider I/O cells are inserted to the right and left of VDDBC and VDDBR pads to separate these power lines. Otherwise, these power lines are connected at the I/O pads, resulting in short circuits in the power supply. For more detail on the divider cell insertion, refer to CUBE_TOP/pr/scripts/ioplace.tcl.

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

Unfortunately, a DRC error cannot be removed by the auto place and route by ICC, so I recommend you to execute each place-and-route command manually rather than "make pr" for CUBE_TOP layout. Launch IC Compiler with GUI mode.

```
icc_shell -gui
```

Execute each place-and-route command in CUBE_TOP/pr/scripts/CUBE_TOP.tcl manually until "STOP HERE. PLEASE RUN DRC." line. To do so, copy and paste each command to "icc_shell>" of MainWindow and type Enter.

After completing all commands before "STOP HERE. PLEASE RUN DRC." line, execute "Verification $\mapsto$ DRC" of LayoutWindow.

You have to remove only a DRC error of "MaxWidth 1" by hand. For more detail on how to remove it, refer to "Appendix B: CUBE_TOP DRC-free how-to".

After fixing the "MaxWidth 1" error, try DRC again and check the DRC report to confirm that the error has been removed. If the error is removed, execute all the remaining commands after "STOP HERE. PLEASE RUN DRC." line.

12 CUBE_TOP: Frame

[Modification for CUBE_TOP/frame/Makefile] For "setup" rule, set file paths of required technology files.

[Modification for CUBE_TOP/frame/Makefile] For "stdcell" rule, set a file path of standard cell gds file (cs202sz_uc.gds). In the same way, set file paths of cs202_fm.gds, frames.str, MB8AW4203_FRAME.gds, and cs202_io.gds to the appropriate rules.

The chip frame MB8AWXXXX_FRAME.gds will be provided for each chip application from VDEC. Use the appropriate frame for your tape-out.

Generate symbolic links to gds files of cube core, dummy inductors (ptp and sb4), and CUBE_TOP in CUBE_TOP/frame/input/.

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

Read all gds files including cube core, dummy inductors (ptp and sb4), standard cells, I/O cells, frame, and so on.

```
make base
```

Embed all child modules to the frame. For more detail on building the frame, refer to "Appendix C: CUBE chip frame insertion".

```
make CUBE_TOP.addframe
```

The final gds file that includes everything is saved in CUBE_TOP/frame/CUBE_TOP.gds.

13 CUBE_TOP: DRC, LVS, ERC, and ANT verifications

[Modification for CUBE_TOP/verify/Makefile] For "setup" rule, set file paths of required technology files.

[Modification for CUBE_TOP/verify/Makefile] For "CUBE_TOP.cdl" fule, set a file path of standard cell cdl file.

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

"make setup" generates symbolic links to Fujitsu 65nm verification scripts. It also generates symbolic links to Verilog models of cube core, dummy inductors (ptp and sb4), and CUBE_TOP for LVS in CUBE_TOP/frame/input/. To generate the Verilog model for chip-level LVS by IC Compiler, you need to perform write_verilog with -no_physical_only_cells option.

Generate SPICE netlists for LVS from the Verilog models for LVS. Please execute "make CUBE_TOP.cdl" in an environment where ruby command is installed since we use ruby for the text processing.

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

SPICE netlists for LVS are saved in CUBE_TOP/frame/cdl.

At first, we declared VDD and VSS as global in the SPICE netlists. However, since the LVS tool mixed up VDD and VDDBC/VDDBR and reported LVS errors, we are not using any global declarations. That is, we have to specify VDD, VSS, (VDE, VDDBC, VDDBR) ports for each module in the SPICE netlists. Also, we have to remove "VNW=VNW" and "VPW=VPW" from I/O and corner cells.

CUBE_TOP.cdl includes child cdl files, such as cube.cdl, sb4.cdl, ptp.cdl, cs202sz_uc.cdl, and cs202_io.cdl. It is OK for the LVS at local machines. For the final LVS at VDEC web site, on the other hand, the cdl file must be self-contained (i.e., all the cdl files should be a single file). For more detail on the cdl file preparation, refer to CUBE_TOP/verify/Makefile.

Perform Design rule check (DRC).

```
./cal_drccs2001
Answer the questions.
./CUBE_TOP_drc_run.csh
```

Perform Layout versus schematic (LVS) and Electric rule check (ERC).

```
./cal_lvscs2001
Answer the questions.
./CUBE_TOP_lvs_run.csh
```

Perform Antenna rule check (ANT).

```
./cal_antcs2001
Answer the questions.
./CUBE_TOP_ant_run.csh
```

"make setup" has generated the default answer files (.rsf.setup_ant, .rsf.setup_lvs, and .rsf.setup_drc). For the questions from the verification scripts, you can just select the default values.

Section 2:

CUBE chip specification

CUBE chip specification (English version)

Hiroki Matsutani *

October 20th, 2010

Table of contents

1 Chip I/O and control registers

There are only 12 pins available for data I/O, clk, and rst_. These pins are used as follows. The reset signal is active-low.

```
IN  Clock (clk), 1-bit
IN  Rest (rst_), 1-bit, active-low
IN  Input data (idata), 4-bit
OUT Output data (odata), 3-bit
IN  Register bank select (sel), 2-bit
IN  Register write enable (wr), 1-bit
```

For the control register write, first select the register bank by sel signal, then assert the write enable (wr) so that the input data (idata) are saved to the specified register bank.

For the control register read, the value of the register bank specified by sel signal appears at odata.

```
    Table 1: Control register format. X indicates the bit may be deletable.
  ======================================================================
  Control register 0 (sel == 0):
    idata[0]      Core select (core 0 or core 1)
    idata[1]      Packet injection to pre-specified dest using vch
    idata[2]      Dummy mode (Do not set 1)
  X idata[3]      Clear the packet counter of selected core
    odata[0]      Packet-injection ready status for VC0 of selected core
    odata[1]      Packet-injection ready status for VC1 of selected core
  X odata[2]      Selected core (core 0 or core 1)
  ----------------------------------------------------------------------
  Control register 1 (sel == 1):
    idata[3:0]    Destination (dest) address of packets (4-bit)
```

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

```
X odata[2:0]    Packet counter value (0-3 bit)
-----------------------------------------------------------------
Control register 2 (sel == 2):
  idata[0]      Virtual channel (vch) ID of packets (1-bit)
  idata[1]      Burst mode enable
  idata[2]      NoC mode (0: Point-to-point, 1: Shared bus)
  idata[3]      Start reading the packet counter of selected core (cread)
  odata[2:0]    Packet counter value (45-bit, 3-bit for each cycle)
-----------------------------------------------------------------
Control register 3 (sel == 3):
  idata[2:0]    Chip ID (0-7)
  idata[3]      Am I the top chip?
X odata[2:0]    Packet counter value (4-7 bit)
-----------------------------------------------------------------
```

When cread signal is asserted, the packet counter value of selected core is read from odata of register 2. That is, 3-bit value of the counter is read sequentially using 15 cycles in total.

When inject signal is asserted with the burst mode ON, packets to dest using vch are continuously injected until the burst mode is disabled.

Refer to sim/test.v for examples of the chip initialization (e.g., Chip ID and mode), the packet injection procedure, and the packet counter read procedure.

2 Architecture, topology, and routing

Each chip has two cores. Each core has a router.

Up to eight chips (16 cores) can be stacked in a package, although we just stack only four chips (8 cores) in the simulator (see sim/test.v) for ease of understanding.

Chip-3 is the top chip and Chip-0 is the bottom chip. In each chip, left core (router) has an even ID while the right core (router) has an odd ID.

```
Chip-3 [Router_6] <--> [Router_7]
           V               A
Chip-2 [Router_4] <--> [Router_5]
           V               A
Chip-1 [Router_2] <--> [Router_3]
           V               A
Chip-0 [Router_0] <--> [Router_1]
```

Two cores on the same chip are connected via a bi-directional horizontal link. Even routers have a downstream link while odd routers have a upstream link.

Packets using virtual channel 0 cannot use any horizontal links, except routers located in the top or bottom chips. For example, a packet from router 1 to router 2 goes through router 1, router 3, router 5, router 7, router 6, router 4, and router 2.

Packets using virtual channel 1 can use the horizontal links only when the destination is located in the same chip. Otherwise, they also follow the same rule of virtual channel 0. That is, a packet from router 1 to router 2 goes through router 1, router 3, and router 2.

3 Packet format and flit format

Each packet consists of a head flit, three data flits, and a tail flit.

Destination address is stored in the head flit. Payload data are stored in the data and tail flits. Crossbar switch allocation is released when the tail flit goes through the crossbar.

Table 2: Flit format. X indicates the field may be deletable.

Head flit:		
	flit[33:32]	Flit type (2'b01 indicates a head flit)
	flit[31:8]	Random value generated by m-sequence
X	flit[7:4]	Flit ID in a packet (0 for head flit)
	flit[3:0]	Destination core address
Data flit:		
	flit[33:32]	Flit type (2'b10 indicates a data flit)
	flit[31:8]	Random value generated by m-sequence
X	flit[7:4]	Flit ID in a packet (1-3 for body flits)
X	flit[3:0]	Destination core address
Tail flit:		
	flit[33:32]	Flit type (2'b11 indicates a tail flit)
	flit[31:8]	Random value generated by m-sequence
X	flit[7:4]	Flit ID in a packet (4 for tail flit)
X	flit[3:0]	Destination core address

Perform the RTL simulation of sim/test.v. Then you can see that packets are transferred in the ring network.

4 Event ROM format

In sim/test.v, packets are injected to the network according to the Event ROM (sim/event.hex).

Table 3: Event ROM format. The values are in hexadecimal.

event[43:12]	Clock that indicates when the packet is injected
event[11:8]	Virtual channel used (VC0 or VC1)
event[7:4]	Source core (0-7)
event[3:0]	Destination core (0-7)

At the packet injection clock, the source core starts the packet injection procedure that configures necessary control registers of the core. It takes several cycles for the procedure before the packet is actually sent.

Below is an example of Event ROM file. The first line specifies the NoC mode and the second line specifies the number of events in the ROM.

```
0                  // NoC mode 0-3 (See Table 1 of Section 1)
3                  // Number of events is 3
00000010_0_3_2     // At 16th clock, core 3 sends to core 2 using VC0
00000020_1_7_6     // At 32nd clock, core 7 sends to core 6 using VC1
00000030_1_2_3     // At 48th clock, core 2 sends to core 3 using VC1
```

5 Probe signals

In addition to the 12 I/O pins, only the top chip provides additional 9 probe pins for measuring its internal states.

Table 4: Probe signals (9 pins)

probe[1:0]	Chip ID of core 0 (id_0[2:1])
probe[2]	Timeslot signal of core 0 (ts_control)
probe[4:3]	Flit type of a link from core 0 to router 0
probe[6:5]	Flit type of a link from router 0 to router 1
probe[8:7]	Flit type of a link from router 1 to core 1

The following measurements are possible.

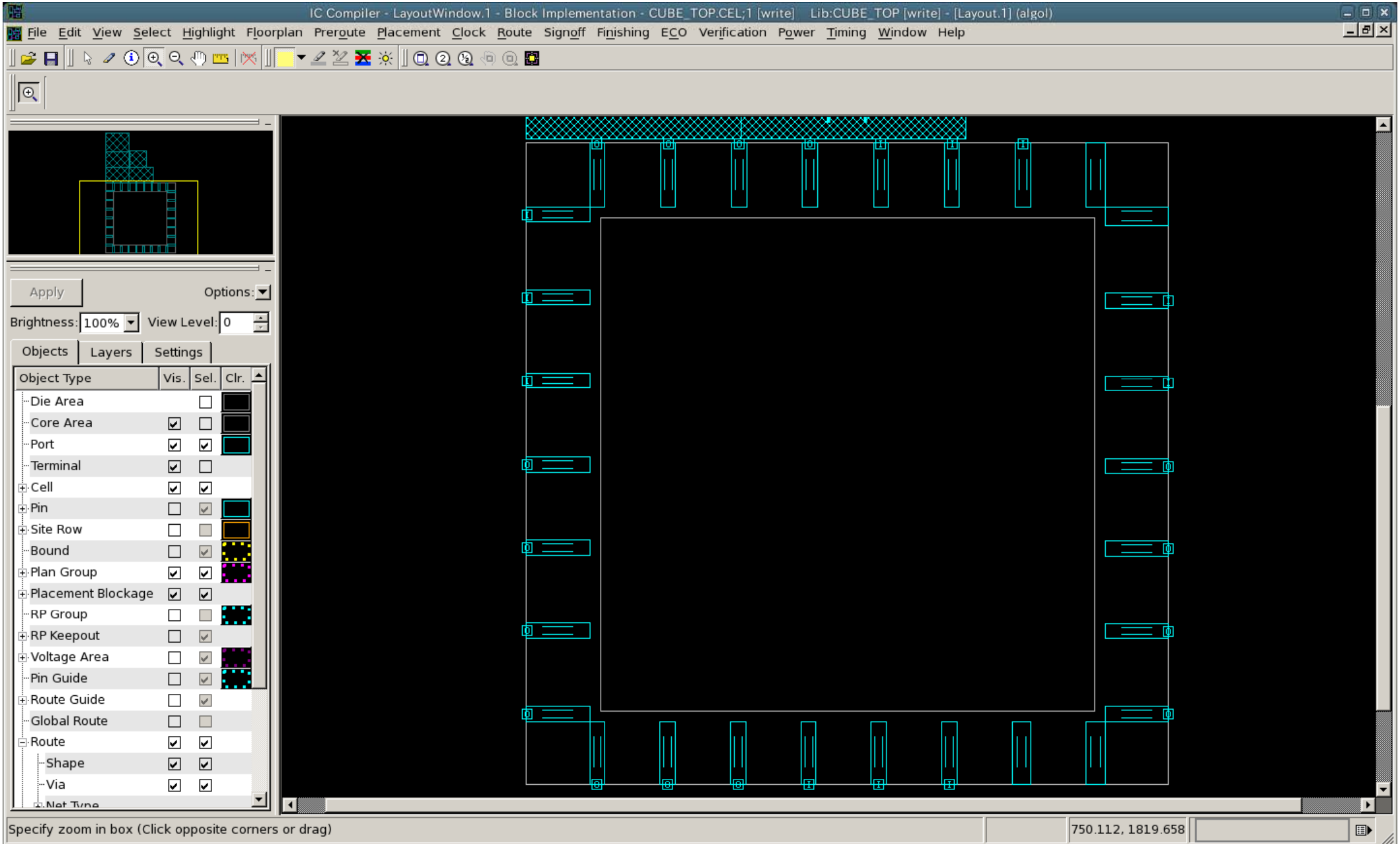
- Write a control register. Then read probe[1:0] to check the register has a correct value.
- Read probe[2] to check the shared bus controller gives a time-slot for each plane correctly.
- Send a packet from core 0 to core 1. Then read probe[8:3] to check the packet is transferred on the link correctly.

Appendix A:

CUBE_TOP layout flow

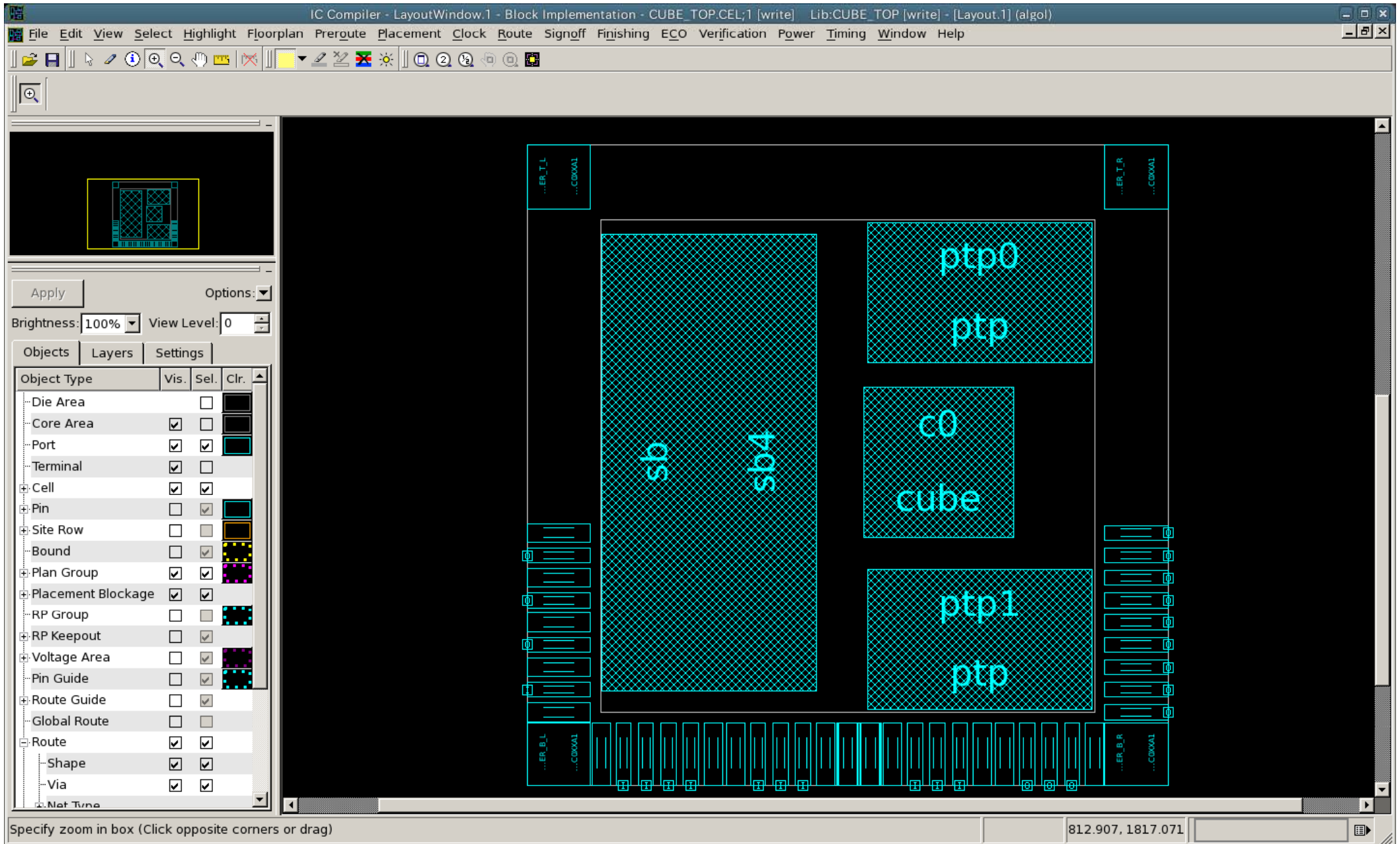
CUBE_TOP layout flow 1/8

Specify the chip area (1320um x 1320um) by initialize_floorplan command.



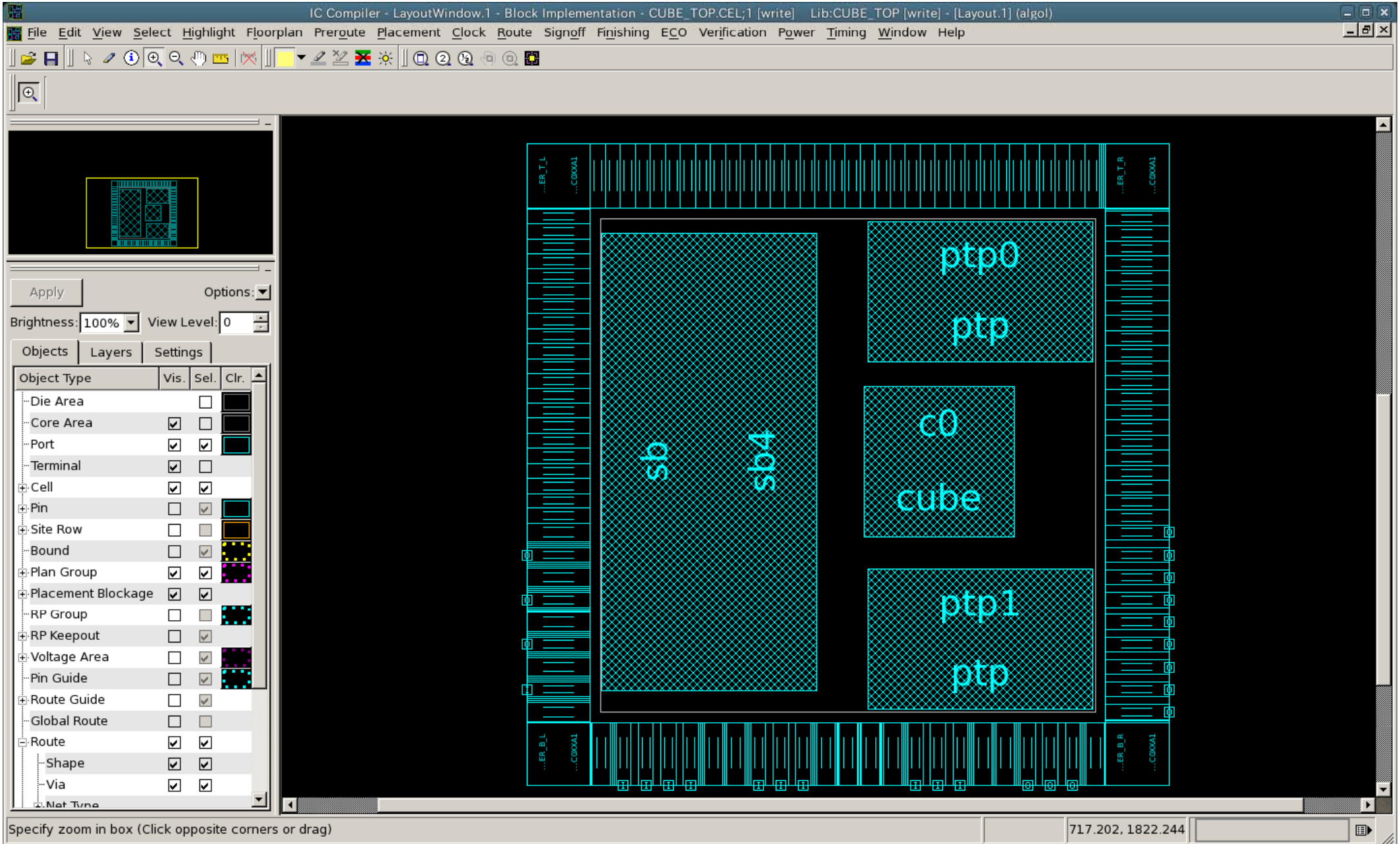
CUBE_TOP layout flow 2/8

Place macros (cube, ptp0, ptp1, and sb) on the chip using iogen.tcl and ioplace.tcl.



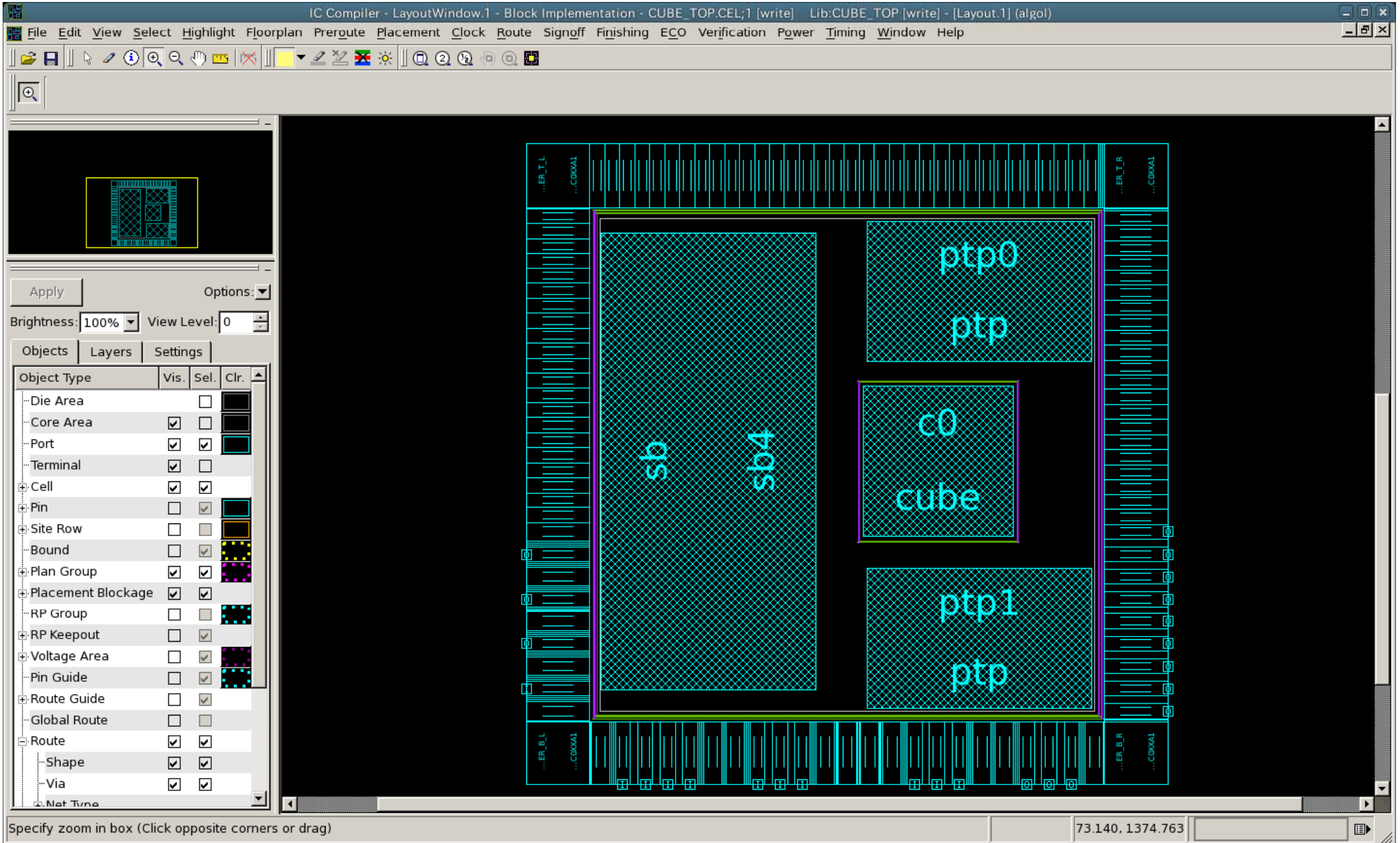
CUBE_TOP layout flow 3/8

Fill up unused I/O area with filler cells by insert_pad_filler command.



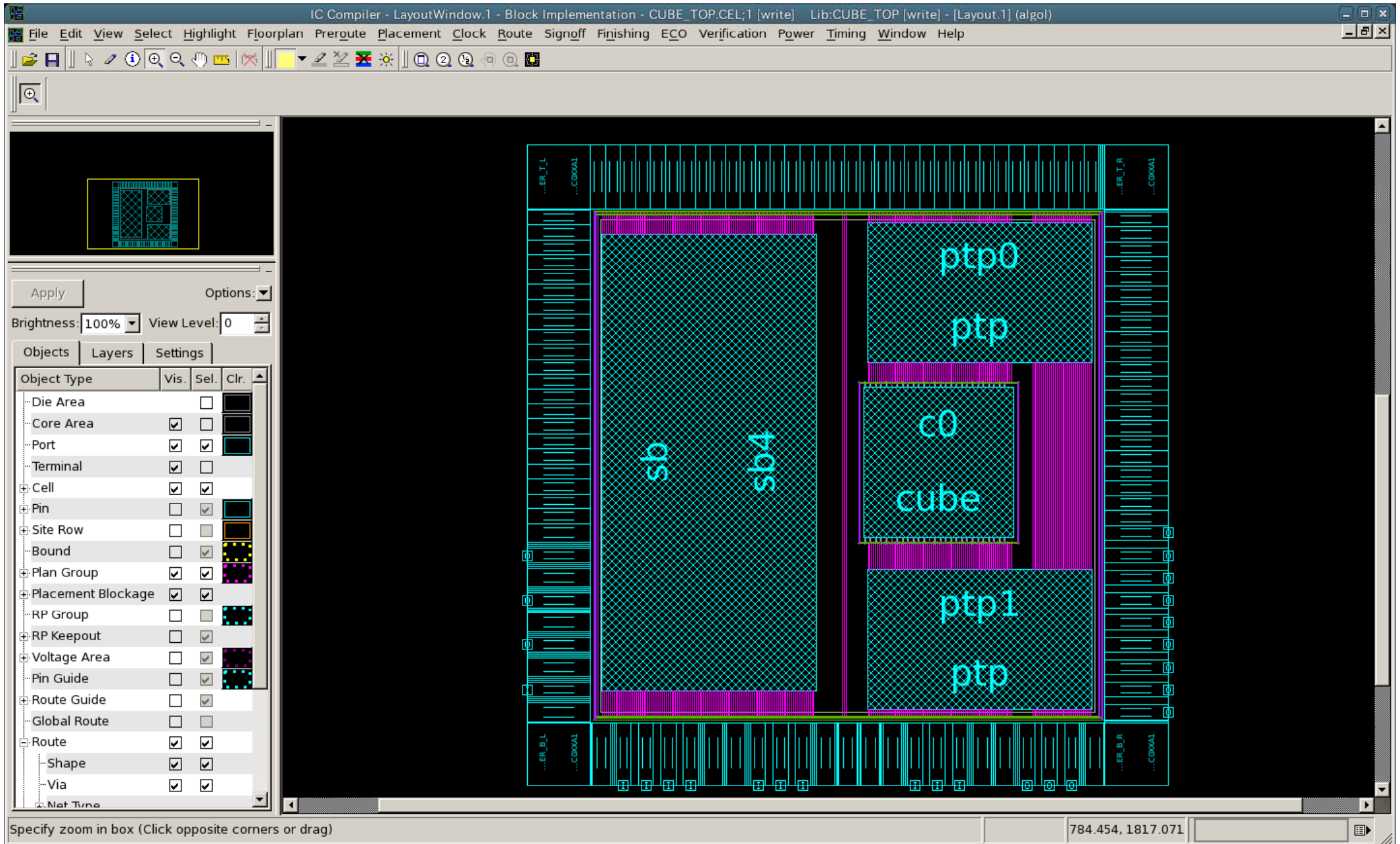
CUBE_TOP layout flow 4/8

Create power/ground rings by create_rectangular_rings command.
We need four rings including VDD, VDDBC, VDDBR, and VSS.



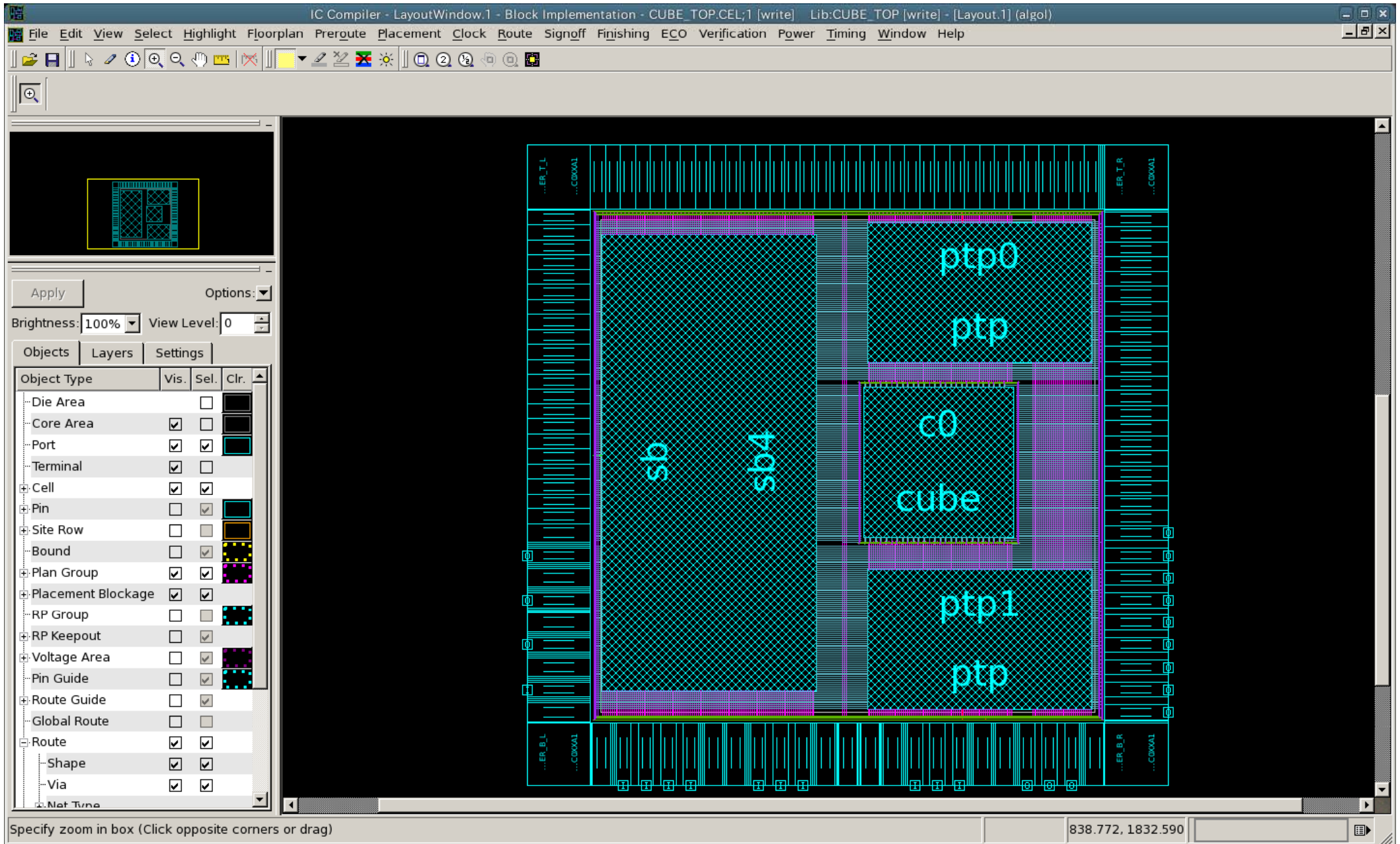
CUBE_TOP layout flow 5/8

Create vertical power straps by create_power_straps command.



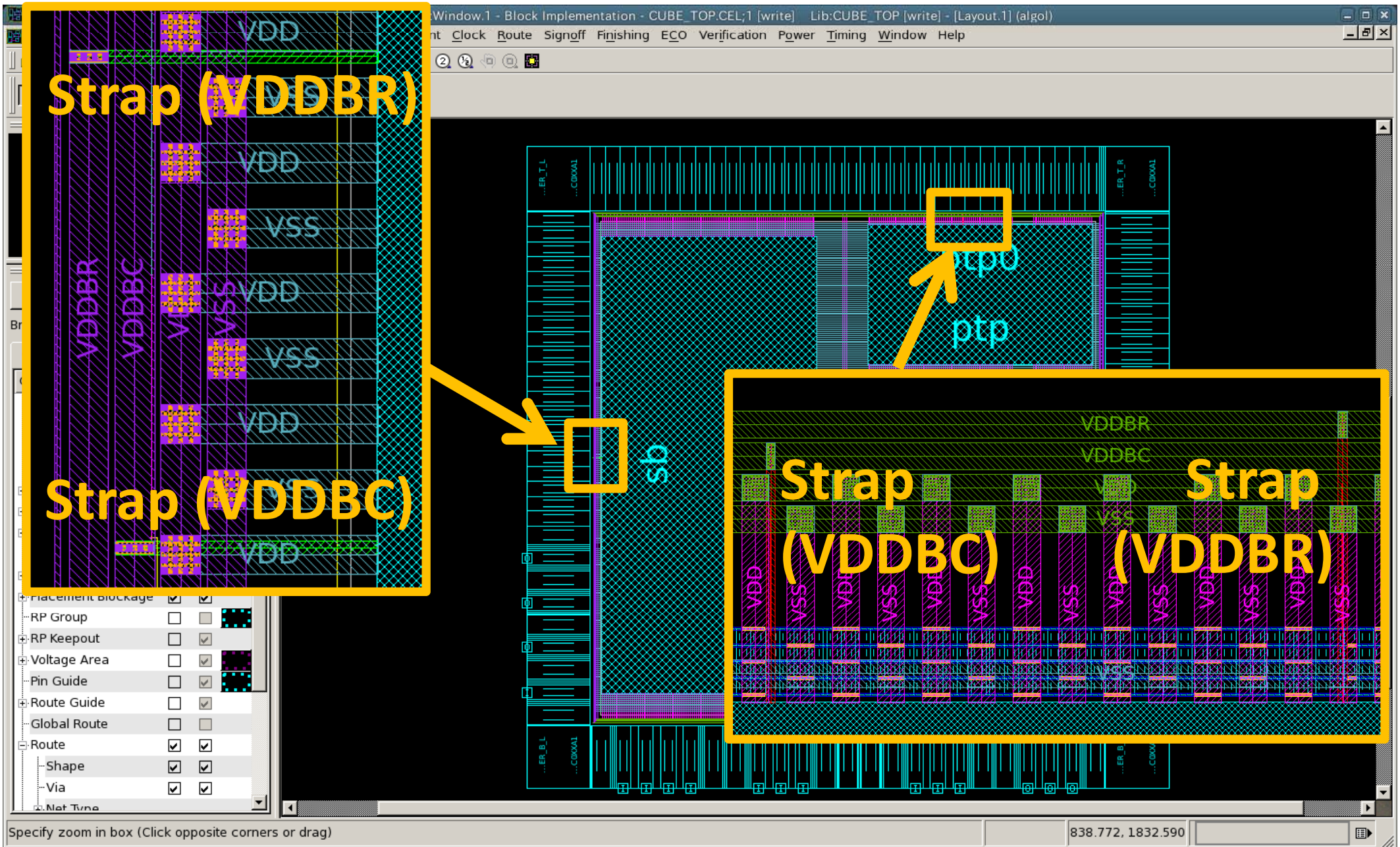
CUBE_TOP layout flow 6/8

Create horizontal power straps, because the inductors (ptp0, ptp1, and sb) require the horizontal power straps in addition to vertical ones.



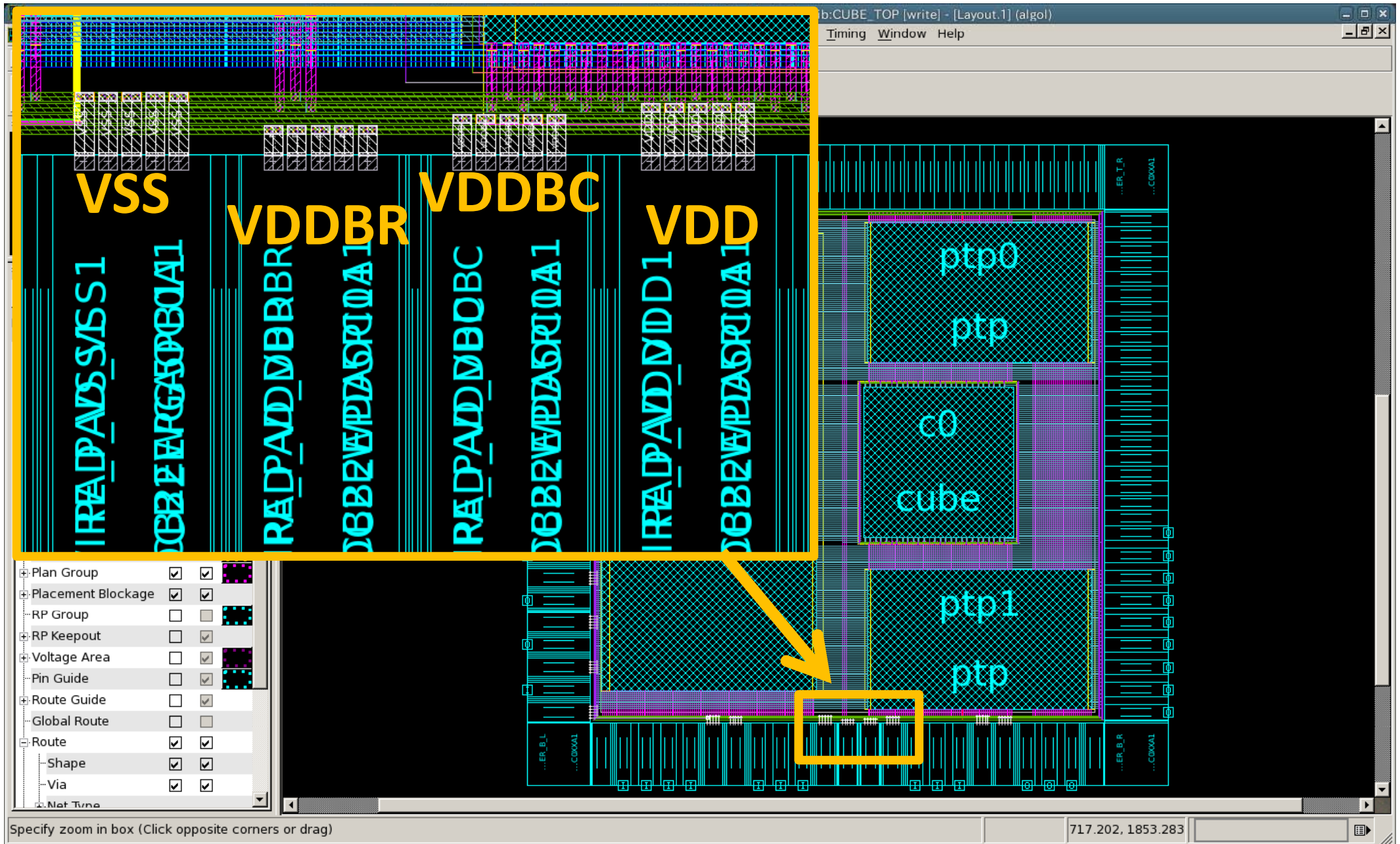
CUBE_TOP layout flow 7/8

Create straps between VDDBR/VDDBC rings and VDDBR/VDDBC ports of the inductors (ptp0, ptp1, and sb).



CUBE_TOP layout flow 8/8

Connect power/ground pads (VDD, VDDBR, VDDBC, and VSS) and corresponding power/ground rings by preroute_instances command.



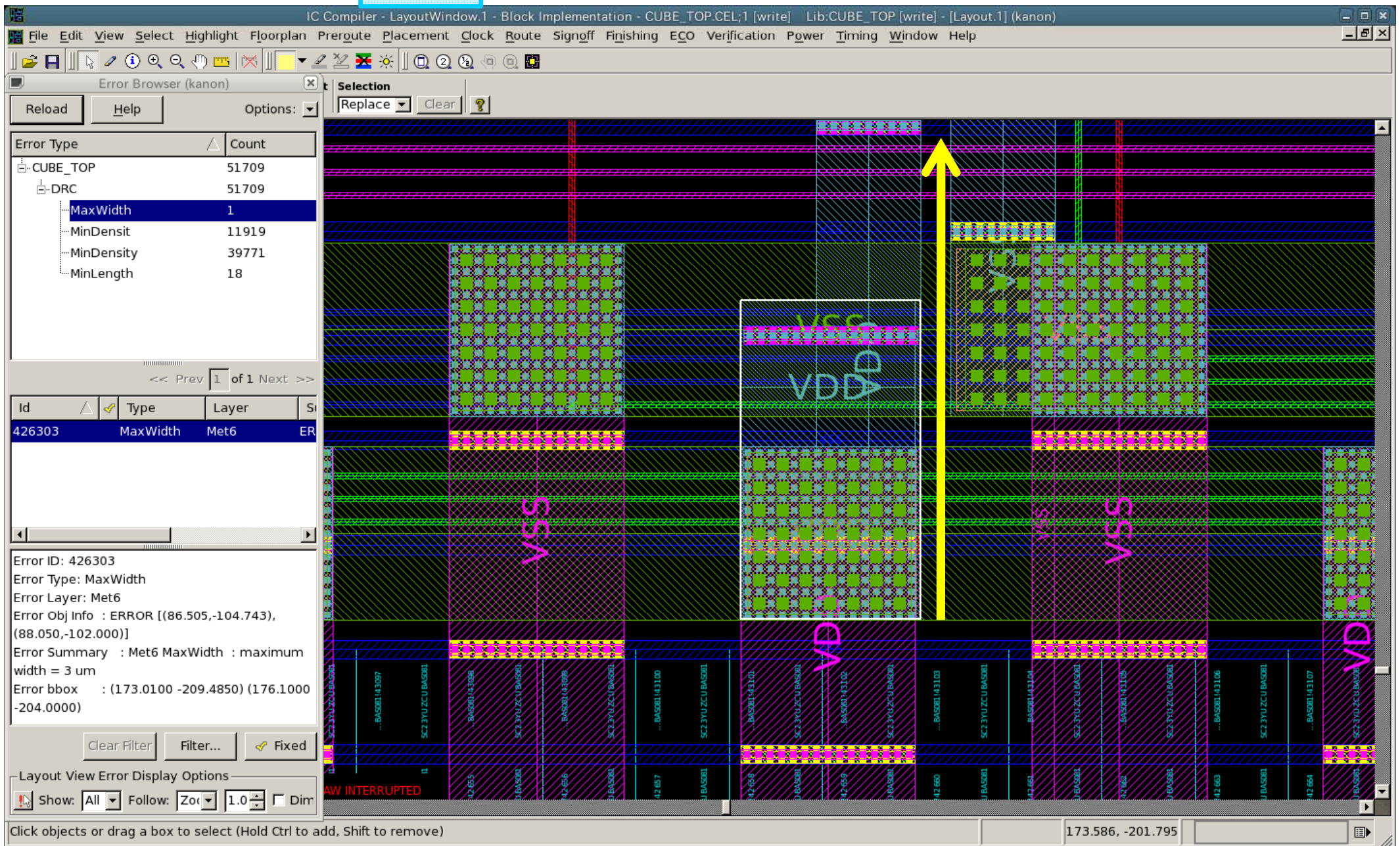
Appendix B:

CUBE_TOP DRC-free how-to

CUBE_TOP DRC-free how-to 1/4

The width of VDD area surrounded by white line should be reduced.

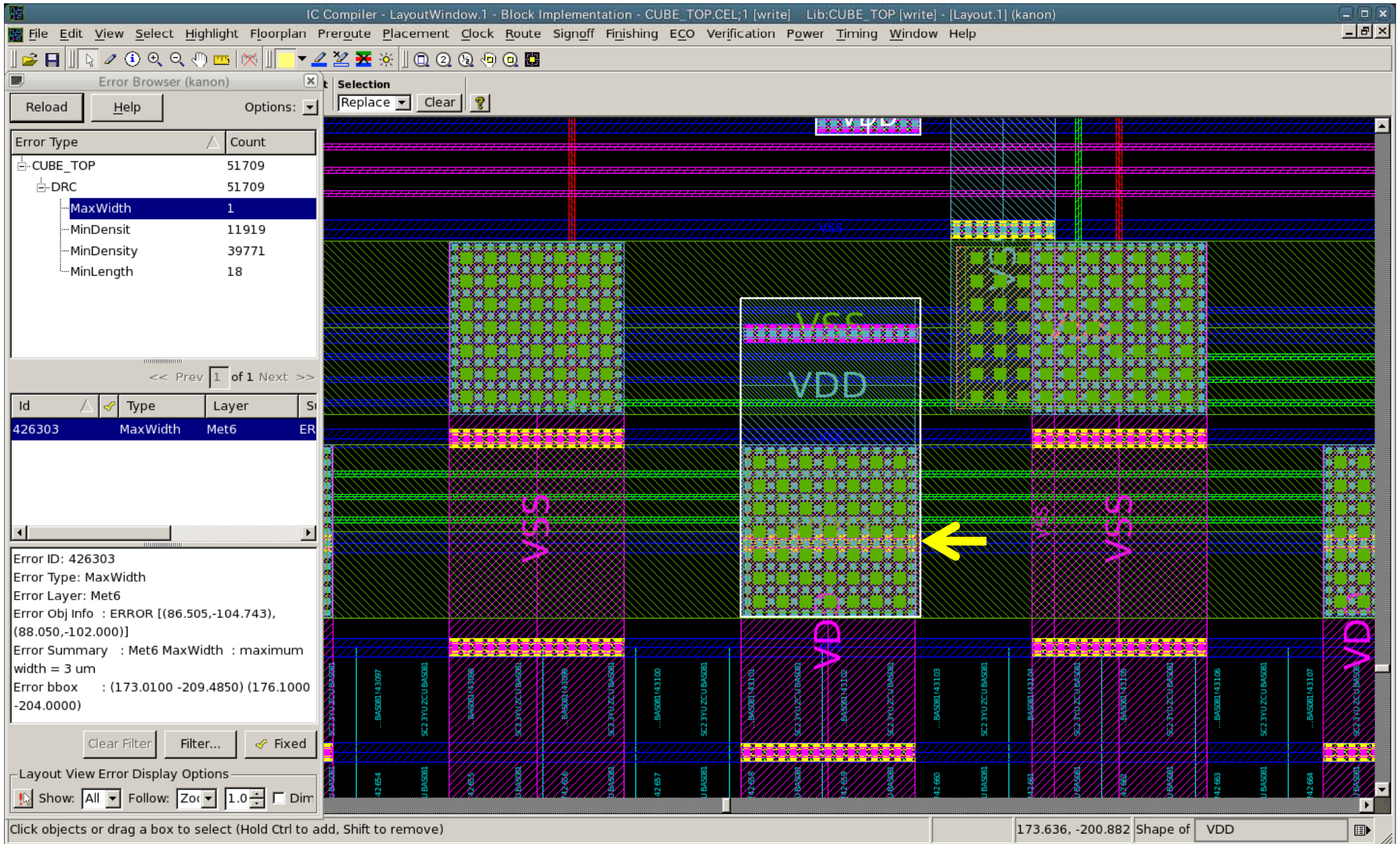
First, the height of VDD area is shortened (Resize it as the yellow arrow shows).



CUBE_TOP DRC-free how-to 2/4

The right edge of VDD area surrounded by white line should be moved to left.

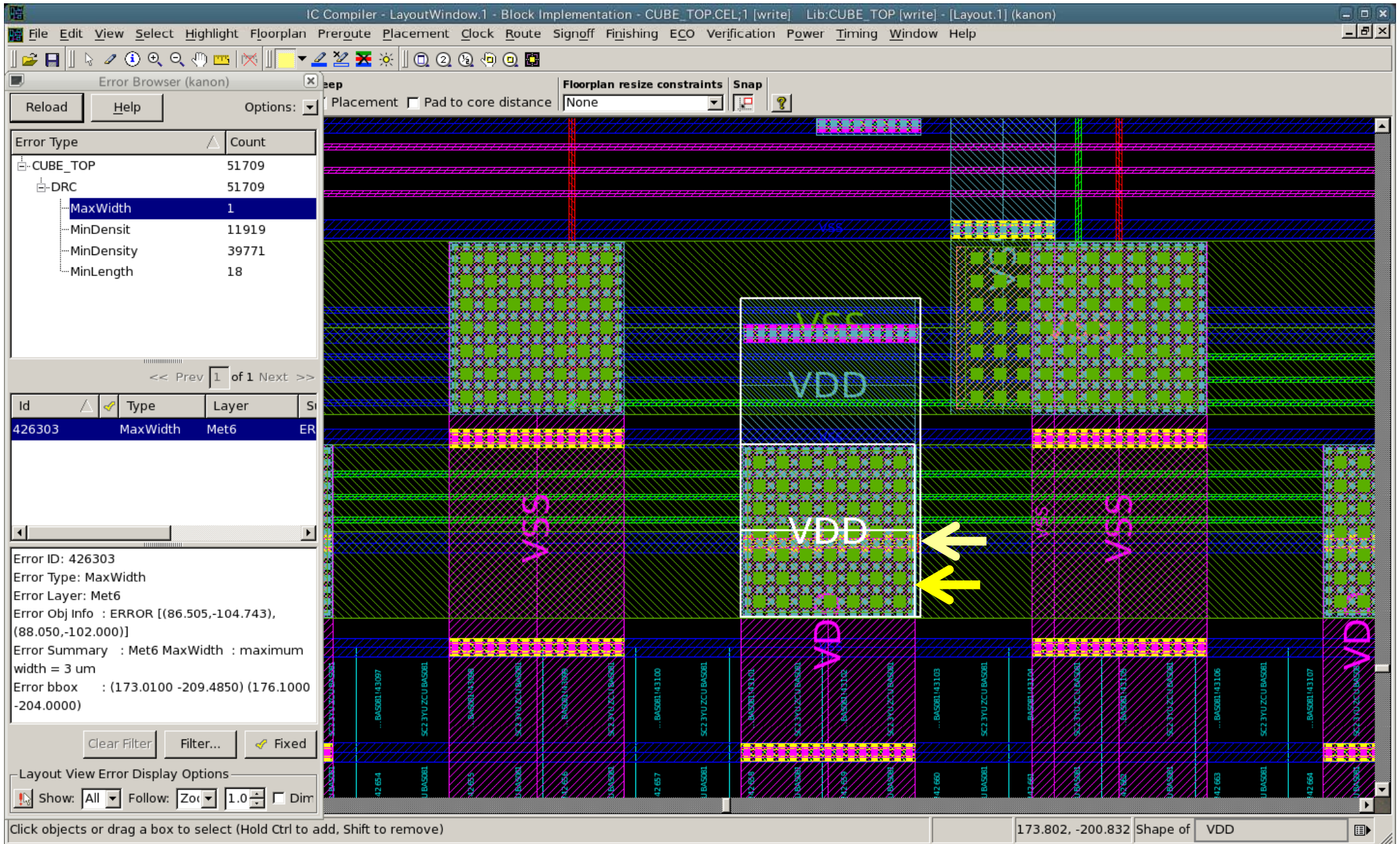
(Keep in mind the location of yellow arrow)



CUBE_TOP DRC-free how-to 3/4

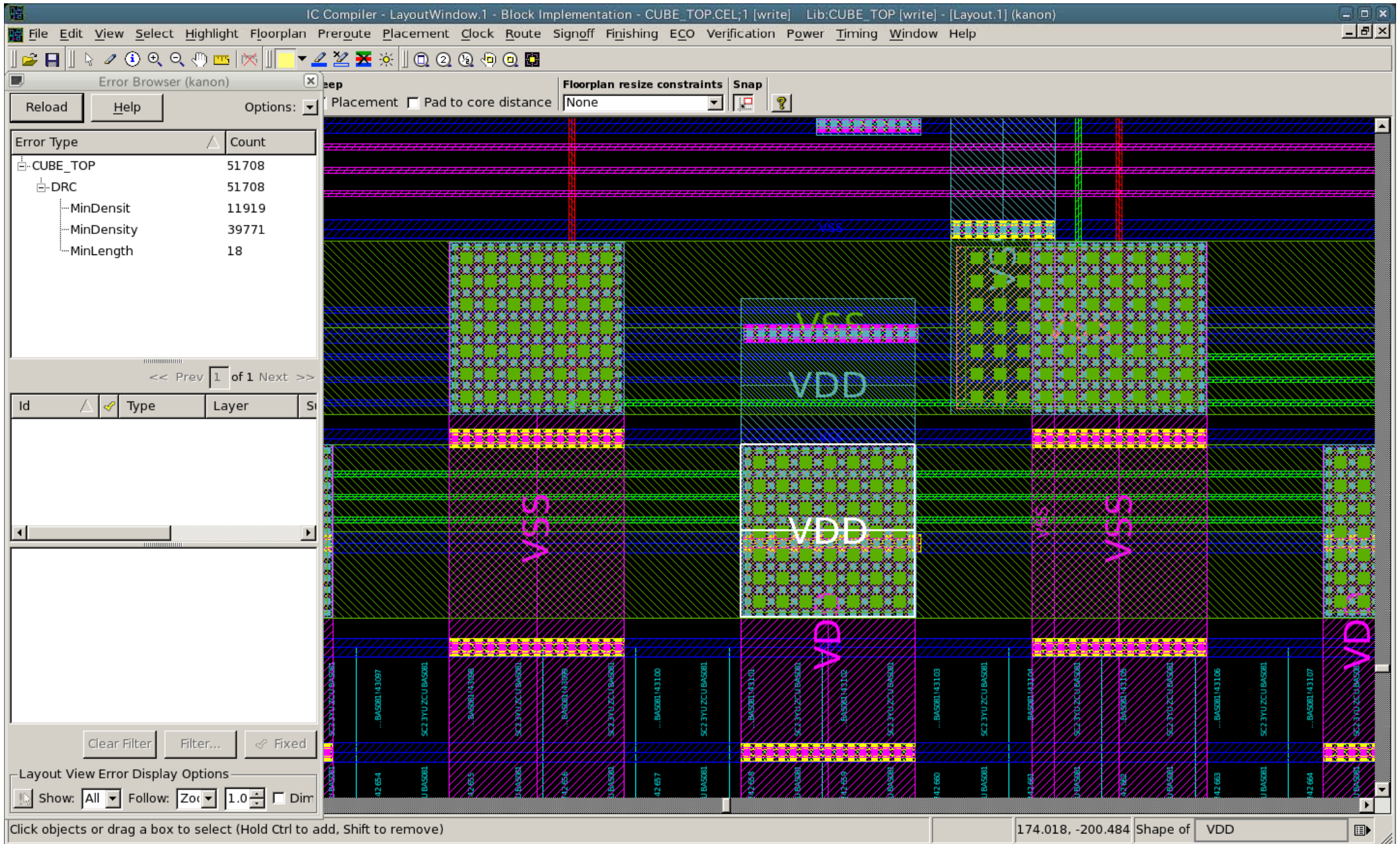
The right edge of the **VDD** area has been slightly moved to left!

(Compare the locations of new and old yellow arrows)



CUBE_TOP DRC-free how-to 4/4

Because the width of the **VDD** area is within an acceptable range, the next DRC doesn't report the "MaxWidth x1" error. Ignore the other errors here.

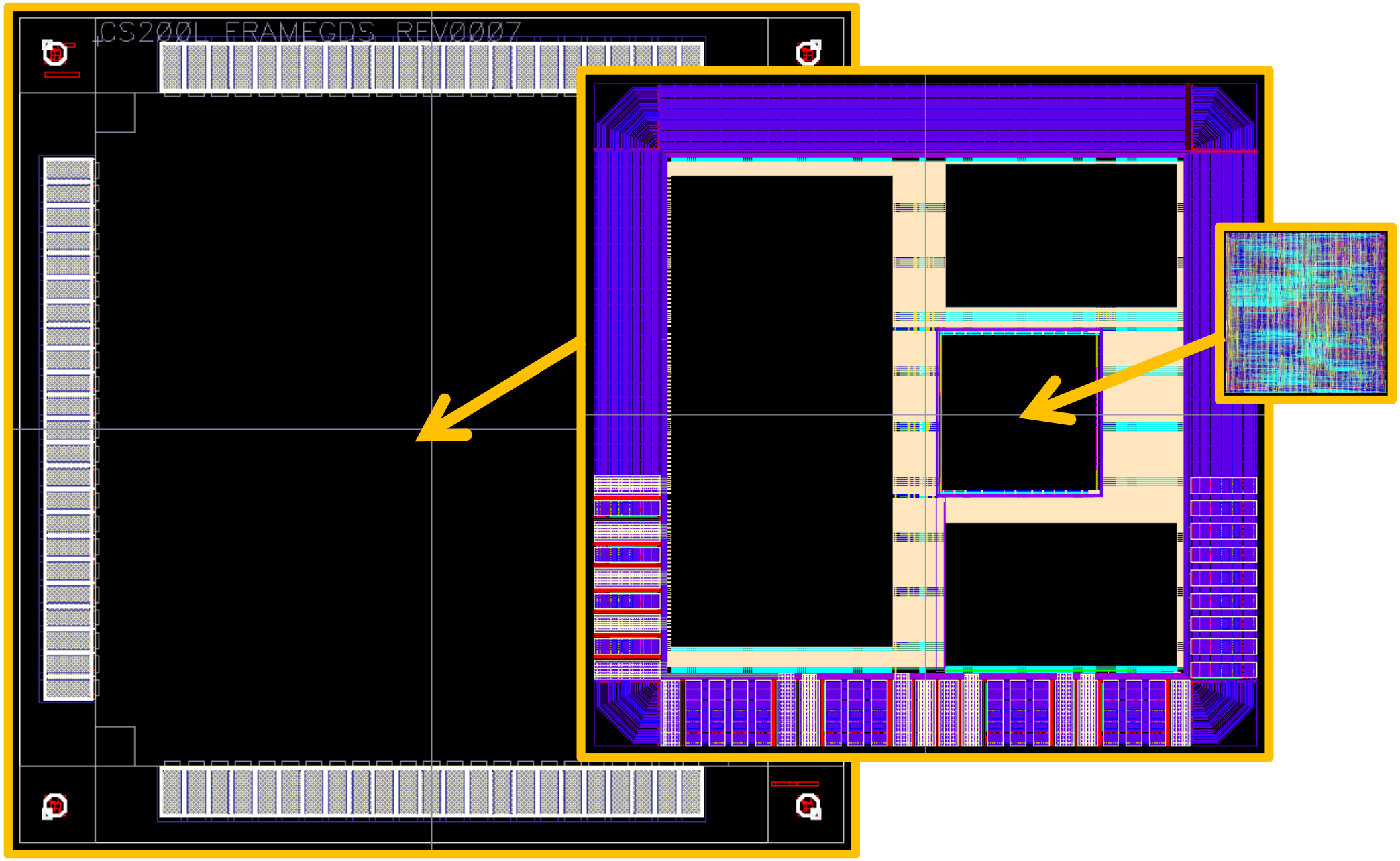


Appendix C:

CUBE chip frame insertion

CUBE chip frame insertion 1/2

Dummy inductors (ptp0, ptp1, and sb), cube core, and CUBE_TOP are embedded in the chip frame (E5FRAM). Note we use real inductors for the real tape-out.



CUBE chip frame insertion 2/2

Dummy inductors (ptp0, ptp1, and sb), cube core, and CUBE_TOP are embedded in the chip frame (E5FRAM). Note we use real inductors for the real tape-out.