

VLSI Testing and Fabrication Course Project

Semi-custom design of Newton Raphson Square Root Algorithm

**By- Anmol Agrawal 211010215
&
Aadi Juvekar 211010201**

AIM: To perform RTL to GDS flow of Newton Raphson square root algorithm.

Tools Used:-

- **Xilinx Vivado** => To test if the Verilog code and test bench are working properly, Xilinx Vivado software was used.
- **Cadence nclaunch** => We use this software to verify the functionality of the code. Xilinx was used as a test run, and nclaunch was finally used to verify the code.
- **Cadence genus** => This tool converts a high-level description of a digital circuit and generates an optimized netlist that meets certain design constraints. Genus might involve technology mapping, which is the process of mapping logical gates to specific cells in a target technology library.
- **Cadence innovus** => Innovus provides capabilities for physical implementation, including place and route (P&R), floorplanning, and global optimization.

THEORY:-

The Newton-Raphson square root algorithm is an iterative numerical method used to find the square root of a real number. In the context of VLSI (Very Large Scale Integration), which deals with designing and

manufacturing integrated circuits, the algorithm can be implemented efficiently in hardware to compute square roots. Here's a brief explanation of the theory behind the Newton-Raphson square root algorithm in the context of VLSI:

1. Algorithm Overview:

The Newton-Raphson square root algorithm is based on the iterative formula,

$$X_{n+1} = \frac{1}{2} \left(X_n + \frac{A}{X_n} \right)$$

where A is the input value, and X_n is the approximation of the square root after the n-th iteration.

The iterations continue until the approximation converges to the actual square root with the desired precision.

2. Initial Approximation:

The algorithm requires an initial guess for the square root, denoted as X_0 . This initial guess significantly influences the convergence speed and stability of the algorithm.

VLSI implementations may use fixed-point or floating-point representations for the input and intermediate values.

3. Hardware Implementation:

The iterative nature of the algorithm makes it well-suited for parallel and pipelined hardware implementations, which aligns with the parallel processing capabilities of VLSI circuits.

Dedicated hardware units, such as multipliers, adders, and dividers, are used to perform the arithmetic operations required in each iteration.

In summary, the Newton-Raphson square root algorithm can be implemented in VLSI by leveraging the parallel processing capabilities of the hardware and optimizing for precision, range, and efficiency. The hardware design should address issues such as convergence criteria, error analysis, and control logic to ensure a robust and reliable implementation.

VERILOG CODE :-

```
`timescale 10ps / 1ps

module newton (d,start,clk,clrn,q,busy,ready,count);
input [31:0] d; // radicand:
input start; // .1xxx...x
input clk, clrn; // or: .01xx...x
output [31:0] q;
output reg busy; // busy
output reg ready; // ready
output [2:0] count; // counter
reg [31:0] reg_d;
reg [33:0] reg_x;
reg [1:0] count;
//  $x_{i+1} = x_i * (3 - x_i * x_i * d) / 2$ 
wire [67:0] x_2 = reg_x * reg_x; // xxxx.xxxxxx...x
wire [67:0] x2d = reg_d * x_2[67:32]; // xxxx.xxxxxx...x
wire [33:0] b34 = 34'h300000000 - x2d[65:32]; // xx.xxxxxx...x
wire [67:0] x68 = reg_x * b34; // xxxx.xxxxxx...x
wire [65:0] d_x = reg_d * reg_x; // xx.xxxxxx...x
wire [7:0] x0 = rom(d[31:27]);
assign q = d_x[63:32] + |d_x[31:0]; // rounding up
always @ (posedge clk or negedge clrn) begin
if (!clrn) begin
busy <= 0;
ready <= 0;
end else begin
if (start) begin
reg_d <= d; // load d
reg_x <= {2'b1,x0,24'b0}; // 01.xxxx0...0
busy <= 1;
ready <= 0;
count <= 0;
end else begin
reg_x <= x68[66:33]; // /2
count <= count + 2'b1; // count++
if (count == 2'h2) begin // 3 iterations
busy <= 0;
ready <= 1; // q is ready
end
end
end
end
end
```

```

function [7:0] rom; // about 1/d ^ {1/2}
input [4:0] d;
case (d)
5'h08: rom = 8'hff; 5'h09: rom = 8'he1;
5'h0a: rom = 8'hc7; 5'h0b: rom = 8'hb1;
5'h0c: rom = 8'h9e; 5'h0d: rom = 8'h9e;
5'h0e: rom = 8'h7f; 5'h0f: rom = 8'h72;
5'h10: rom = 8'h66; 5'h11: rom = 8'h5b;
5'h12: rom = 8'h51; 5'h13: rom = 8'h48;
5'h14: rom = 8'h3f; 5'h15: rom = 8'h37;
5'h16: rom = 8'h30; 5'h17: rom = 8'h29;
5'h18: rom = 8'h23; 5'h19: rom = 8'h1d;
5'h1a: rom = 8'h17; 5'h1b: rom = 8'h12;
5'h1c: rom = 8'h0d; 5'h1d: rom = 8'h08;
5'h1e: rom = 8'h04; 5'h1f: rom = 8'h00;
default: rom = 8'hff; // 0 - 7: not used
endcase
endfunction
endmodule

```

TEST BENCH

```

`timescale 10ps / 1ps

module newton_tb();
reg [31:0] d; // radicand:
reg start; // .1xxx...x
reg clk, clrn; // or: .01xx...x
wire [31:0] q;
wire busy; // busy
wire ready; // ready
wire [2:0] count;
newton dut(d,start,clk,clrn,q,busy,ready,count);
initial begin
clk = 0;
forever #20 clk = ~clk;
end
initial begin
clrn = 0;
#50 clrn = 1;
end
initial begin
d = 0;
forever #45 d = ~d;
end

```

```

initial begin
start = 0;
#25 start = 1;
#50 start = 0;
end
endmodule

```

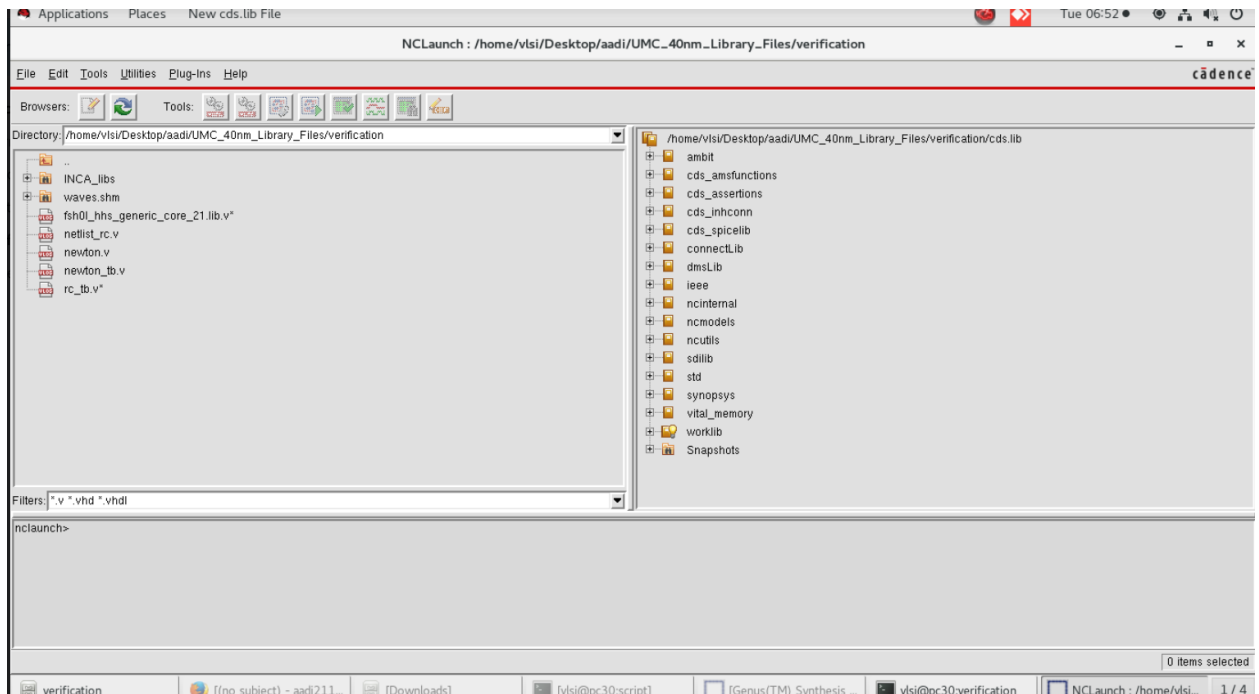
STEP 1 : CADENCE NC LAUNCH :-

nclaunch process for verification , test bench verify from

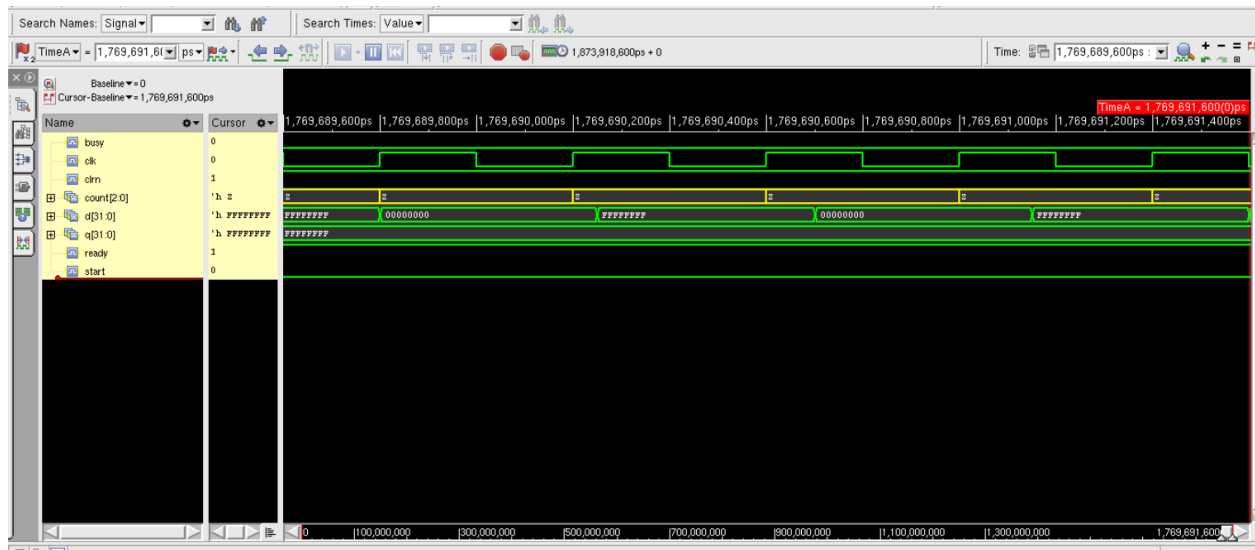
Snapshot > launch simulator with current selection

Select all on right panel > send to waveform window > play / run

1. Launch Cadence nclaunch in the folder with the Verilog and test bench file present.
2. Verify the newton.v and newton_tb.v files
3. Then, under worklib, find newton_tb.v file and verify that file again.
4. Under the snapshot, you will find the file newly generated with the name of newton_tb.v and press *run simulation*.

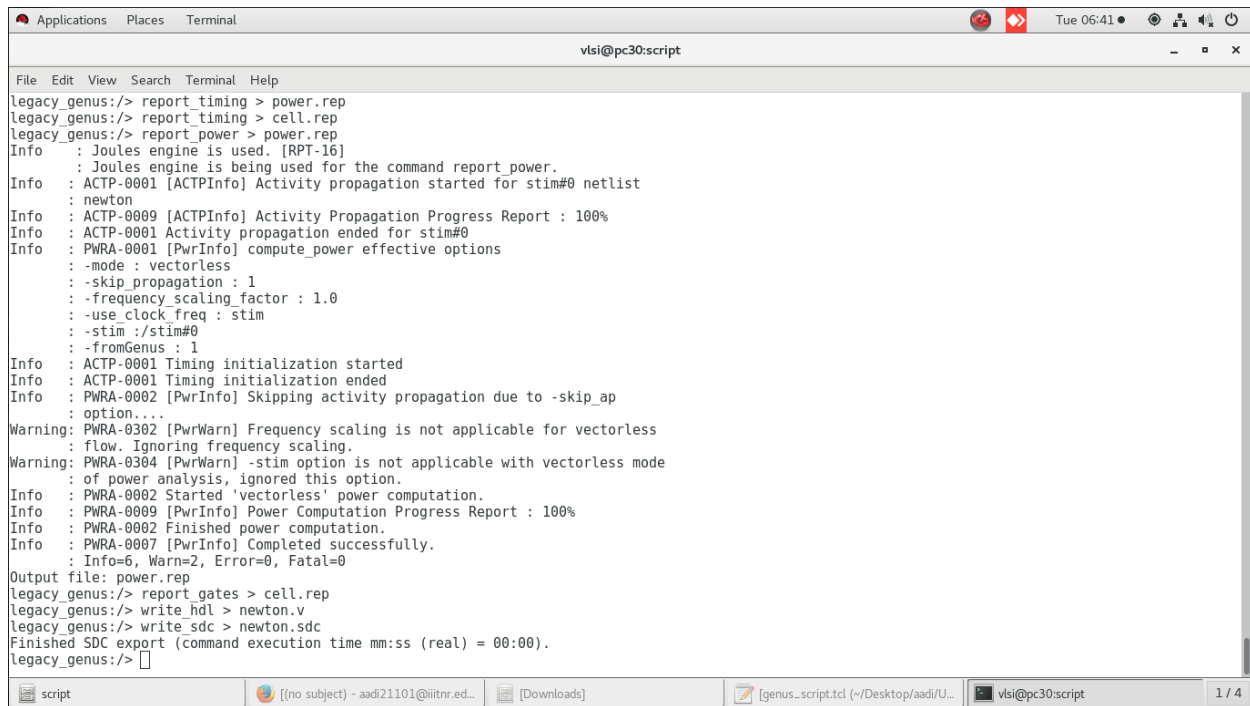


5. Under the new simulation window, select all variables/parameters and select *send to waveform window*.
6. Press Run/stop at will for visible verification.



STEP 2 : GENUS

1. First go the script folder in the foundry files then search a file with extension .tcl.
2. Open it and replace our verilog file in the destination specified.
3. Then open genus in terminal with genus -legacy_ui
4. Write the following commands in genus terminal:-
 - source genus_script.tcl
 - syn_gen
 - syn_map
 - syn_opt
 - report_timing
 - report_timing > time.rep
 - report_power > power.rep
 - report_gates > cell.rep
 - write_hdl > netlist_netwon.v
 - write_sdc > netlist_newton.sdc



```
File Edit View Search Terminal Help
vlsi@pc30:script
legacy_genus:/> report_timing > power.rep
legacy_genus:/> report_timing > cell.rep
legacy_genus:/> report_power > power.rep
Info : Joules engine is used. [RPT-16]
Info : Joules engine is being used for the command report_power.
Info : ACTP-0001 [ACTPInfo] Activity propagation started for stim#0 netlist
Info : newton
Info : ACTP-0009 [ACTPInfo] Activity Propagation Progress Report : 100%
Info : ACTP-0001 Activity propagation ended for stim#0
Info : PwRA-0001 [PwrInfo] compute_power effective options
Info : -mode : vectorless
Info : -skip_propagation : 1
Info : -frequency_scaling_factor : 1.0
Info : -use_clock_freq : stim
Info : -stim :/stim#0
Info : -fromGenus : 1
Info : ACTP-0001 Timing initialization started
Info : ACTP-0001 Timing initialization ended
Info : PwRA-0002 [PwrInfo] Skipping activity propagation due to -skip_ap
Info : option....
Warning: PwRA-0302 [PwrWarn] Frequency scaling is not applicable for vectorless
Info : flow. Ignoring frequency scaling.
Warning: PwRA-0304 [PwrWarn] -stim option is not applicable with vectorless mode
Info : of power analysis, ignored this option.
Info : PwRA-0002 Started 'vectorless' power computation.
Info : PwRA-0009 [PwrInfo] Power Computation Progress Report : 100%
Info : PwRA-0002 Finished power computation.
Info : PwRA-0007 [PwrInfo] Completed successfully.
Info : Info=6, Warn=2, Error=0, Fatal=0
Output file: power.rep
legacy_genus:/> report_gates > cell.rep
legacy_genus:/> write_hdl > newton.v
legacy_genus:/> write_sdc > newton.sdc
Finished SDC export (command execution time mm:ss (real) = 00:00).
legacy_genus:/> []
```

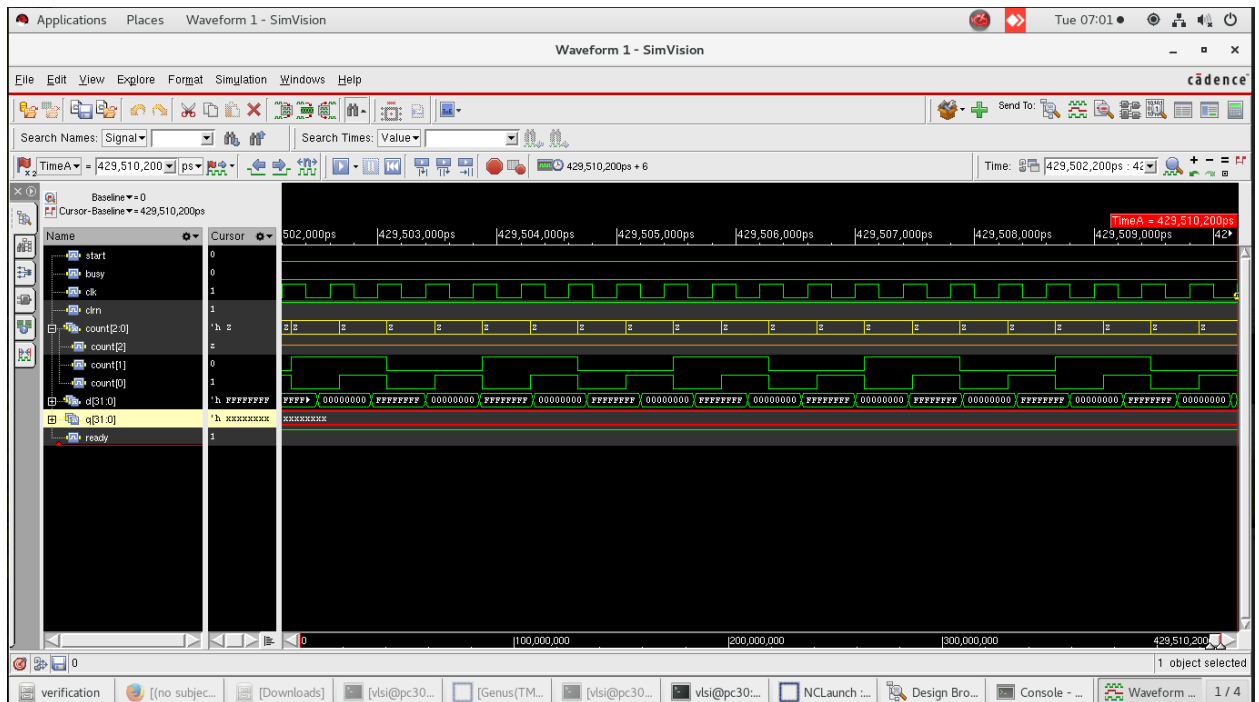
5. Close the genus terminal

STEP 3 : VERIFICATION

1. Create a new folder named “verification”, in *UMC_40nm*.
2. Go to the libs folder under that you will find simulation folder in which there will be two files with .v and .sdc extension.
3. Copy them and paste them in the verification folder.
4. Change the extension of .sdc file to .v file.
5. Opne nclaunch.

NCLAUNCH

1. Verify the two pasted files, “netlist_newton.v” and “newton_tb.v” similarly as mentioned earlier with nclaunch.



STEP 4 : INNOVOUS

1. Follow the following steps-

File > import design > verilog file > click the file then click on add

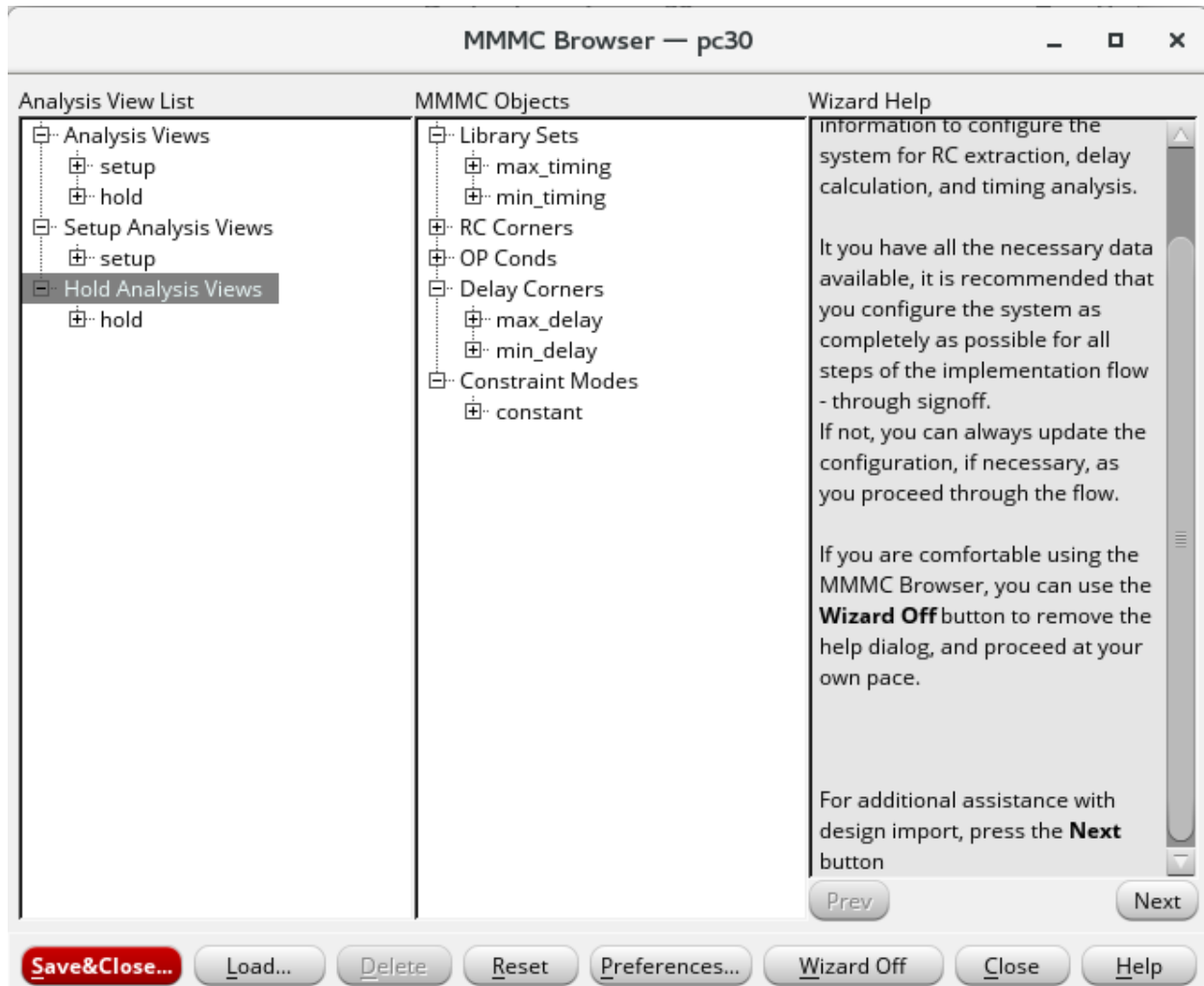
LEF Files : in libs/backend lib/

header -> fsh0l -> FSH0L

{ Power nets : VDD

Grounds nets : GND

MMC Obj : Library Sets : max_timing}



Add delay corner -> lib set : max timing
Constraint file : sdc file

Analysis views : setup : max_delay
hold : min_delay
Save innovus : innovus.view
innovus terminal : checkDesign -all

Floor planning : FloorPlan -> specify floorplan h/w : 0.7
10

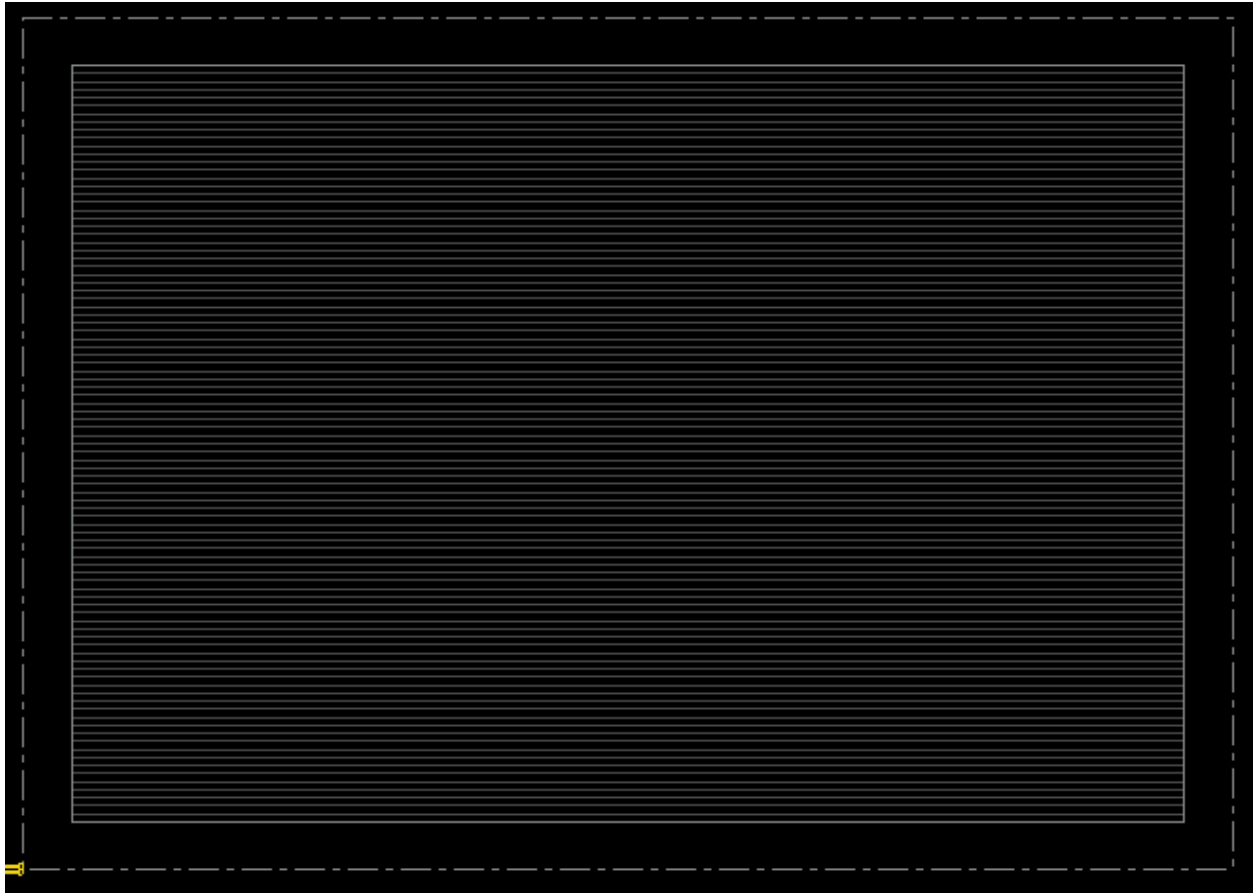
Power planning : add ring : nets browse : VDD GND { ME7 ME7
ME6 ME6 } OFFSET ENABLE

Power : add stripe : horizontal ME7 : NETS ✓

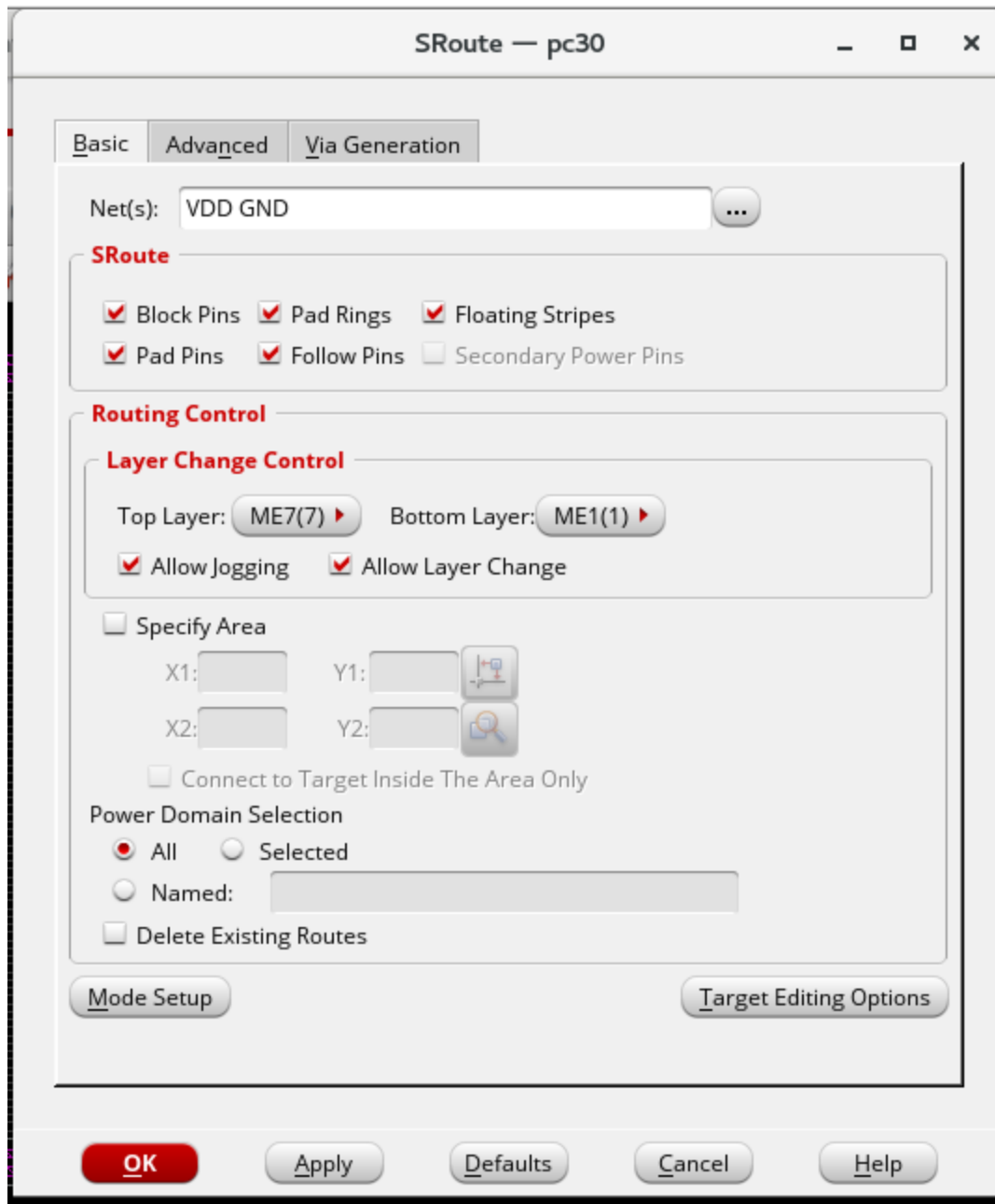
Number of sets : 3

Power : add stripe : vertical ME6 : NETS ✓

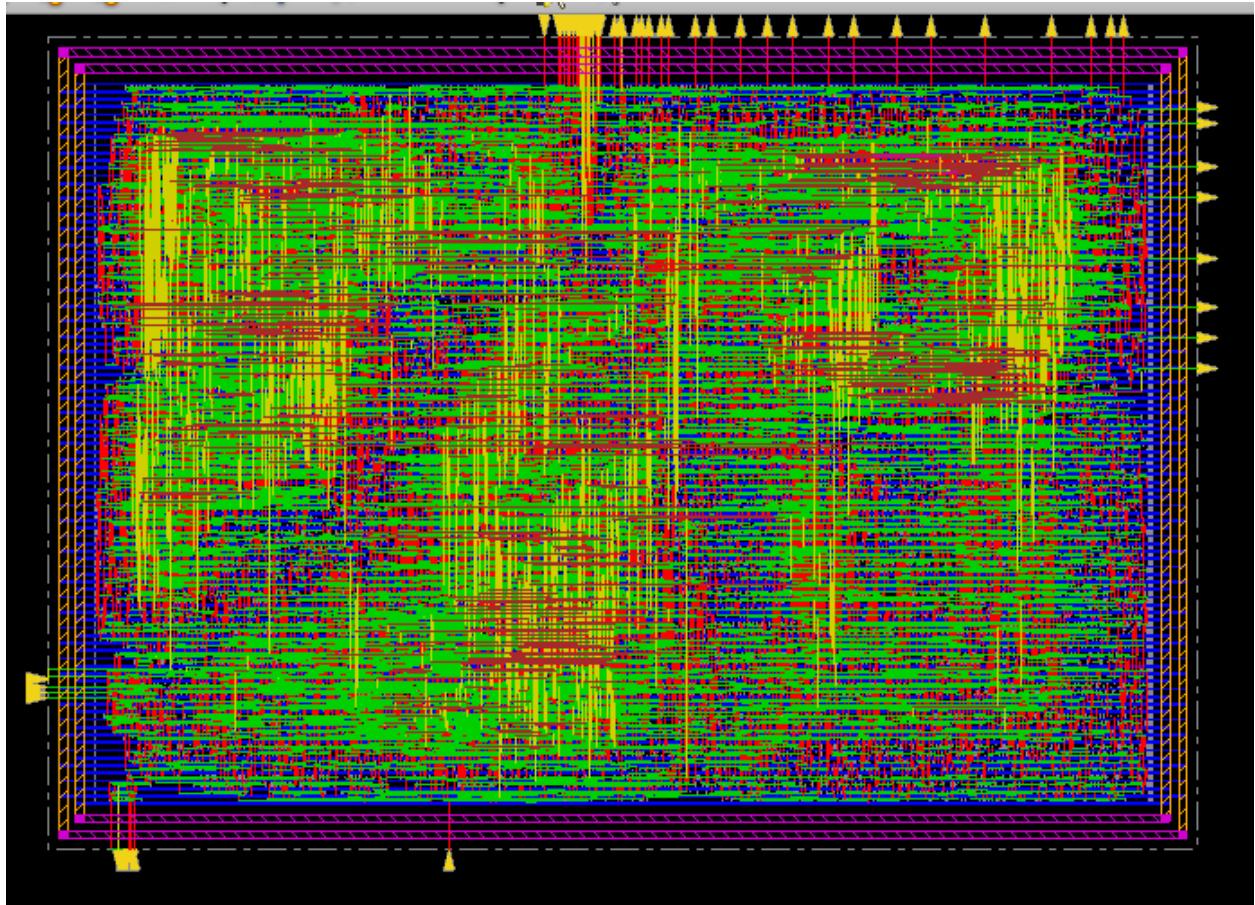
Save file : power_plan.emc



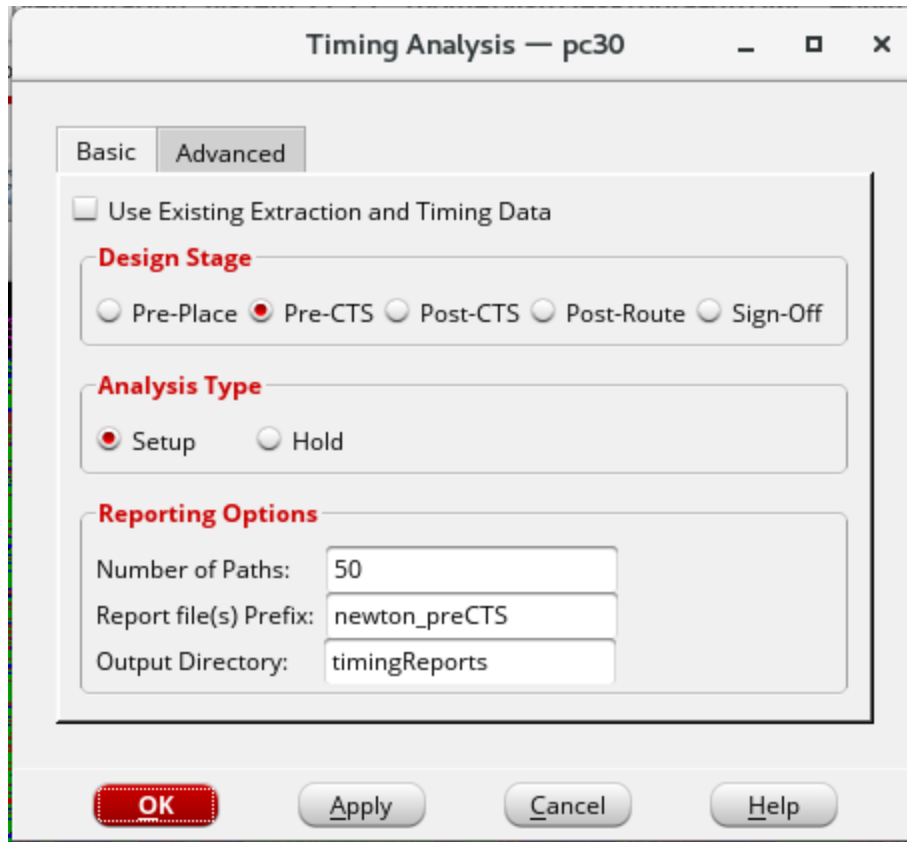
Route : special route : SRoute , complete tick ✓



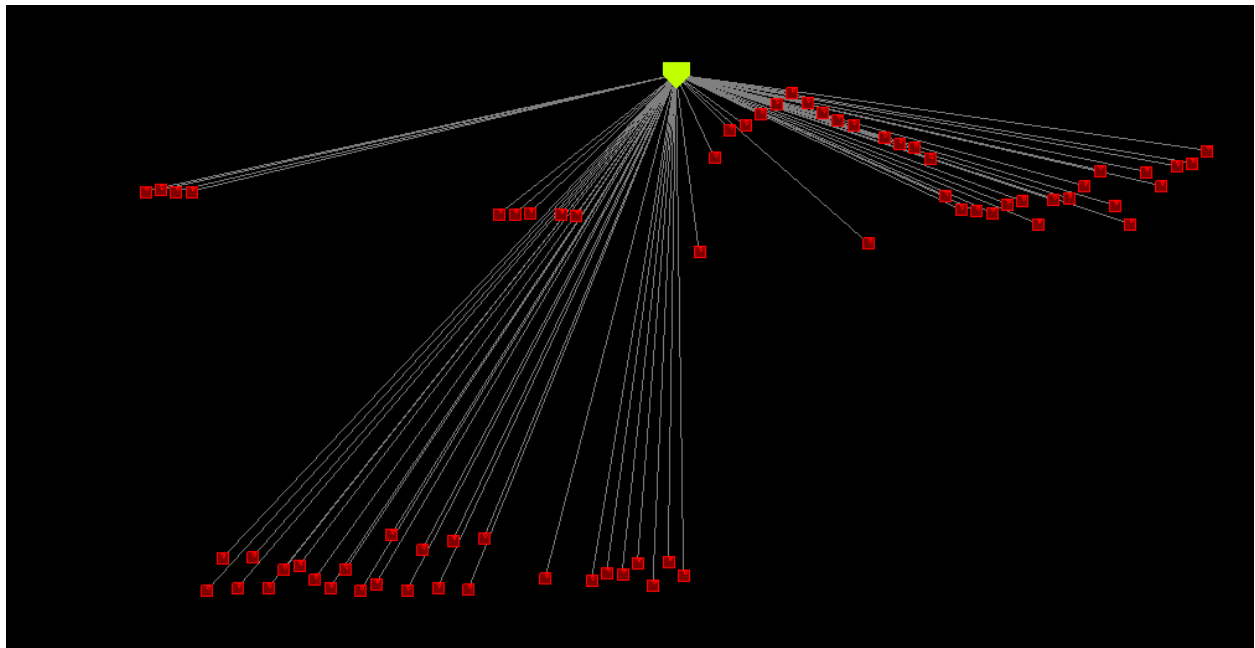
Place : phy cells _> add end cap : select FILLERD2LHHM
 Place : phy cells _> add well tap : select FILLERD2LHHM , 40
 Place : standard cell : run full placement
 Place : standard cell : more : place i/o pins : ok



Timing : report timing : pre-CTS , setup



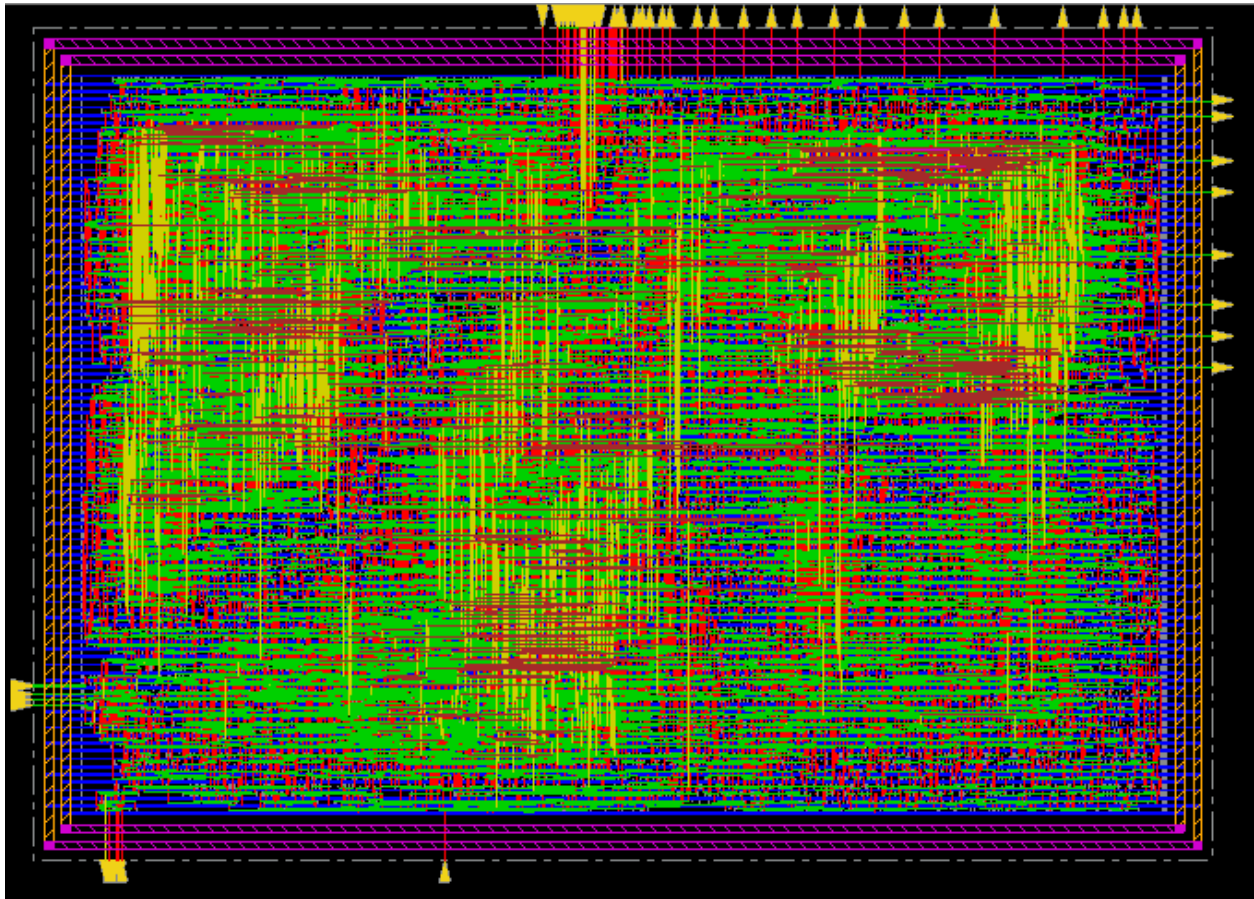
Clock : clock tree debugger



Timing : report timing : post-CTS , hold

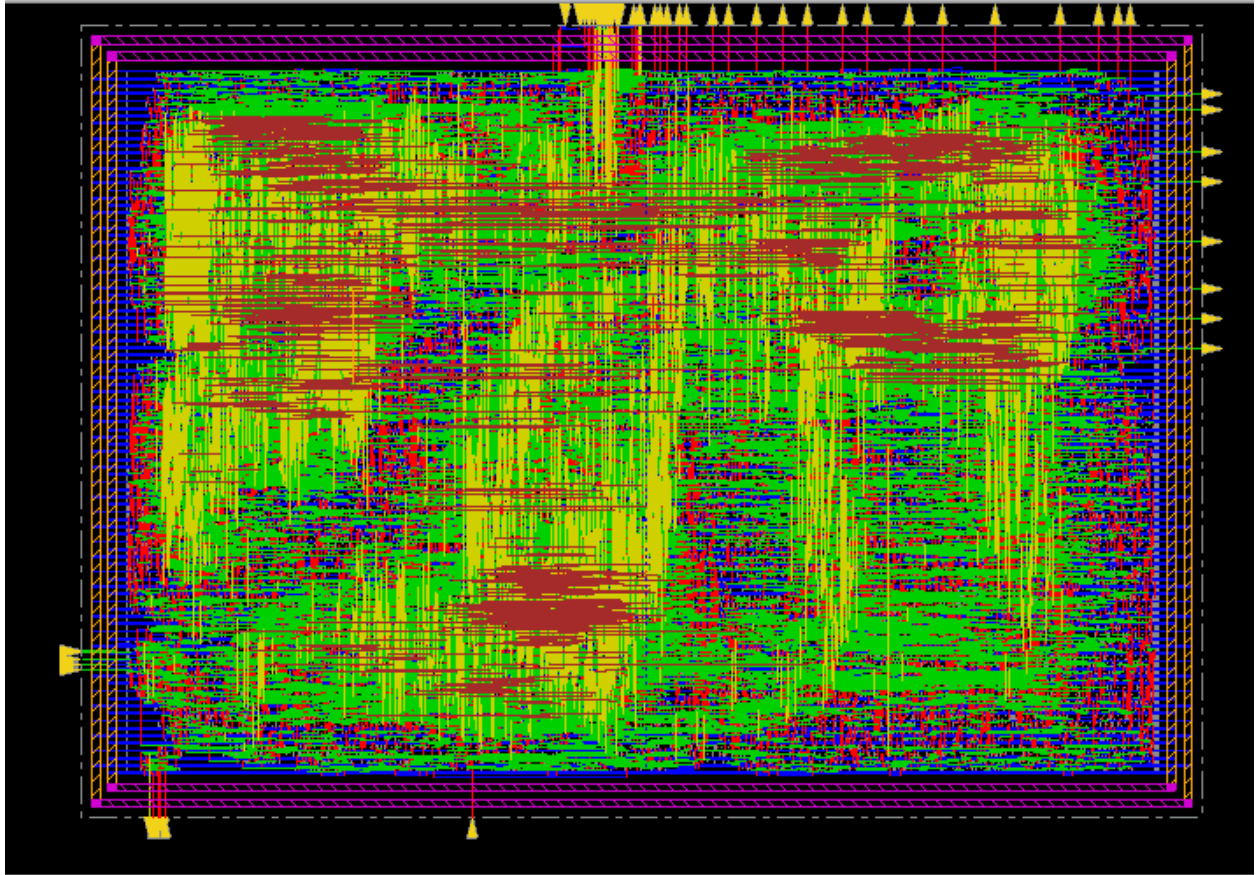
ECO : optimise design : post cts , remove setup ,select hold ok

Route : nanoroute : route : timing driven :

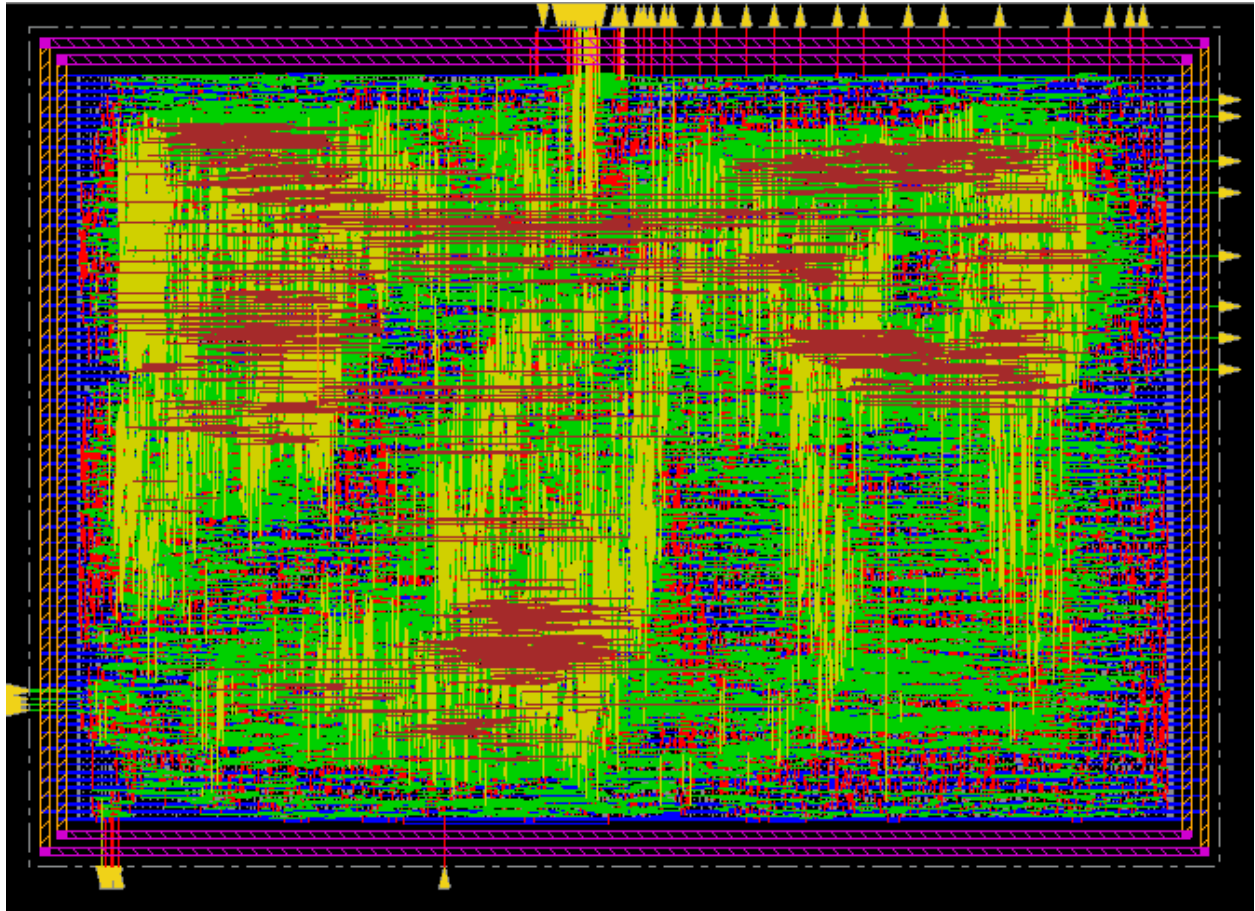


Place : add filler : cell name : select all : ok

Timing : report timing : post route : hold : ok



File : Save : gds : name AadiAnmol



TIMING REPORT:

```
=====
Generated by:      Genus(TM) Synthesis Solution 21.14-s082_1
Generated on:      Nov 28 2023 06:39:13 am
Module:           newton
Technology library: fsh0l_hhs_generic_core_ss1p1v125c 2018Q2v1.1
Operating conditions: _nominal_ (balanced_tree)
Wireload mode:    top
Area mode:        timing library
=====
```



```

g5575_5157_0          QAN2B2LHMMX6      1 2.0 27 402 9998 11
reg_x_reg[33]/D        QDFFNLHHMX6          +0 9998
reg_x_reg[33]/CK       setup          400 +2 10000 R
-----
(clock CLK)            capture          10000 R
-----
Timing slack :      0ps
Start-point : reg_x_reg[18]/CK
End-point   : reg_x_reg[33]/D

```

POWER REPORT

```

Instance: /newton
Power Unit: W
PDB Frames: /stim#0/frame#0
-----
Category      Leakage    Internal    Switching    Total    Row%
-----
memory        0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.00%
register       1.25720e-07 7.41669e-05 7.50023e-05 1.49295e-04 2.31%
latch         0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.00%
logic         7.87030e-06 2.74240e-03 3.57209e-03 6.32236e-03 97.63%
bbox          0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.00%
clock         0.000000e+00 0.000000e+00 4.06560e-06 4.06560e-06 0.06%
pad           0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.00%
pm            0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.00%
-----
Subtotal      7.99602e-06 2.81657e-03 3.65116e-03 6.47572e-03 100.00%
Percentage    0.12%    43.49%    56.38%    100.00% 100.00%
-----

```

CELL REPORT;

```
=====
Generated by:      Genus(TM) Synthesis Solution 21.14-s082_1
Generated on:      Nov 28 2023 06:40:24 am
Module:           newton
Technology library: fsh0l_hhs_generic_core_ss1p1v125c 2018Q2v1.1
Operating conditions: _nominal_ (balanced_tree)
Wireload mode:    top
Area mode:        timing library
=====
```

```
-----
total          10416 25637.270
```

Type	Instances	Area	Area %
sequential	70	439.824	1.7
inverter	1152	1167.533	4.6
buffer	46	69.149	0.3
logic	9148	23960.765	93.5
physical_cells	0	0.000	0.0
total	10416	25637.270	100.0