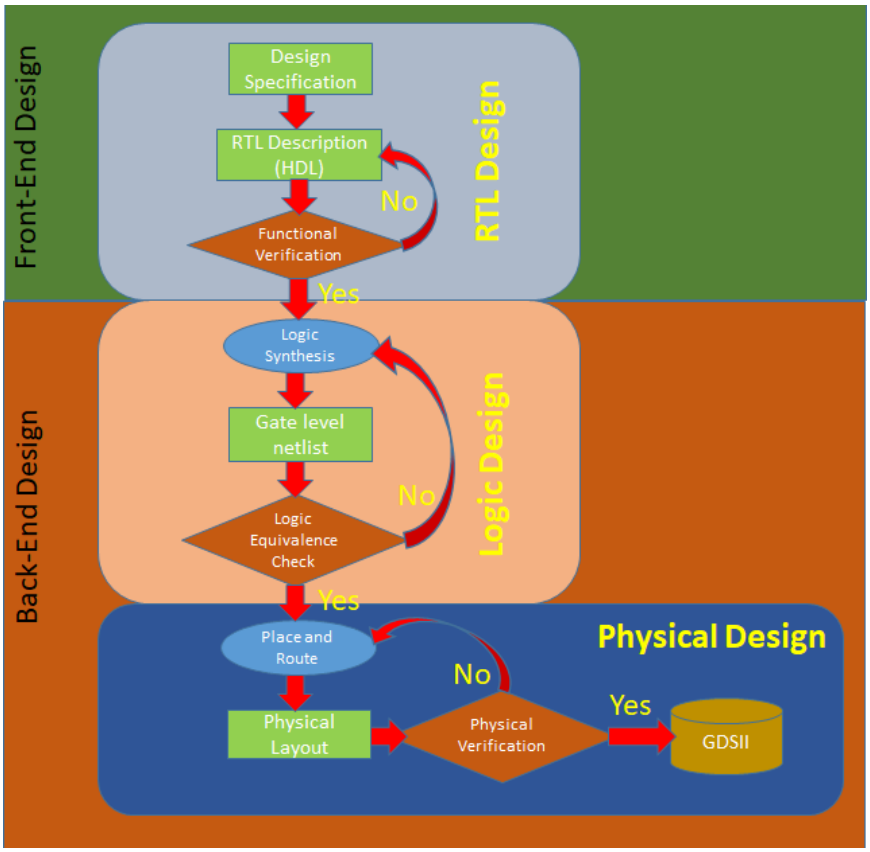
{System}Verilog

Getting started with building your own CPUs



ASIC Design Flow Overview with Synopsys Tools

Acknowledgements -

Abarajithan Gnaneswaran, Kithmin Wickremasinghe

Reference - ASIC Design Flow Tutorial by Varrie Duhaylungsod

Skill Surf

SYNOPSYS® Silicon to Software™

Apr 8, 2023 (v1.1)

Sec 1 - Overview of ASIC Flow

To design a chip, one needs to have an **Idea** about what exactly one wants to design. At every step in the ASIC flow the idea conceived keeps changing forms. The first step to make the idea into a chip is to come up with the **Specifications**.

Specifications are nothing but

- Goals and constraints of the design
- Functionality (what will the chip do)
- Performance figures like speed and power
- Technology constraints like size and space (physical dimensions)
- Fabrication technology and design techniques

A Simple ASIC Design Flow for the Course Context

As specified above, the very first step of ASIC flow is design specification, which usually comes from the customer end, where one writes down the specification of the chip, basically the functionality which they want to develop in a chip. The whole design process is going through various design cycles and it generally takes 6 to 24 months to complete the design depending on the complexity inside the chip.

The complete ASIC design process can be divided into two parts.

- Front End Design (covered in the course up to now)
- Back End Design (covered in this tutorial)

Front End Design:

Front end design process starts with the specification received from the customer end. The next step in the flow is to come up with the **Structural and Functional Description**. It means that at this point one has to decide what kind of architecture (structure) you would want to use for the design, e.g. RISC/CISC, ALU, pipelining etc. To make it easier to design a complex system; it is normally broken down into several subsystems. The functionality of these subsystems should match the specifications. At this point, the relationship between different subsystems and with the top level system is also defined. RTL (**Register Transfer Level**) design engineer converts the specification into an RTL code using the HDL (**Hardware Description Language**) generally either in Verilog or VHDL. Once the RTL code is written, RTL designer simulates the code in RTL Simulator and checks the functionality of the design. Once the functionality of code is correct and verified by the verification engineers and if there is no bug found, this RTL code is taken to the next stage which is **Logic Synthesis**.

Back End Design:

This flow starts with RTL coding and ends with a GDS (**Graphic Data Stream**) II file which is the final output of back end design, so this complete flow is also known as RTL to GDSII (**RTL2GDSII**) flow. RTL code received from the front-end engineer is technology independent, now the next step is **Logic Synthesis** and afterwards **PnR**.

- **Logic Synthesis:**

  In logic synthesis, a high-level description of the design (**RTL Code**) is converted into an optimized gate-level netlist or representation of a given standard cell library and certain design constraints. Now the code is in the form of a gate-level netlist of a particular standard cell library. LEC (**Logic Equivalence Check**) is required in this stage to make sure that there are no

logical changes occurring during the synthesis. During Logical Synthesis, we also get various reports on timing power and area of design. We also get an SDC (Synopsys Design Constraint) file in this stage which is used in the next stage. DFT (**Design For Testability**) Insertion is also done in this stage to verify the chip after fabrication is done. All this is done by Synthesis Tools such as **Design Compiler (Synopsys), Blast Create (Magma), RTL Compiler (Cadence)** etc.

- **Place and Route (PnR):**

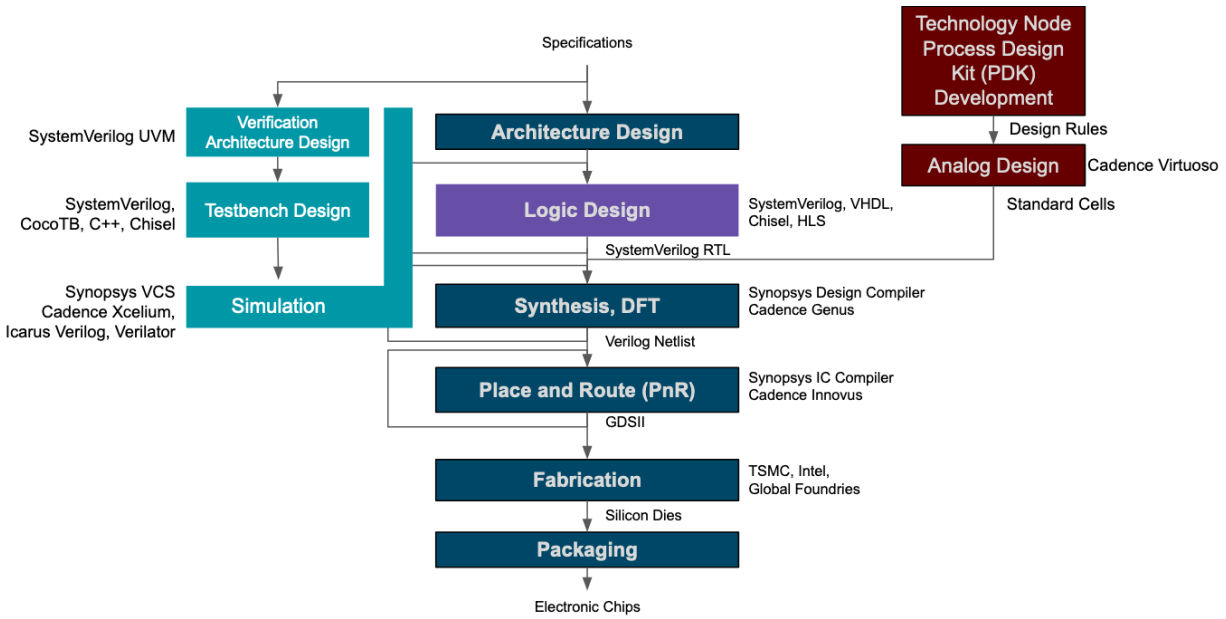
  The next step in the ASIC flow is the **Physical Implementation** of the Gate level Netlist. The Gate level Netlist is converted into geometric representation. The geometric design rules specified in the library. The design rules are nothing but guidelines based on the limitations of the fabrication process. Hence, Gate level Netlist after DFT Insertion and SDC file is taken as input for the PnR and based on standard cells library, PnR starts. The goal of PnR stage is to place all the standard cells, Macros and I/O pads with minimal area, with minimal delay and Route them together in such a way that there is no DRC (**Design Rule Check**) error. The final output of this stage is the layout of design in the form of **GDSII file which is the de facto standard of layout file** in the industry. This step is performed by tools such as **Blast Fusion (Magma), IC Compiler (Synopsys), and Encounter (Cadence)** etc.

  PnR stage is a very challenging stage with large design cycle time depending on the complexity of a chip. This stage is further divided into various sub-stages. The main stages are starting from **Design Import,** followed by **Floor Plan, Power Plan, Placement, CTS** (**Clock Tree Synthesis**), and **Routing**.

After routing we expect the design has met the timing and all DRC, But in the modern chip, it's not easy to close the design in this stage. So further we go to the **Sign Off** stage.

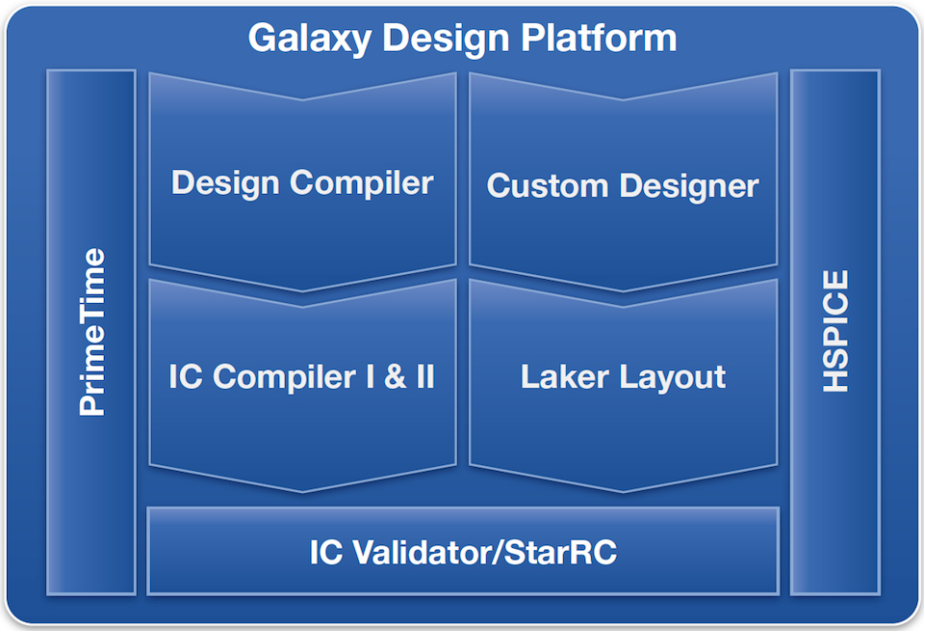
Sign Off:

If there are some timing violations in post route design, we have a further stage called ECO (**Engineering Change Order**) where we can fix the timing violations. Apart from timing violation, there may be issues like IR Drop, DRC Violations all these are fixed in this stage and a final layout file free from all the violation is streamed out in GDSII format. This process is known as tapeout in ASIC flow. This is the final design stage and a GDSII file is sent to the fabrication lab for the fabrication of the chip. After **fabrication**, the wafer is diced into individual chips. Each **Chip** is packaged and tested.



Synopsys Tools and SAED 32nm EDK

We introduce Synopsys RTL design toolkit, which are VCS, Design Compiler, IC Compiler II, VCS RTL Verification solution.



DC Ultra: Concurrent Timing, Area, Power, and Test Optimization

DC Ultra includes innovative topographical technology that enables a predictable flow resulting in faster time to results.Topographical technology provides timing and area prediction within 10% of the results seen post-layout enabling designers to reduce costly iterations between synthesis and physical implementation. DC Ultra also includes a scalable infrastructure that delivers 2X faster runtime on quad-core platforms.

IC Compiler II: Industry Leading Place and Route System

IC Compiler II is a complete netlist-to-GDSII implementation system that includes early design exploration and prototyping, detailed design planning, block implementation, chip assembly and sign-off driven design closure. The foundation, architecture and implementation is based on novel, patented technologies and the software has been written using modern object-oriented languages and tools.

Synopsys Tools Documentation:

- Design Compiler.pdf
- IC Compiler II.pdf

SAED32_EDK Documentation:

- SAED32.28nm_Digital_Standard_Cell_Library_b100_01312012.pdf
- SAED32.28nm_Device_Formation_Rev_1.0.2_b100_01312012.pdf
- SAED32.28nm_Design_Rules_Rev_1.0.2_b100_01312012.pdf
- SAED32.28nm_Spice_Models_Rev_1.0.2_b100_01312012.pdf

Useful Linux Commands

Here are some useful Linux and bash commands that might be helpful for all of you when navigating the directories and files while using the Synopsys tools for this tutorial.

```
-> In Linux, the most important thing is to navigate through directories.
-> Directories end with a "/" while files do not.

-> Each directory (similar to a folder) and file is described by a path.
-> Absolute paths start with "/" from the root directory.
-> Relative paths start with "./" from the current directory.
-> The two most important directories are the following:
 -> "." current directory
 -> ".." upper directory

-> Some commands have flags "-" next to them.
-> Use the TAB button to autocomplete commands, filenames, and paths.

--------------------------------------------------------------------------------
NAVIGATING THROUGH DIRECTORIES
--------------------------------------------------------------------------------

pwd
- Shows the current working directory

ls
- Displays the contents of the current directory
- It hides contents beginning with "."

ls -la
- Displays all the contents on a list format

cd <directory_path>
- Changes the directory to <directory_path>
- Similar to going to another folder
- For example, "cd .." returns to the upper directory

--------------------------------------------------------------------------------
MAKING, COPYING, AND REMOVING DIRECTORIES
--------------------------------------------------------------------------------

mkdir <directory_name>
- Creates a new directory with the name of <directory_name>
- You can create multiple directories at the same time, just keep adding the
paths of the new directories

cp -r <source1> <source2> <source3> ... <destination>
- Copies the directories <source1> <source2> <source3> to <destination>
```

- The "-r" flag is needed to recursively copy all of the contents inside the source directories
- You can copy multiple directories but the last path will always be the destination

rm -r <source1> <source2> <source3> ...
- Removes the directories <source1> <source2> <source3>
- The "-r" flag is needed to recursively remove all the contest inside the directories
- You can delete multiple directories at the same time

mv <source> <destination>
- Moves the directory <source> to <destination>
- Can also be used to rename the <source> directory

MAKING, COPYING, AND REMOVING FILES
---

touch <filename.extension>
- Creates a file of any extension

cp <source1> <source2> <source3> ... <destination>
- Copies multiple files to a single destination

rm <source1> <source2> <source3> ...
- Removes multiple files

mv <source> <destination>
- Moves files from the <source> to the <destination>
- Can be used to renamed the <source> file

cat <filename>
- Displays the contents of a file

source <filename>
- Runs the contents of a file

vim <filename>
- Text editor to edit files

---B
ASIC VIM COMMANDS
---

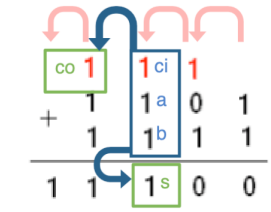
i
- Press "i" to enter "insert mode" where you can write and delete text in your file.
- A "-- INSERT --" message will appear.
- Use ESC to exit insert mode.

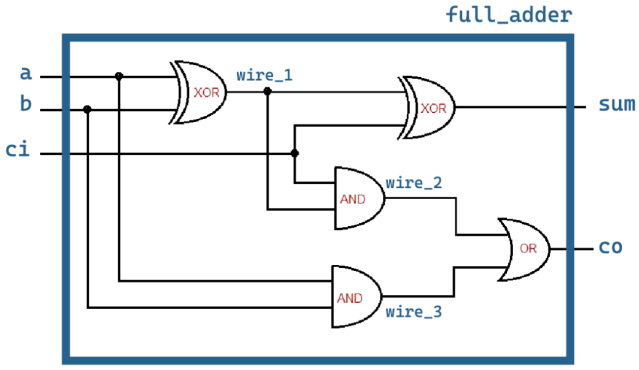
```
/<term>
- Highlights all the instances of <term> inside the file.
- Use "n" to go to the next instance and "SHIFT + N" to return to the previous
instance.

u
- Undoes work you have done on the file.

:set number
- Displays the line numbers at the left of the file.

:w
- Saves the file.

:q
- Quits and exists VIM.
- Only works if the file has been saved.
- Use ":q!" to quit and avoid saving changes.
```

Sec 2 - Step by Step approach to ASIC Flow

In this Tutorial, you will learn how to do RTL (register transfer level) design to build your circuit with HDL (hardware description language) for EDA (electronic design automation). We will take the full_adder circuit system verilog code that we developed in this course (https://github.com/SkillSurf/systemverilog/blob/master/rtl/full_adder.sv) and synthesize the RTL using the SAED 32nm EDK and Synopsys Design Compiler. Next, we will get the final layout after doing PnR using the Synopsys IC Compiler II. Finally, we will export the final .gds file and visualize it using kLayout.

This is the diagram and the .sv code of the full_adder.






```
/*
  This code describes a full_adder circuit
*/
module full_adder                    Module name
(
  input  logic a, b, ci,
  output logic sum, co               Input, output ports
);

  logic wire_1, wire_2;              Intermediate wires
  assign wire_1 = a ^ b;             // bitwise XOR
  assign wire_2 = wire_1 & ci;       // bitwise AND

  wire   wire_3 = a & b;             logic wire_1 = a^b
                                     // bitwise AND
                                     is not same
  always_comb begin
    co  = wire_2 | wire_3;           // bitwise OR
    sum = wire_1 ^ ci  ;             // bitwise XOR
  end

endmodule                            Three equivalent ways
                                     of combinational
                                     assignments
```

Synthesize a RTL for ASIC with Synopsys Design Compiler

In this sub section of the tutorial, you will gain experience using Synopsys Design Compiler (DC) to perform hardware synthesis. A synthesis tool takes an RTL hardware description and a standard cell library as input and produces a gate level netlist as output. The resulting gate-level netlist is a completely structural description with standard cells only at the leaves of the design.

Internally, a synthesis tool performs many steps including high-level RTL optimizations, RTL to unoptimized boolean logic, technology independent optimizations, and finally technology mapping to the available standard cells. Good RTL designers will familiarize themselves with the target standard cell library so that they can develop an intuition on how their RTL will be synthesized into gates.

In this tutorial, you will use Synopsys Design Compiler to elaborate the RTL for our full_adder circuit, set optimization constraints, synthesize the design to gates, and prepare various area and timing reports. You will also learn how to use the Synopsys Design Vision tool to visualize the synthesized design.

1. Make a folder called **0_SysV_<YourName>** in the Home directory (eg: **0_SysV_KithminR**). This is the folder inside which you will be saving all your practical files and doing your work.

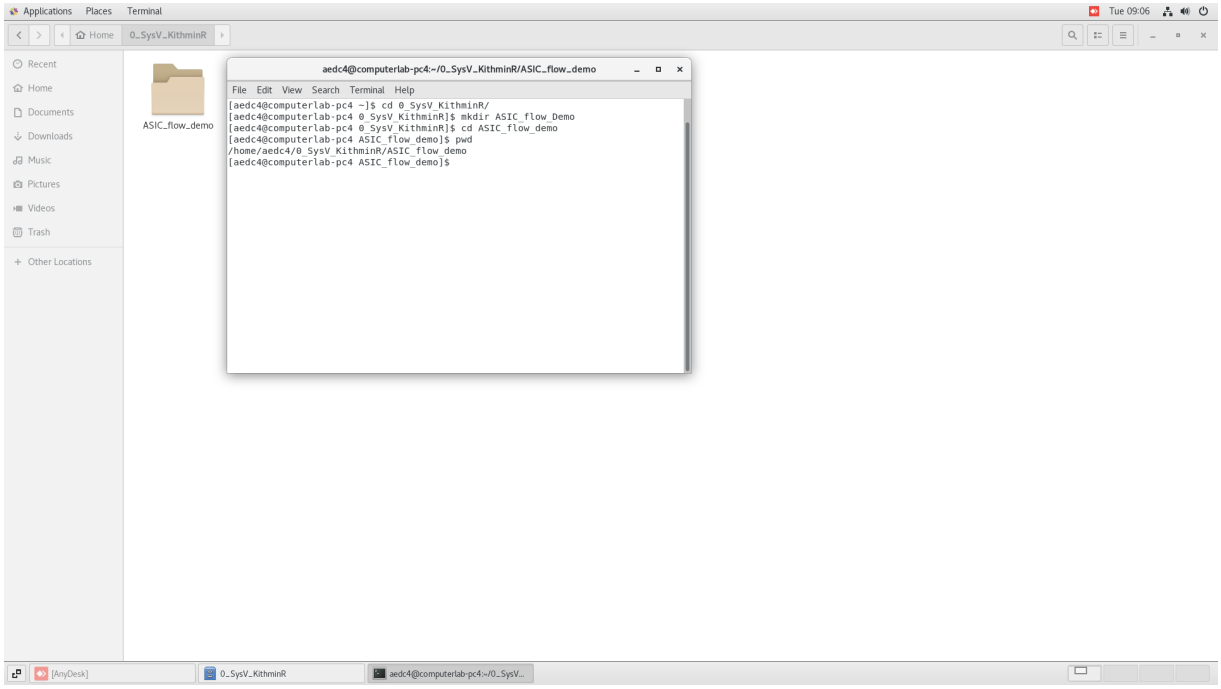
   IMPORTANT: Please don't tamper with or copy files from anyone else's folders. PLEASE DON'T TURN OFF THE COMPUTERS!

2. Type the following commands to make your ASIC_flow working directory;

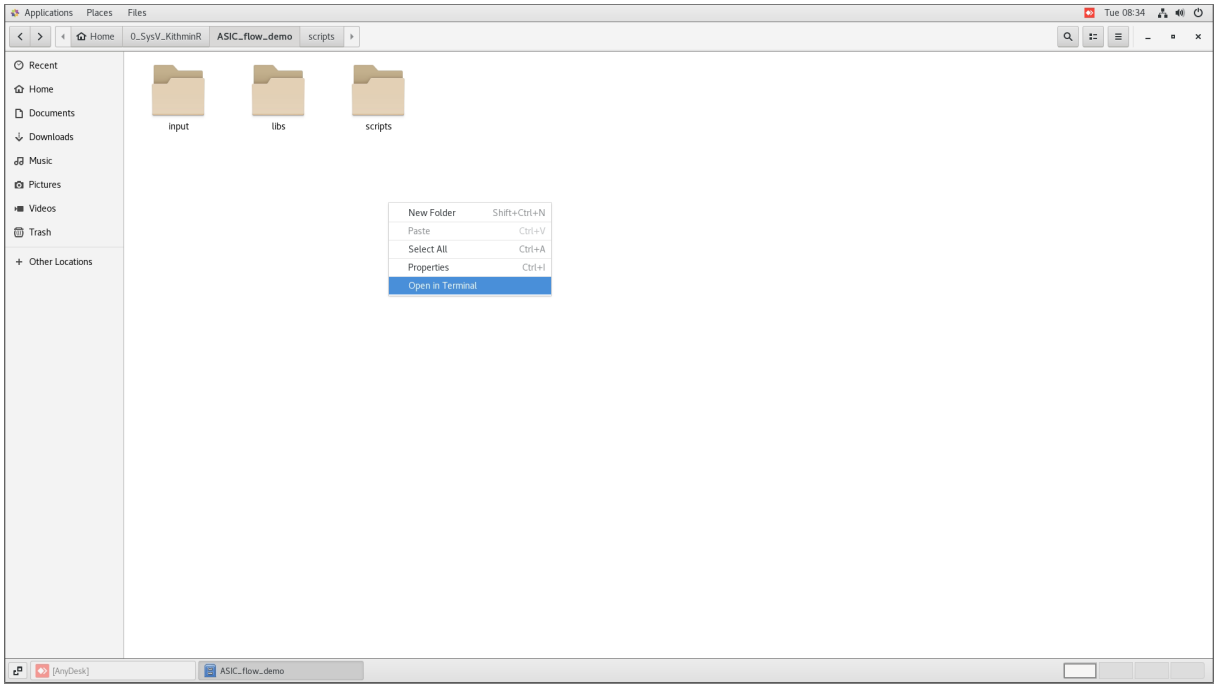
   $ cd 0_SysV_<YourName>/
   $ mkdir ASIC_flow_demo
   $ cd ASIC_flow_demo/

   Check your present working directory by typing 'pwd'.

   $ pwd



3. The following project hierarchy will be used, where you can download files inside into your ASIC_flow_demo/ folder from Google Drive or Github. The last 3 folders will be created later while running the script.
    a. |-libs/ - This is the directory containing the 32nm library files used for the project
    b. |-input/rtl/ - Contains HDL source files (.sv and .v)
    c. |-scripts/ - Keeps a copy of DC scripts being executed
        i. |-scripts/icc - Keeps a copy of ICC2 scripts being executed
    d. |-output/ - Design Compiler outputs
    e. |-log/ - Stores generated log files
    f. |-report/ - Design Compiler generated reports
4. Open the terminal from the /ASIC_flow_demo folder



Environment Setup

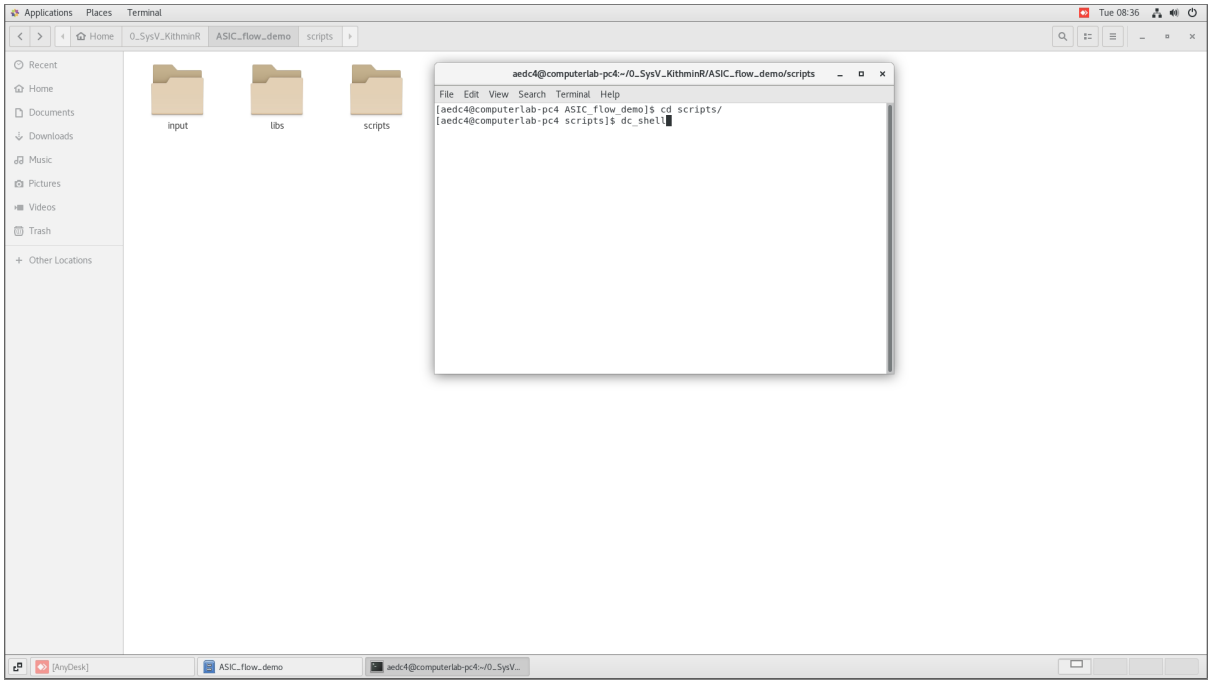
To be filled (continue to the next section).

Launching Synopsys Design Compiler

5. Launch the Synopsys Design Compiler shell from inside the /scripts folder, using the terminal.

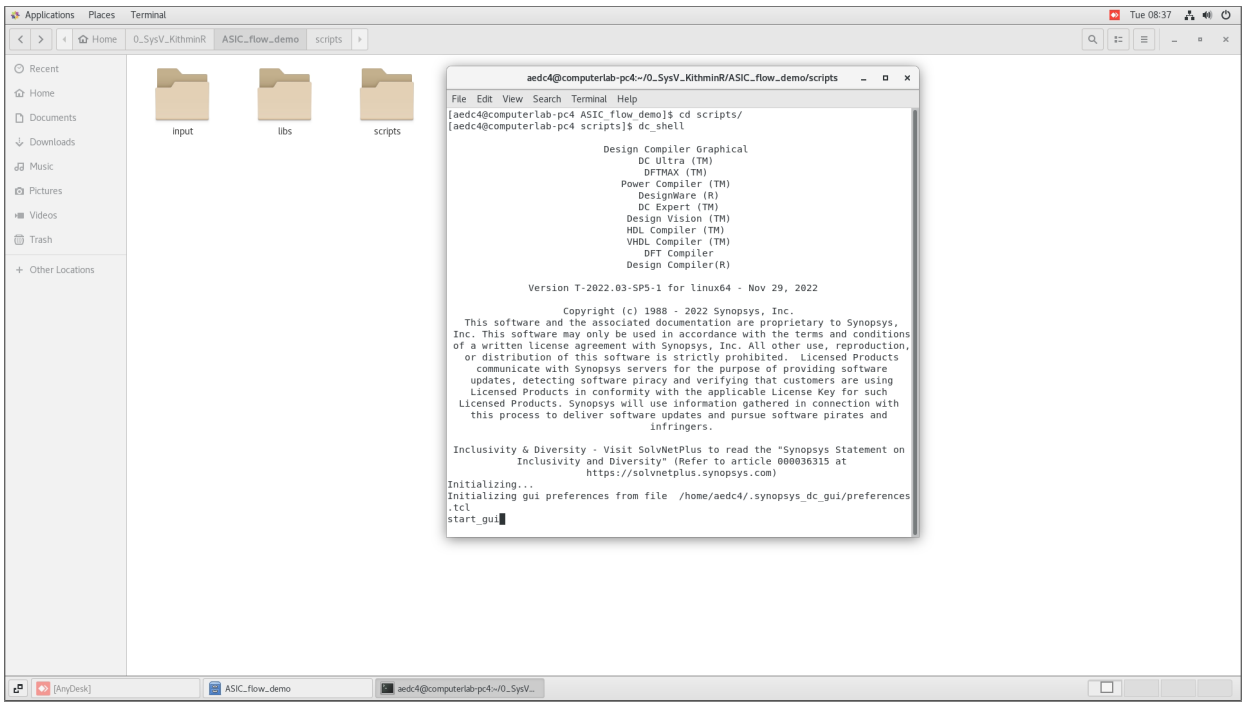
   $ cd scripts

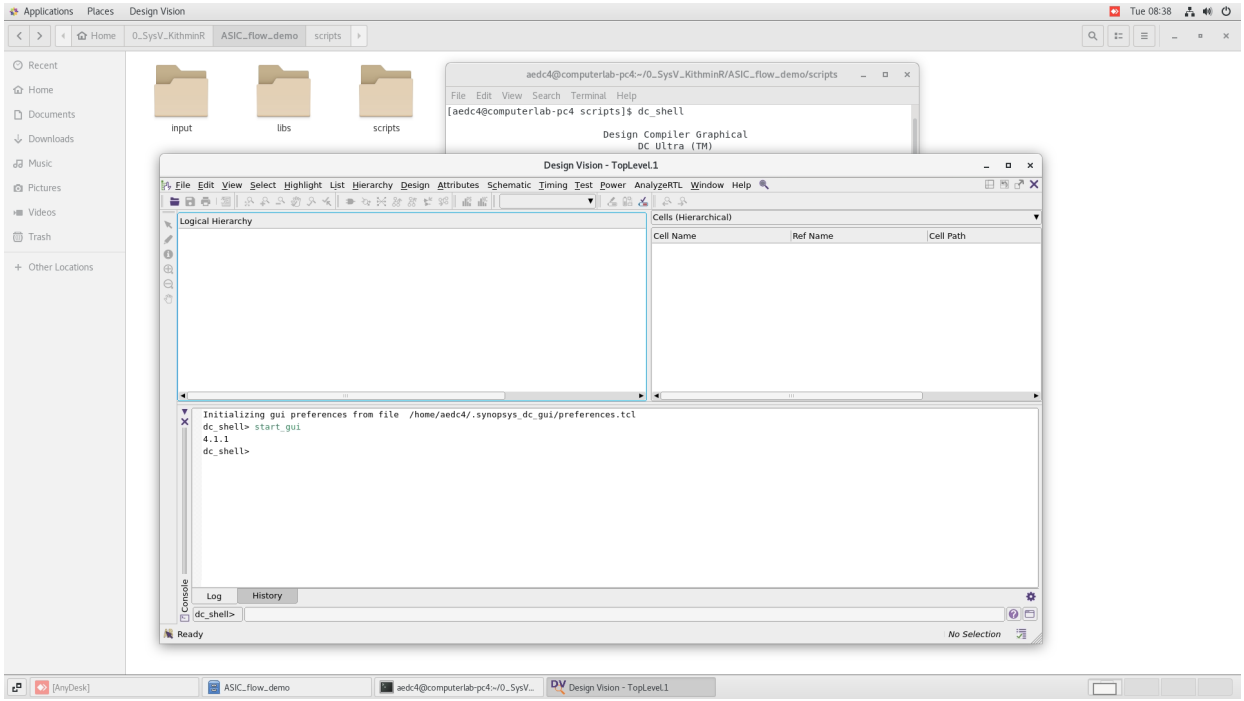
   $ dc_shell



6. Start DC by launching the GUI.

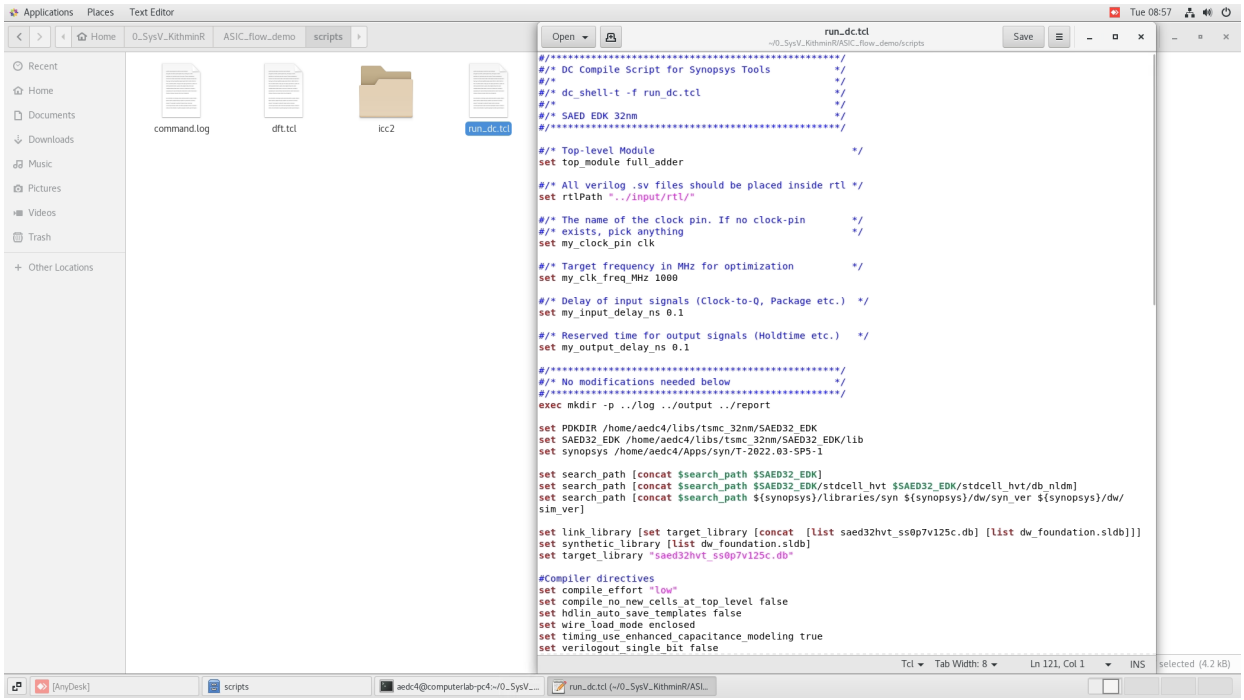
   dc_shell > start_gui





The TCL Instruction File

The Tool Command Language (TCL) instructions file contains a sequence of instructions we need to execute in the RTL compiler environment. Having it as a file is convenient because we don't need to type each and every command manually. While it is highly recommended you go through the following steps to understand what we are doing in the TCL file, one can skip to the Completed TCL File if they just want to copy-and-paste. You can open up the run_dc.tcl file which looks like this.



Writing the TCL file (Line by Line)

To be filled (continue to the next section).

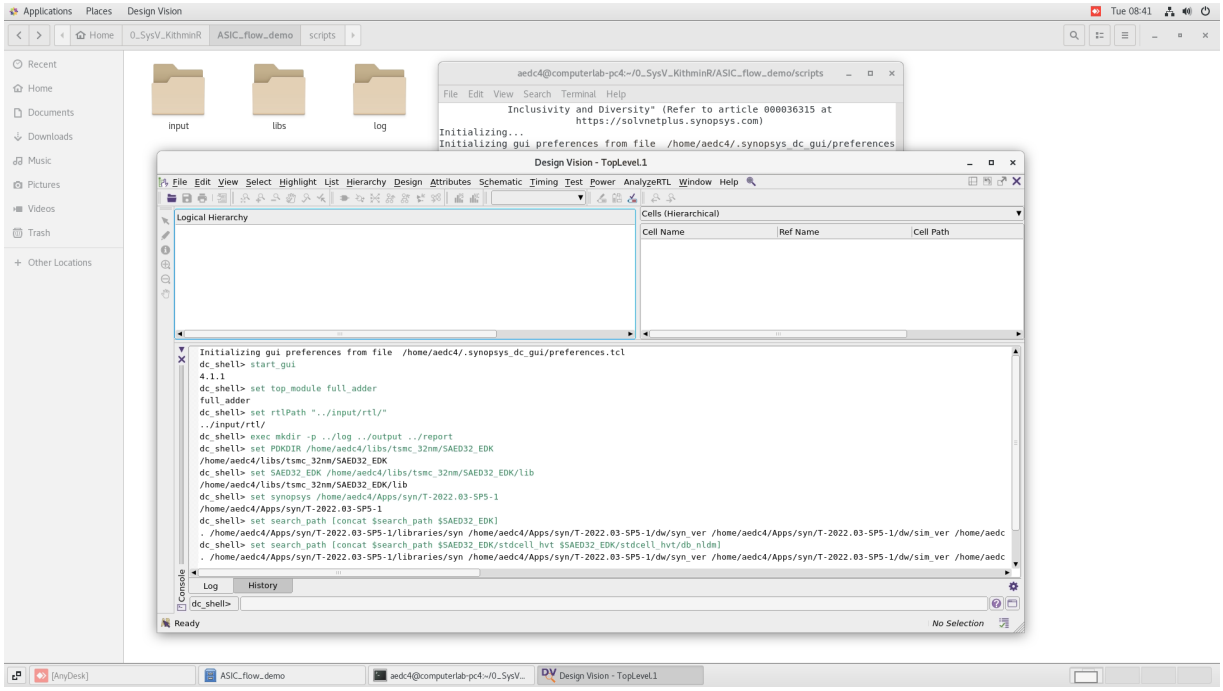
run_dc.tcl

```
#/*****************************************************/
#/* DC Compile Script for Synopsys Tools              */
#/*                                                   */
#/* dc_shell-t -f run_dc.tcl                          */
#/*                                                   */
#/* SAED EDK 32nm                                     */
#/*****************************************************/

#/* Top-level Module  - MODIFY as required            */
set top_module full_adder

#/* All verilog .sv files should be placed inside rtl */
set rtlPath "../input/rtl/"

#/* The name of the clock pin. If no clock-pin        */
#/* exists, pick anything                             */
set my_clock_pin clk

#/* Target frequency in MHz for optimization          */
set my_clk_freq_MHz 1000

#/* Delay of input signals (Clock-to-Q, Package etc.)  */
set my_input_delay_ns 0.2

#/* Reserved time for output signals (Holdtime etc.)   */
set my_output_delay_ns 0.2

# MODIFY as required - aedc4 might need to change
set PDKDIR /home/aedc4/libs/tsmc_32nm/SAED32_EDK
set SAED32_EDK /home/aedc4/libs/tsmc_32nm/SAED32_EDK/lib
set synopsys /home/aedc4/Apps/syn/T-2022.03-SP5-1

#/*****************************************************/
#/* No modifications needed below                     */
#/*****************************************************/
```

Running the TCL Instruction File (Line by Line)

7. You can copy cells line by line and execute them on the dc_shell terminal as shown below.

   dc_shell >



8. Note that before running the commands, there are some things that might need to be modified. For example, check the run_dc.tcl script to make sure the following file paths are correctly specified. If not, please change the paths by using the pwd command inside the folders.

   $ pwd

```
#/* Top-level Module  - MODIFY as required                */
set top_module full_adder

# MODIFY as required - aedc4 might need to change
set PDKDIR /home/aedc4/libs/tsmc_32nm/SAED32_EDK
set SAED32_EDK /home/aedc4/libs/tsmc_32nm/SAED32_EDK/lib
set synopsys /home/aedc4/Apps/syn/T-2022.03-SP5-1
```

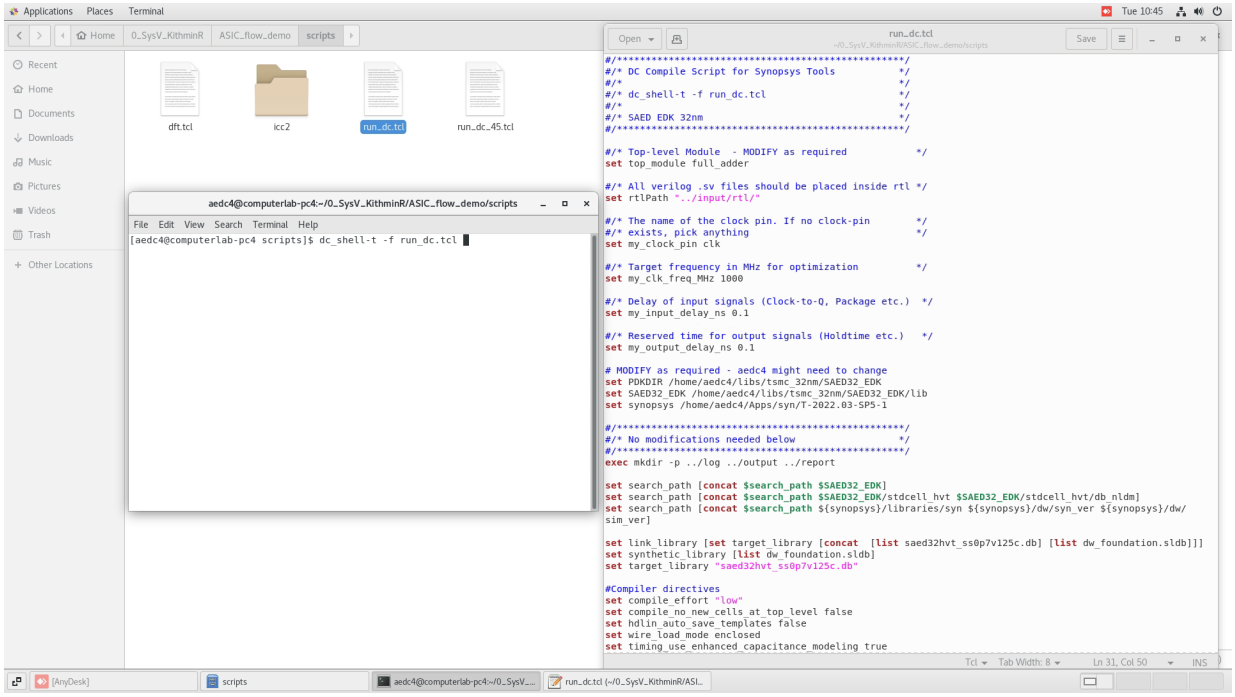
For some machines;

```
set synopsys /home/aedc4/synopsys/apps/syn/T-2022.03-SP5-1
```

Running the TCL Instruction File from the Terminal

9. Once you have made the above adjustments based on the machine you are using, you can also run the entire run_dc.tcl file from the terminal as well. You can use the command;

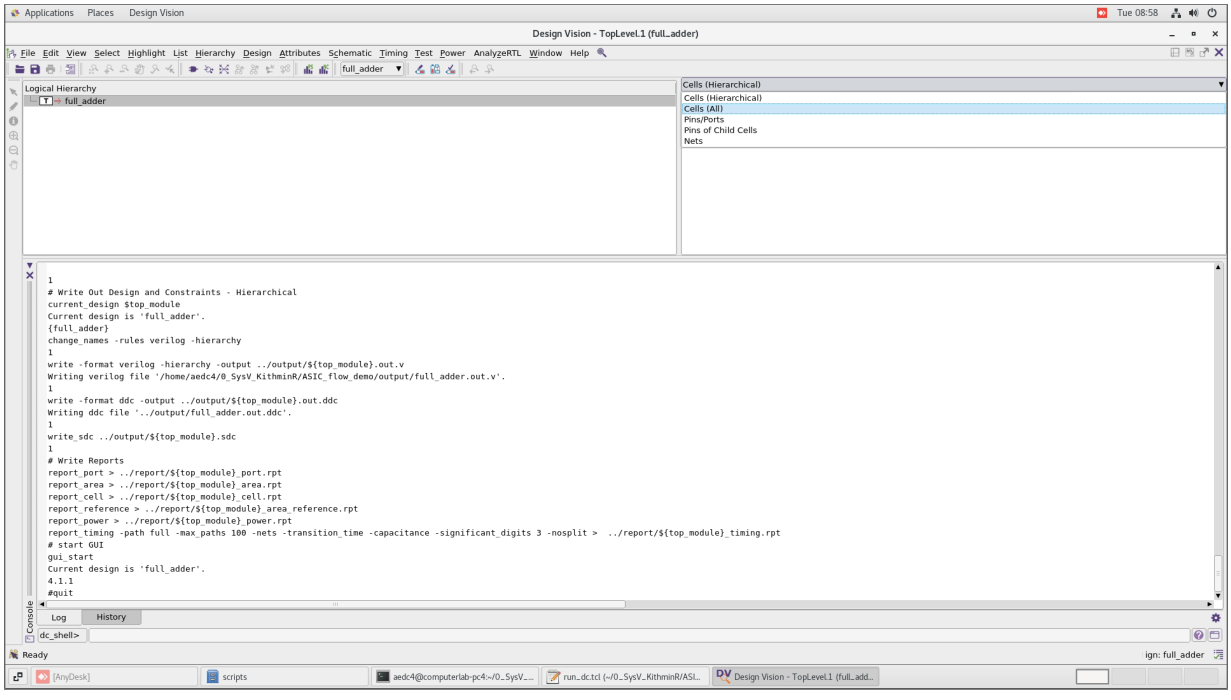
   $ dc_shell-t -f run_dc.tcl



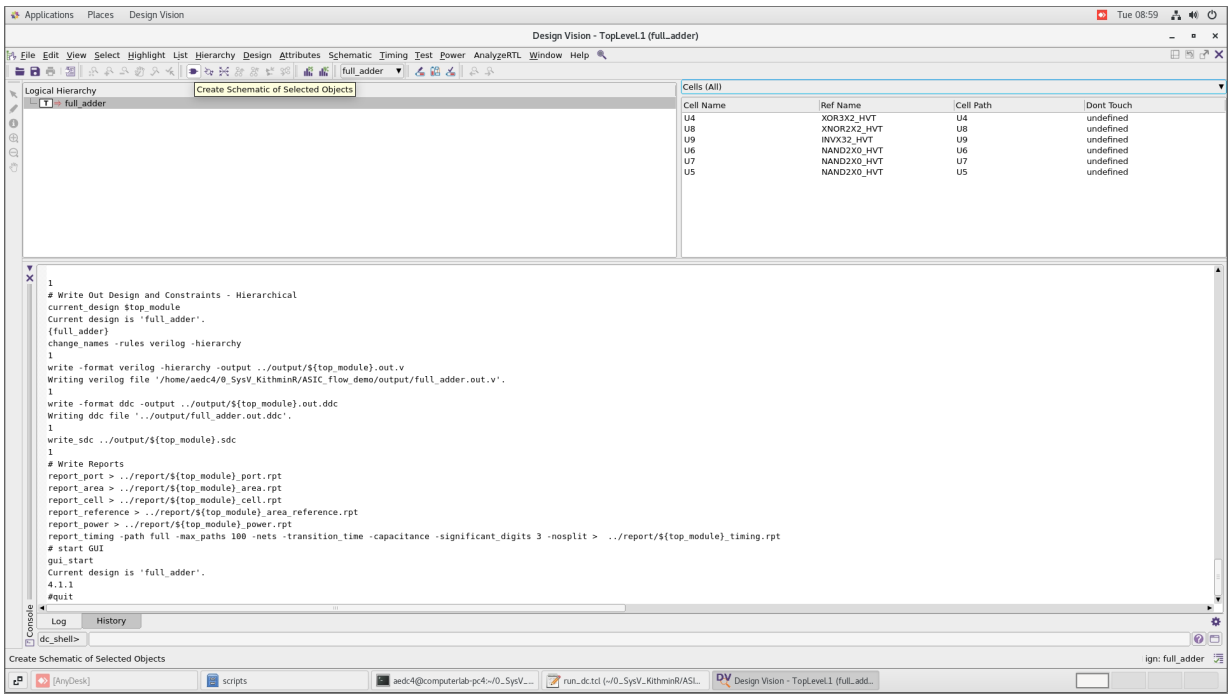
Analyzing the Compiled RTL Netlist through Design Compiler

10. Check your present working directory by typing ‘pwd’.

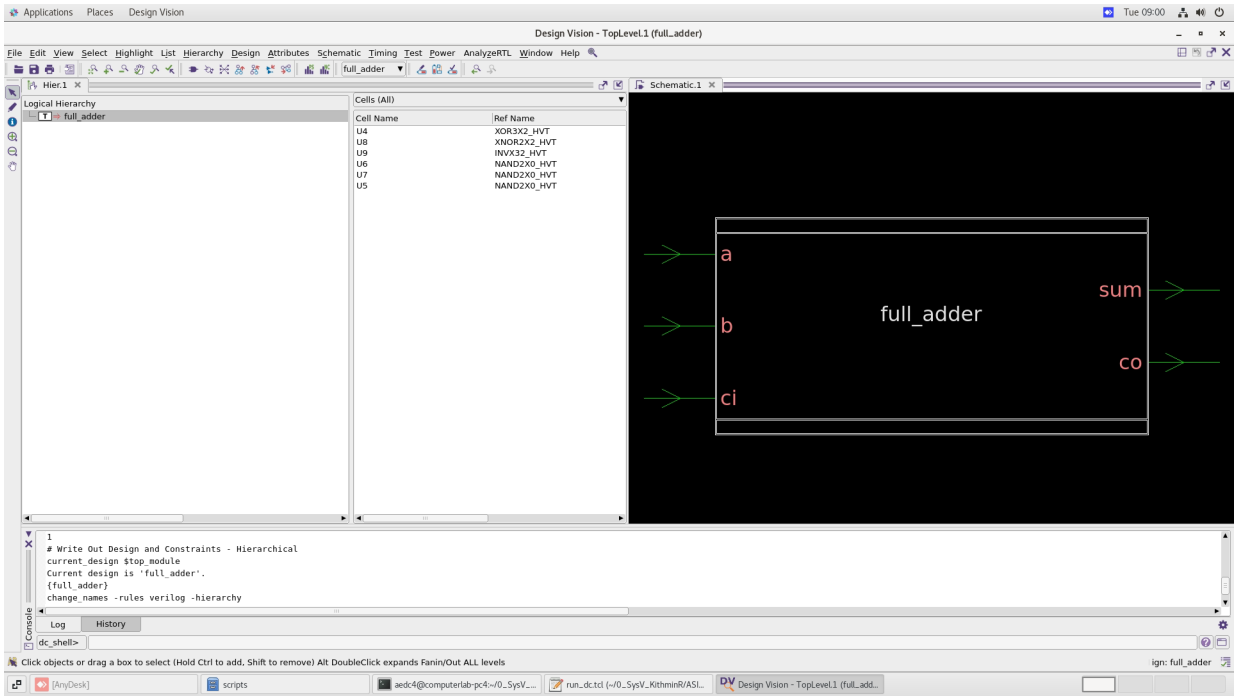
$ pwd



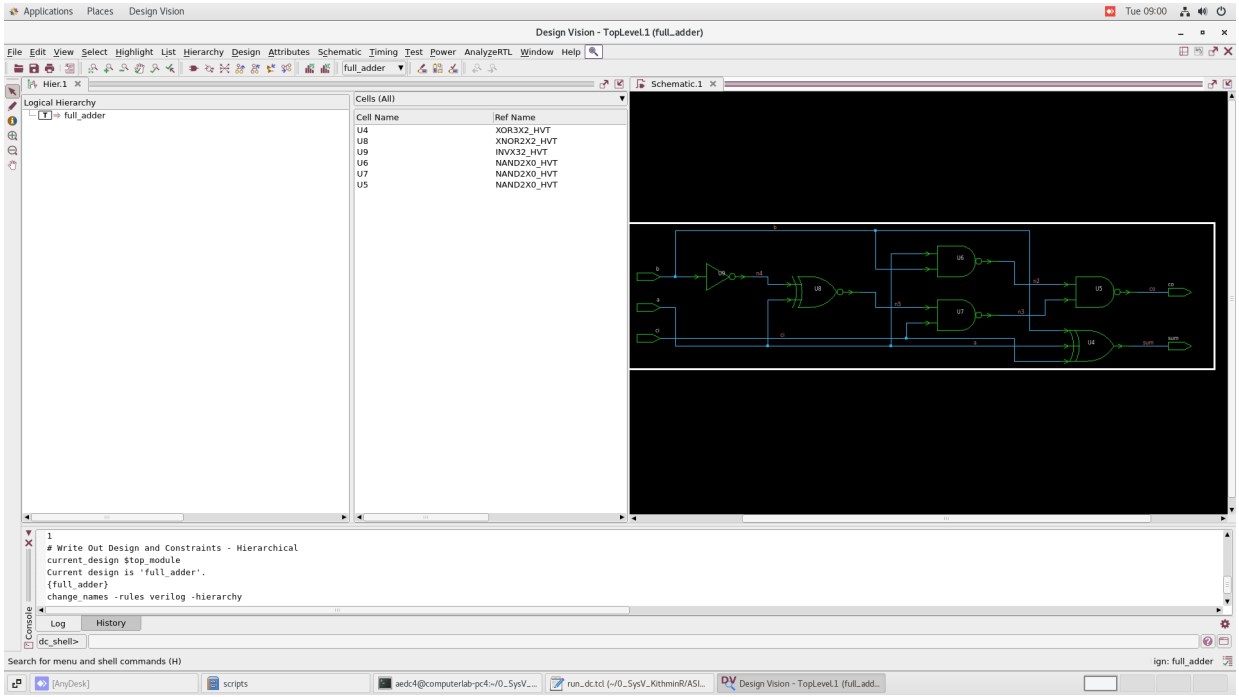
Create the Schematic of your compiled netlist



Look at the top_level module of the design

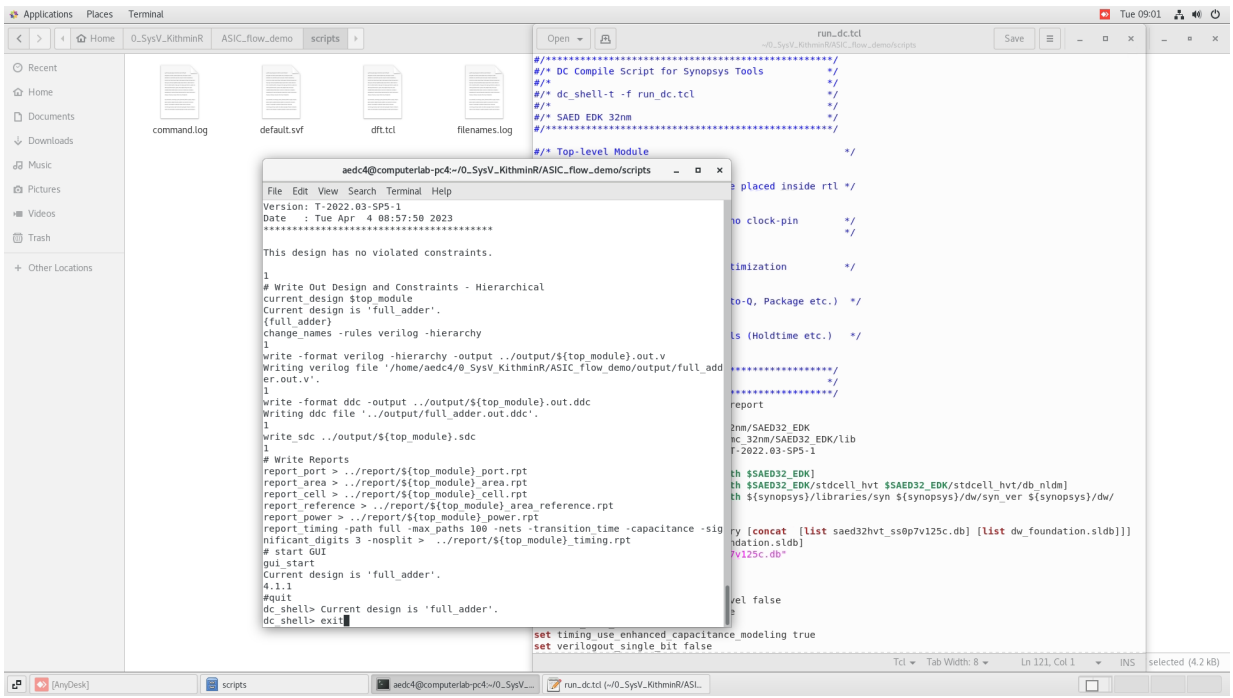


Double-click the top-level module to look inside the design and see the connections.



You can exit the GUI after you have looked at the schematic.

dc_shell > exit



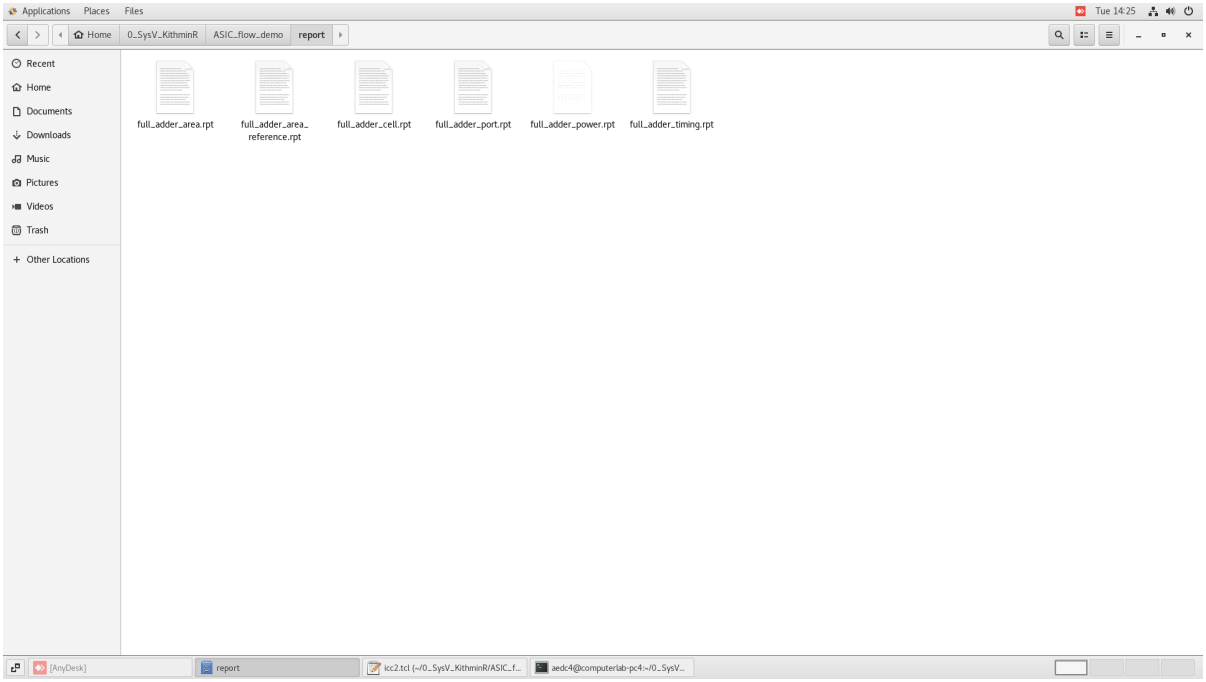
11. You can find the synthesized verilog netlist file full_adder.out.v inside the output/ folder, which should look like this.

full_adder.out.v

```
////////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : T-2022.03-SP5-1
// Date      : Tue Apr  4 14:25:03 2023
////////////////////////////////////////////////////////////////


module full_adder ( a, b, ci, sum, co );
  input a, b, ci;
  output sum, co;
  wire   n2, n3, n4, n5;

  XOR3X2_HVT U4 ( .A1(b), .A2(a), .A3(ci), .Y(sum) );
  NAND2X0_HVT U5 ( .A1(n2), .A2(n3), .Y(co) );
  NAND2X0_HVT U6 ( .A1(a), .A2(b), .Y(n2) );
  NAND2X0_HVT U7 ( .A1(n5), .A2(ci), .Y(n3) );
  XNOR2X2_HVT U8 ( .A1(n4), .A2(a), .Y(n5) );
  INVX32_HVT U9 ( .A(b), .Y(n4) );
endmodule
```

Analyzing the Compiler Reports

12. You can also go inside the report/ folder to find the different reports generated during the synthesis process of the full adder circuit. Each of the reports contain information about
    a. Ports
    b. Area
    c. Power
    d. Timing
    e. No. of Cells



Place and Route in IC Design with Synopsys IC Compiler II

In this sub section of the tutorial, you will gain experience transforming a gate-level netlist into a placed and routed layout using Synopsys IC Compiler II (ICC2). ICC takes a synthesized gate-level netlist and a standard cell library as input, then produces layout as an output.

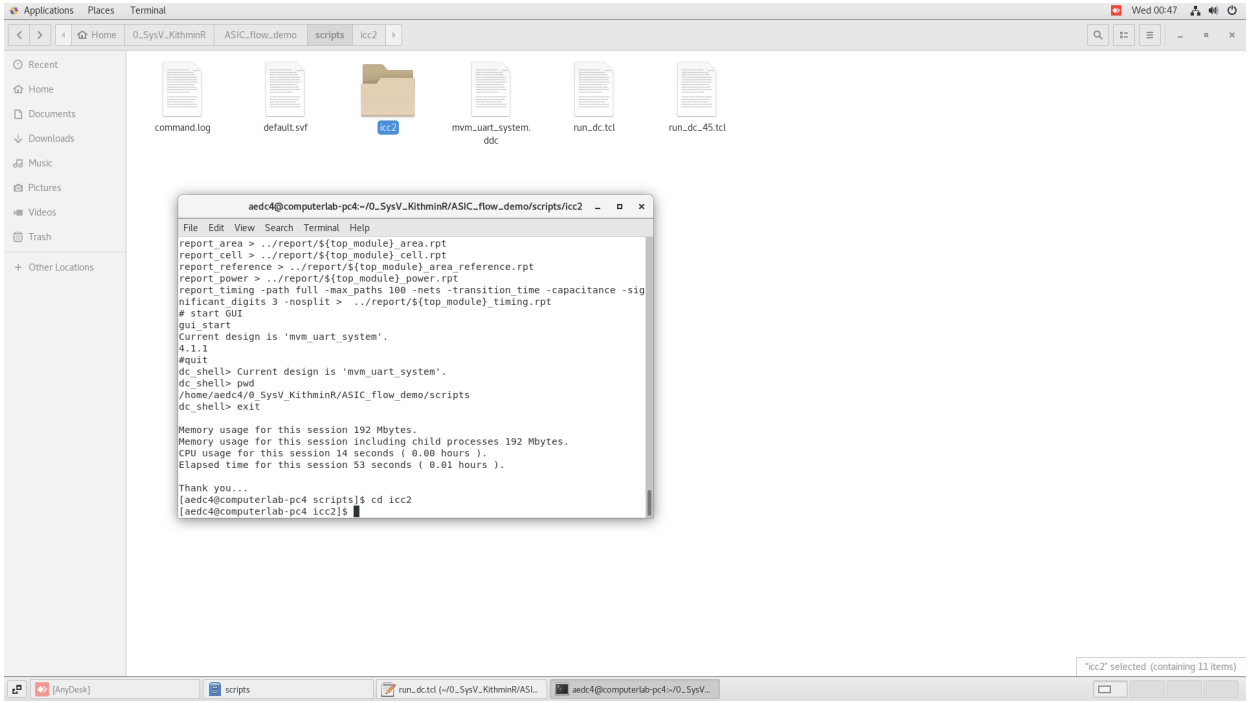
The first goal of a Place and Route (PnR) tool is to determine where each gate should be located on the physical chip (the placement portion of place and route). This process leverages heuristic algorithms to group related gates together in hopes of minimizing routing congestion and wire delay. Typically PnR tools will focus their effort on minimizing the delay through the critical path, and to this end will resize gates, insert new buffers, and even perform local re-synthesis. Additionally, PnR tools often have secondary algorithms to help reduce area for non-critical paths. After the placement, ICC will attempt to route the design while minimizing wire delay. Along with the routing, PnR tools often handle clock tree synthesis, power routing, and block level floorplanning.

For this tutorial, we will generate a layout for the gate-level netlist of the full_adder circuit synthesized in the previous section. Once the netlist has successfully been placed and routed, you should be able to see all the instantiated standard cells and routed metals of the physical implementation. We will then use the kLayout to visualize the layout of your final placed and routed design.

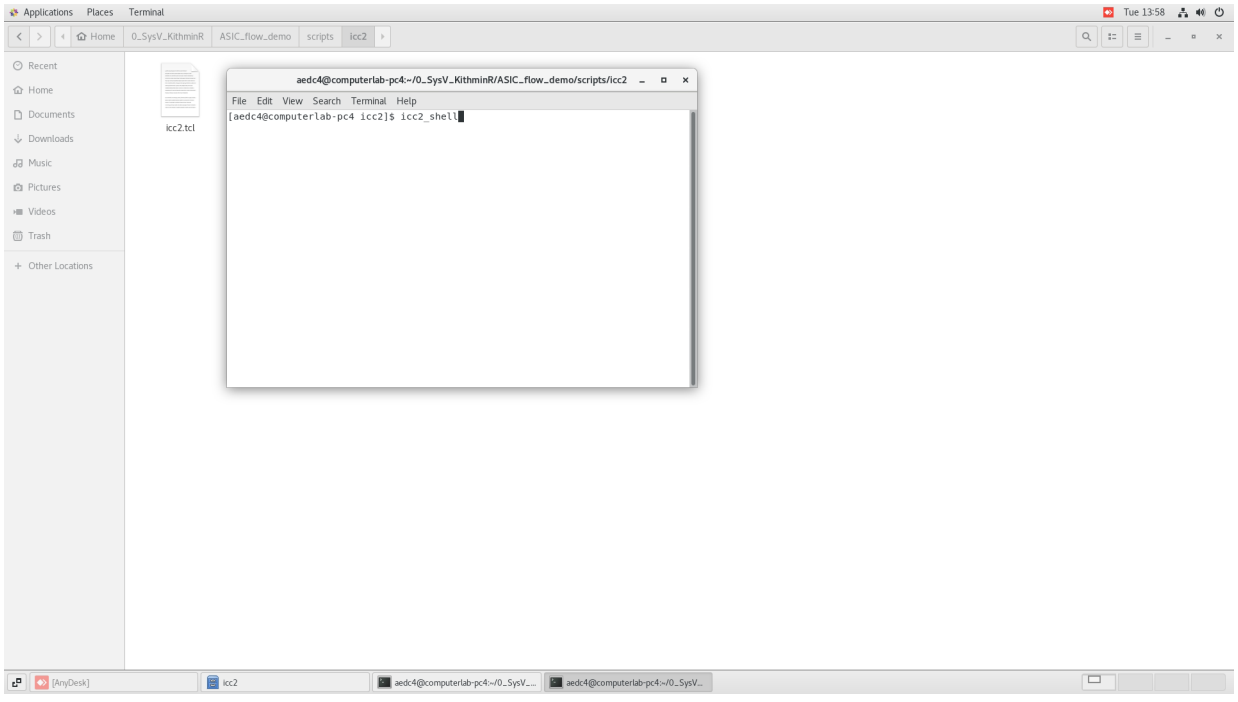
Launching Synopsys IC Compiler II

13. Go inside the folder scripts/ic2/ and launch the IC Compiler II shell using the command icc2_shell

    $ cd icc

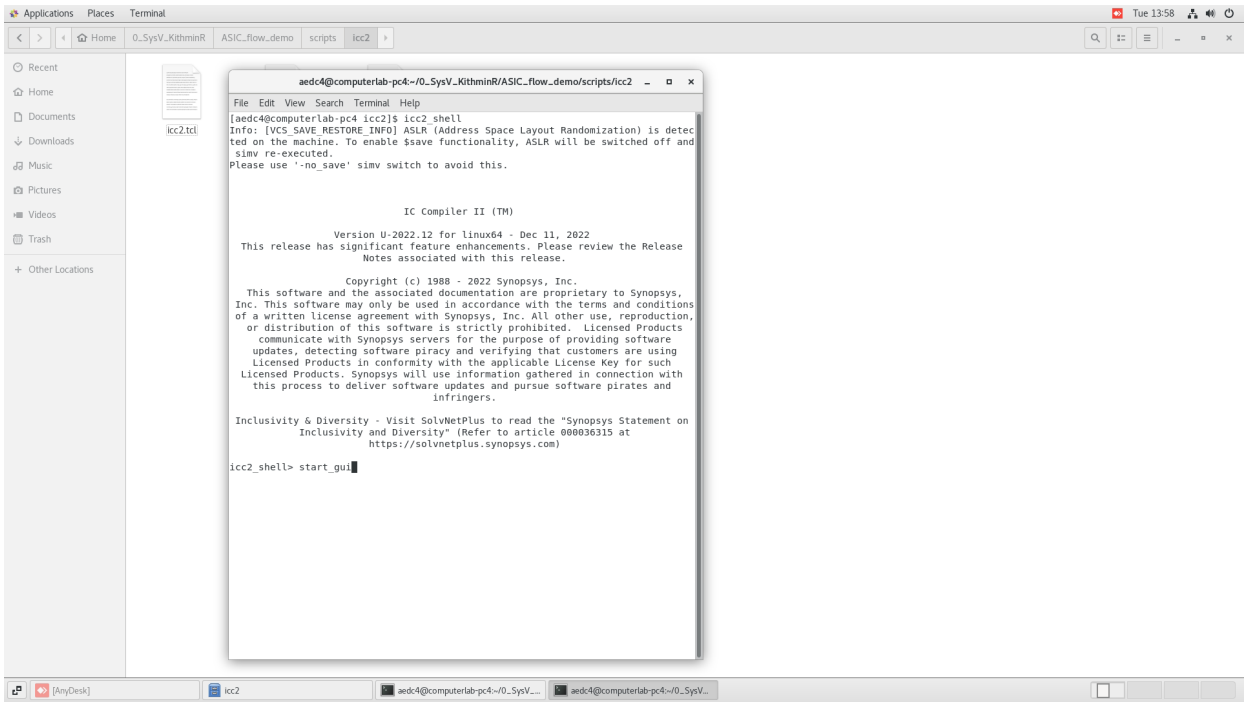


$ icc2_shell



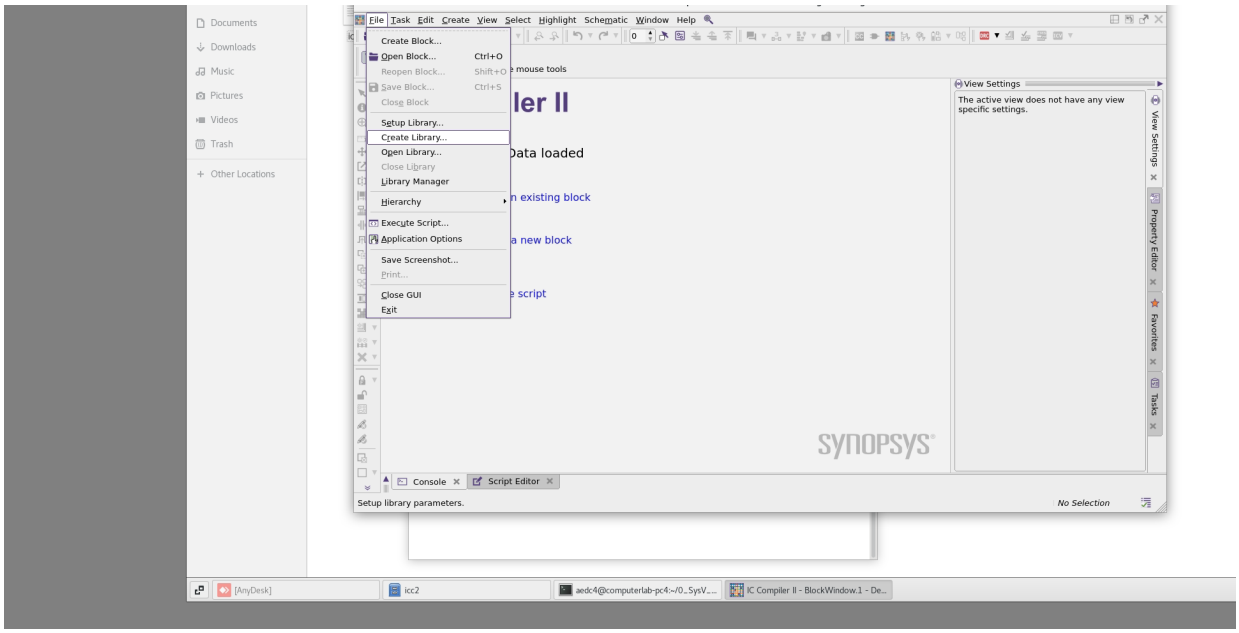
14. Start ICC2 by launching the GUI.

    icc2_shell > start_gui



Creating a Library

15. We begin with creating a library for our project, within which we can create multiple blocks. The library needs to be built using a specific PDH, which in our case will be SAED32_EDK. You can find the documentation about this PDK on the Synopsys Tools Section. Click the option shown to start creating a library.

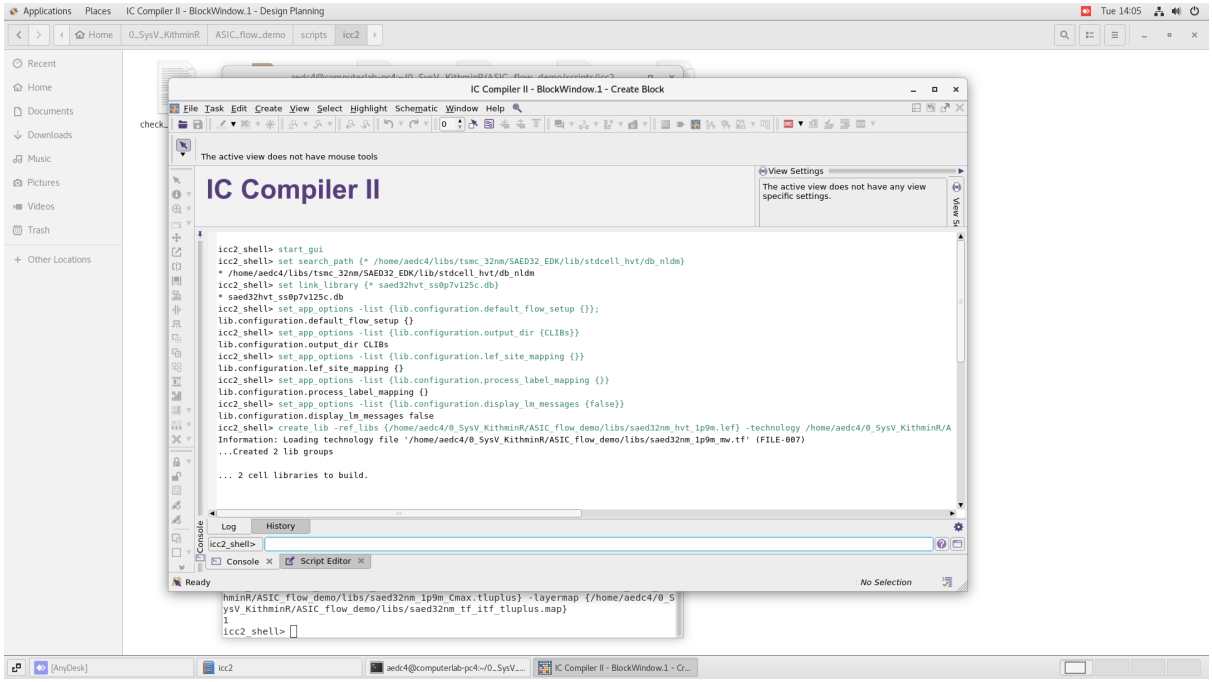


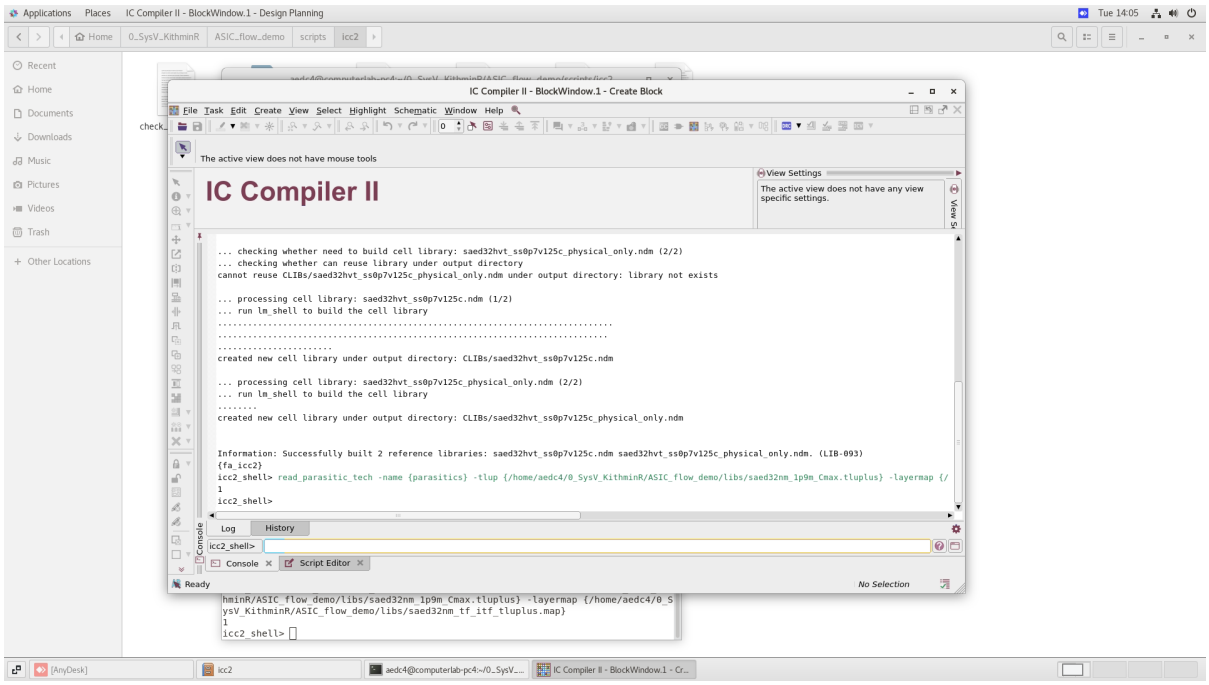
16. Note that before running the commands, there are some things that might need to be modified. For example, check the icc2.tcl script to make sure the following file paths are correctly specified. If not, please change the paths by using the pwd command inside the folders.

    $ pwd

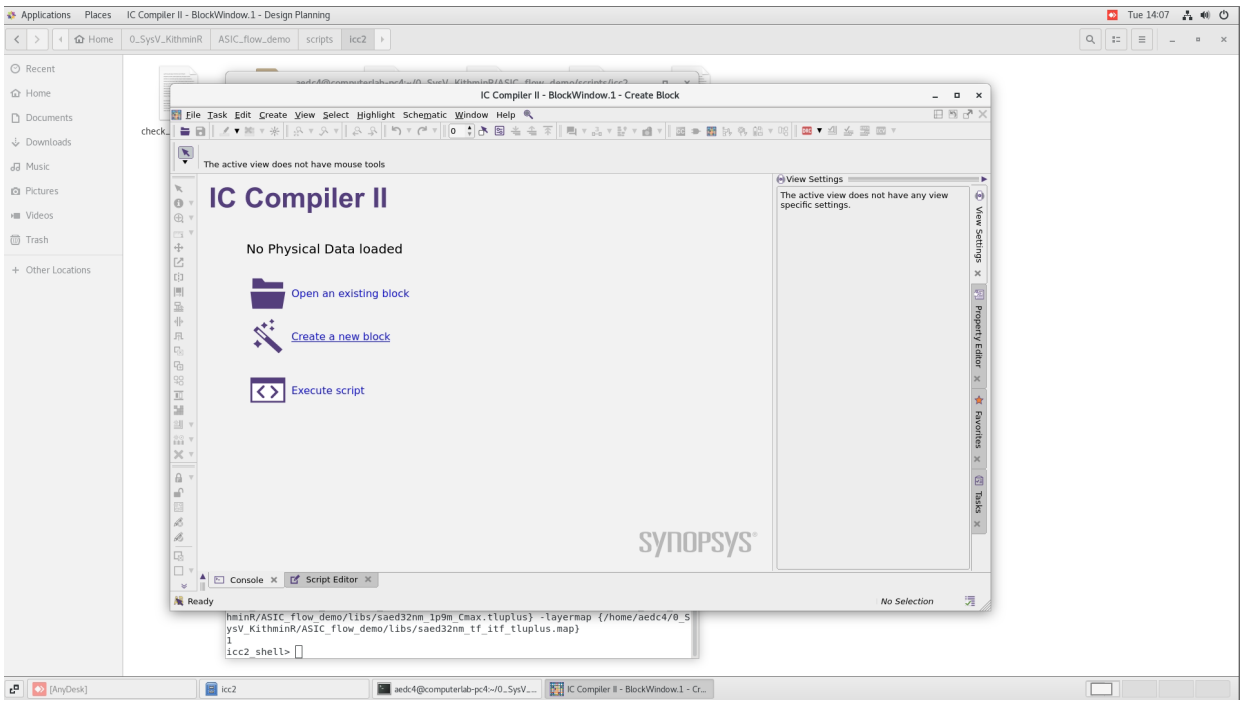
```
#  MODIFY as required

#/* Top-level Module                                    */
set top_module full_adder

#/* Library Name                                        */
set library_name fa_icc2

set PDKDIR /home/aedc4/libs/tsmc_32nm/SAED32_EDK
set SAED32_EDK /home/aedc4/libs/tsmc_32nm/SAED32_EDK/lib
set synopsys /home/aedc4/Apps/syn/T-2022.03-SP5-1

set search_path [concat $search_path $SAED32_EDK/stdcell_hvt
$SAED32_EDK/stdcell_hvt/db_nldm]
set link_library {* saed32hvt_ss0p7v125c.db}
set_app_options -list {lib.configuration.default_flow_setup
{}};
set_app_options -list {lib.configuration.output_dir {CLIBs}}
set_app_options -list {lib.configuration.lef_site_mapping {}}
set_app_options -list {lib.configuration.process_label_mapping
{}}
set_app_options -list {lib.configuration.display_lm_messages
{false}}

#-----------------------------------------------
#   Create Library
#-----------------------------------------------

create_lib -ref_libs
{/home/aedc4/0_SysV_KithminR/ASIC_flow_demo/libs/saed32nm_hvt_1p
9m.lef} -technology
/home/aedc4/0_SysV_KithminR/ASIC_flow_demo/libs/saed32nm_1p9m_mw
.tf $library_name

read_parasitic_tech -name {parasitics} -tlup
{/home/aedc4/0_SysV_KithminR/ASIC_flow_demo/libs/saed32nm_1p9m_C
max.tluplus} -layermap
{/home/aedc4/0_SysV_KithminR/ASIC_flow_demo/libs/saed32nm_tf_itf
_tluplus.map}
```

17. Compile all the commands one after the other, as listed in the icc2.tcl file up until the Create Library commands as shown above. The library will be named as fa_icc2.

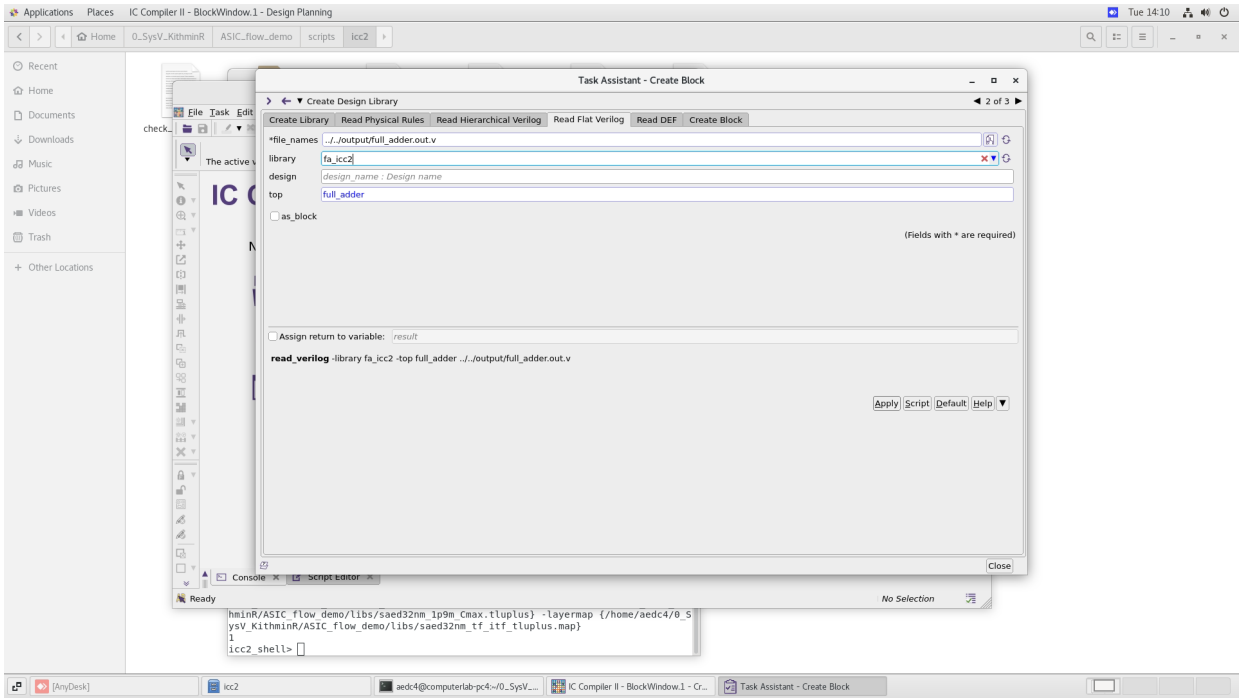




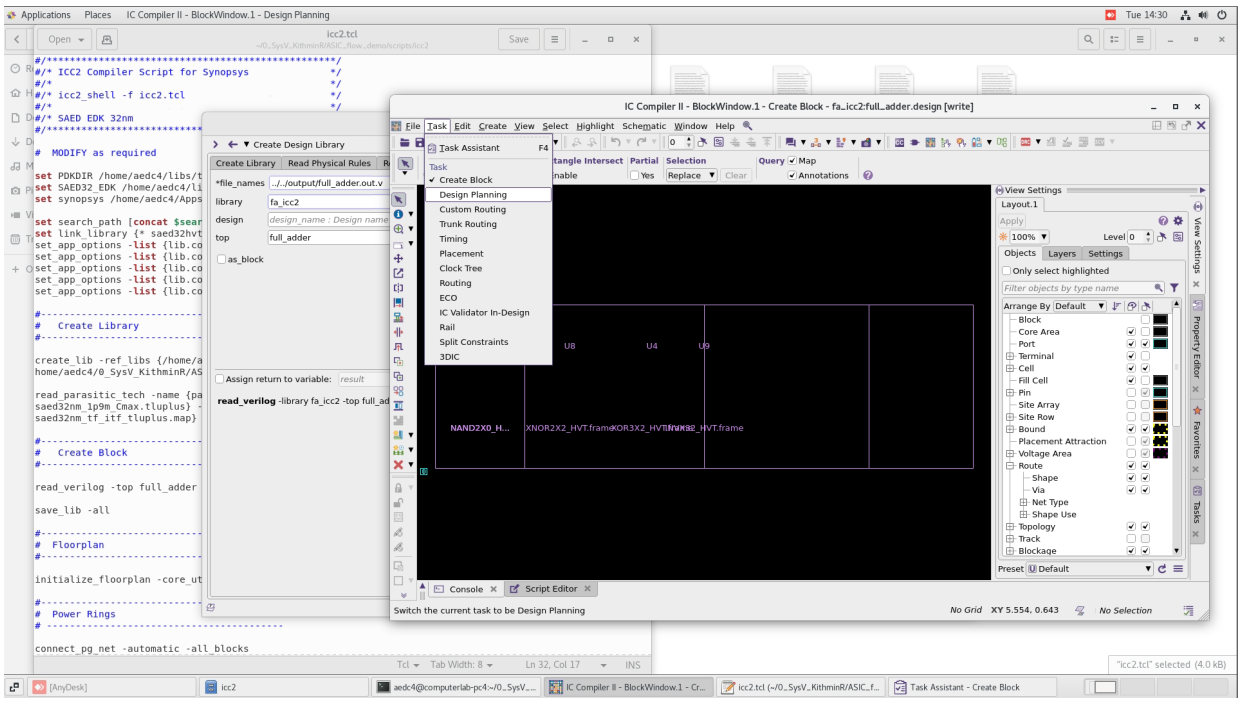
18. We can get started with creating our block now. Use the “Create a new block” functionality given in the GUI.



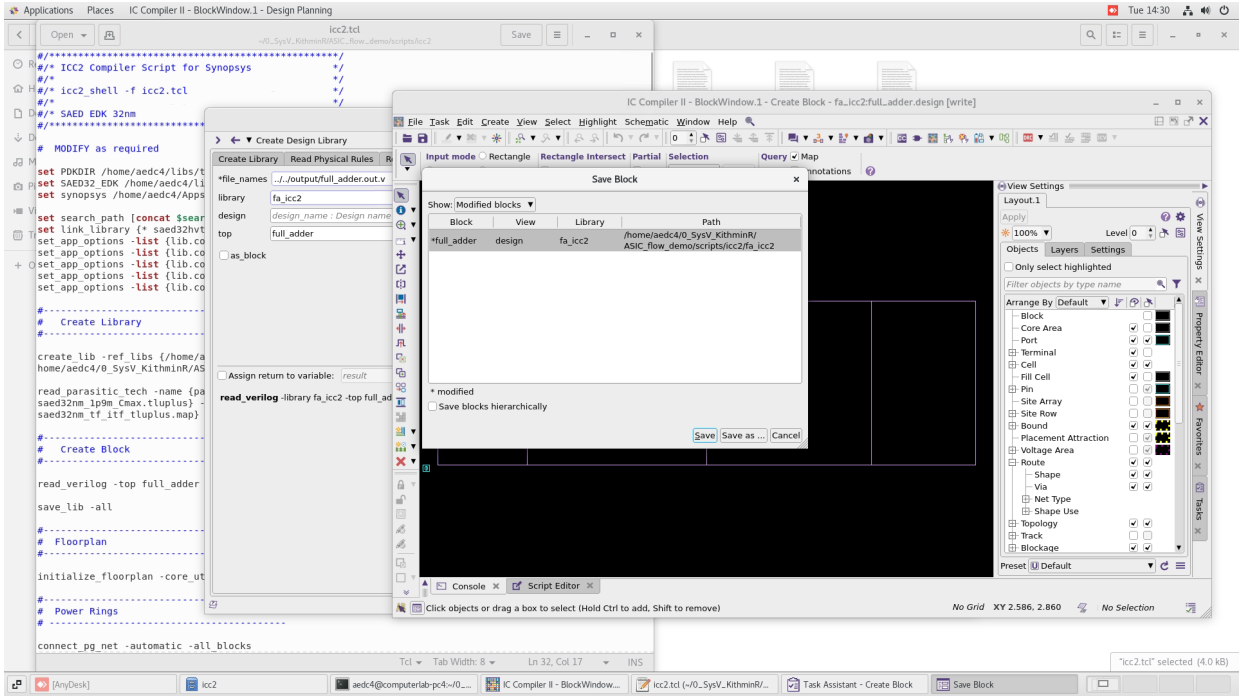
Choose the Read Flat Verilog File option, where you can open the full_adder.out.v file in this window as shown below and name the top module as shown. Press Apply.



You will see some cells are imported as seen below.

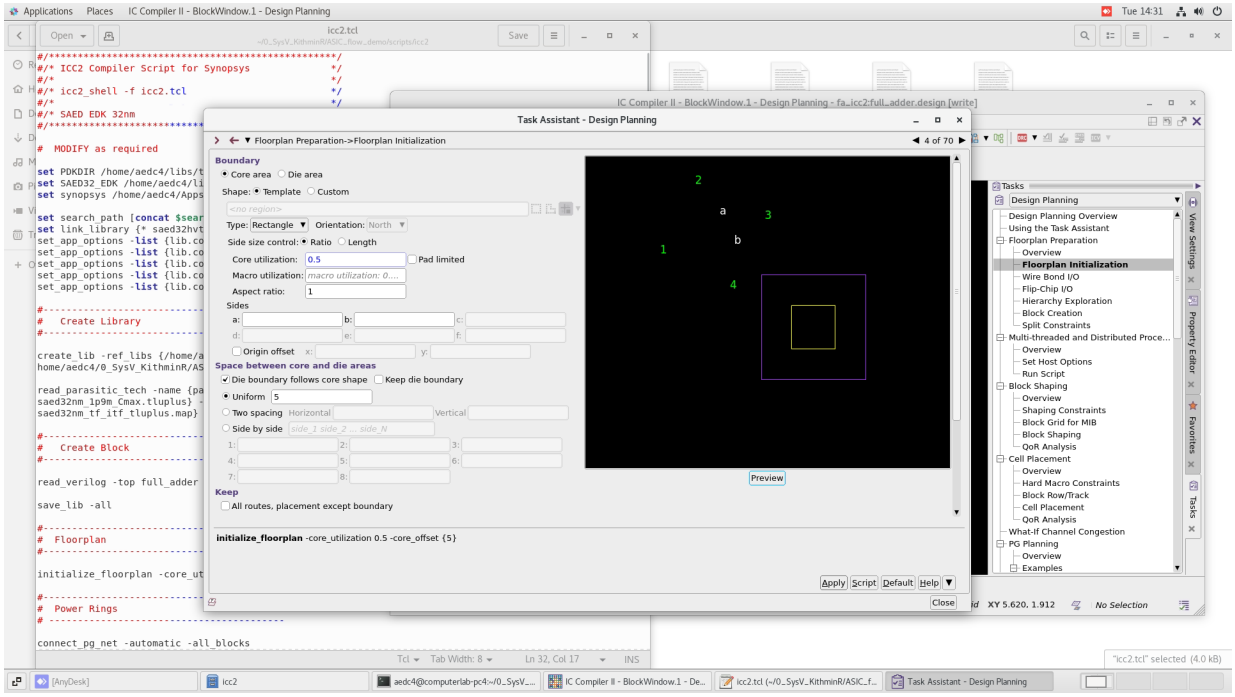


Save your project block as shown.

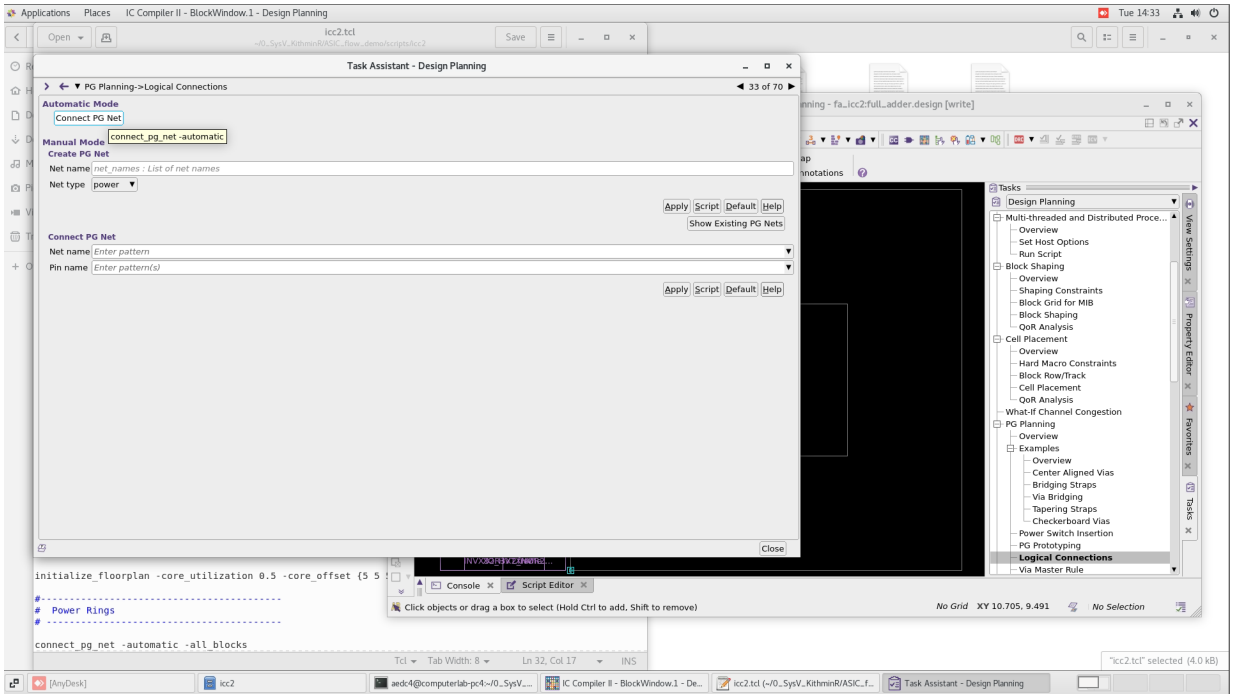


Design Floor and Power Planning Task

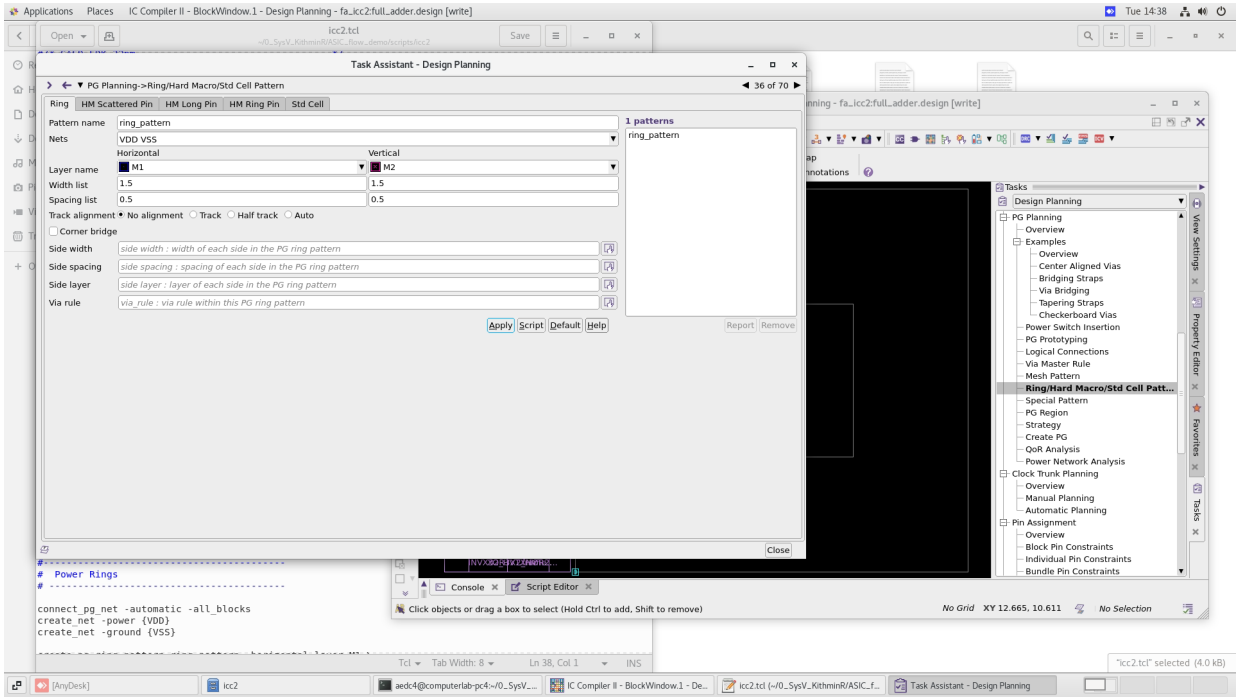
19. We will now go through the sequence of steps involved in the Design Planning Tasks, starting with Floor Planning and then moving on to Power planning. Use the windows to follow the options given, which are quite self explanatory. Floor planning sets up the core block inside for the cell placements and the space between the core and die for the power rings.

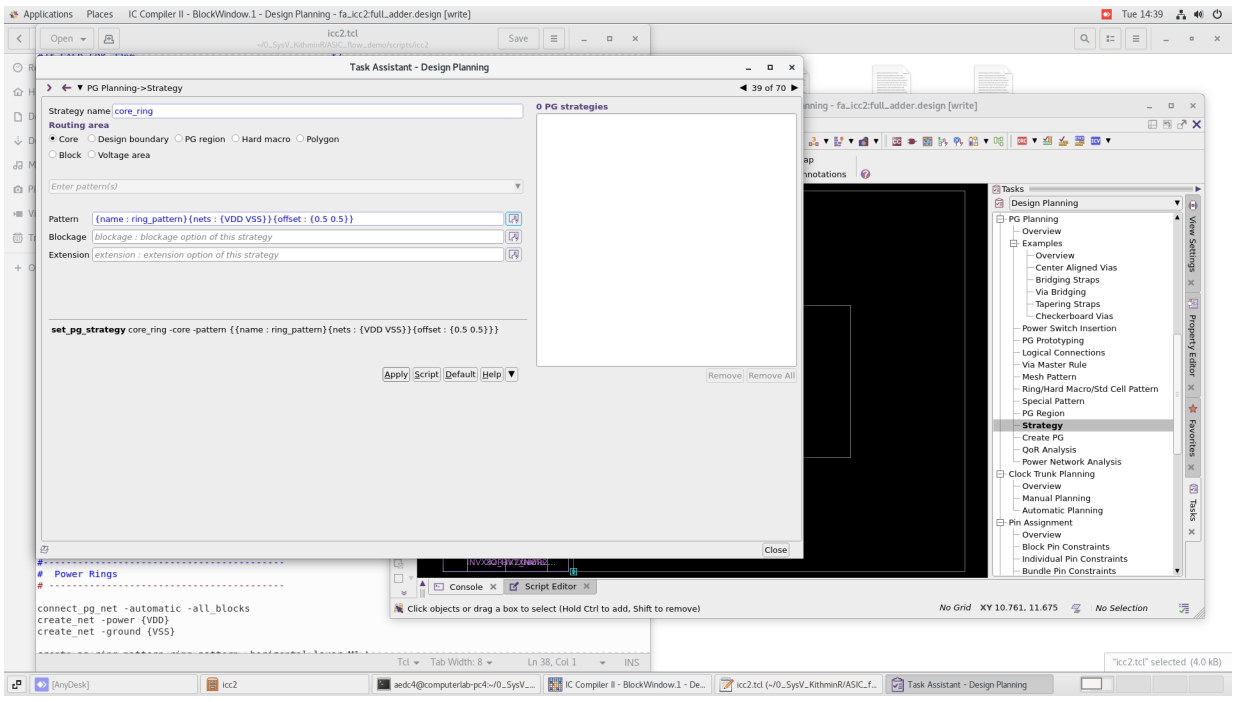


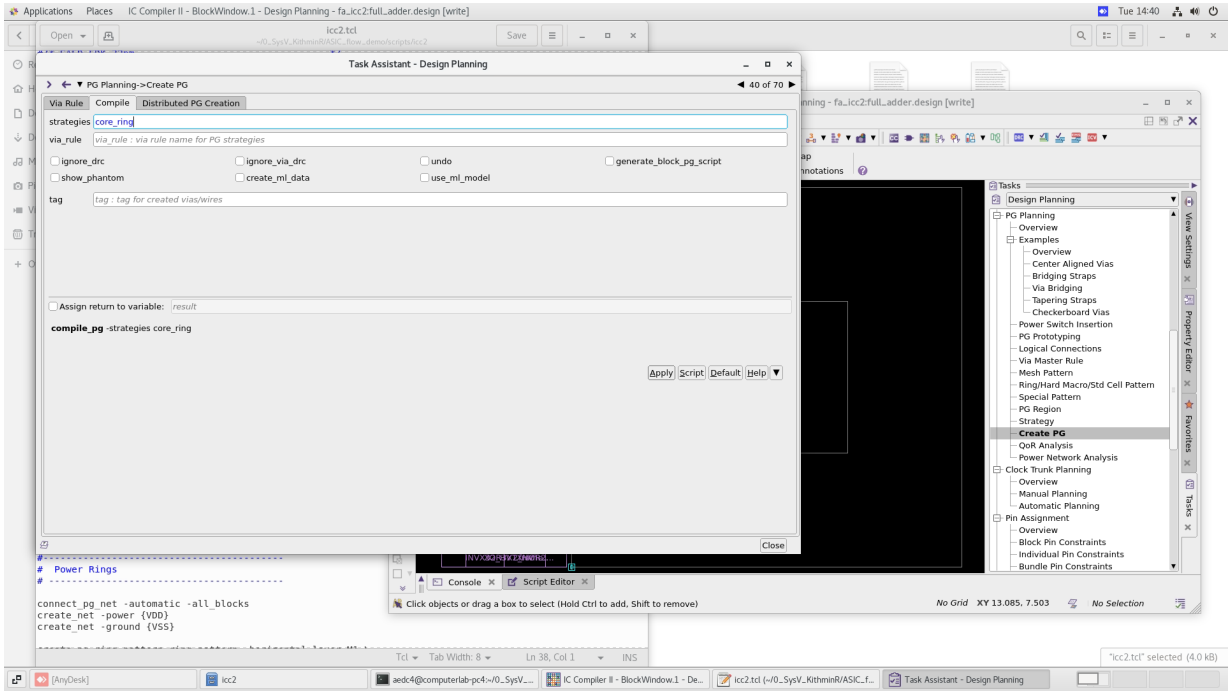
After Floor Planning, You should choose the Connect PG Net option under Logical Connections and then move into Power planning.



The next few steps outline how to create the VDD and VSS rings around the core.

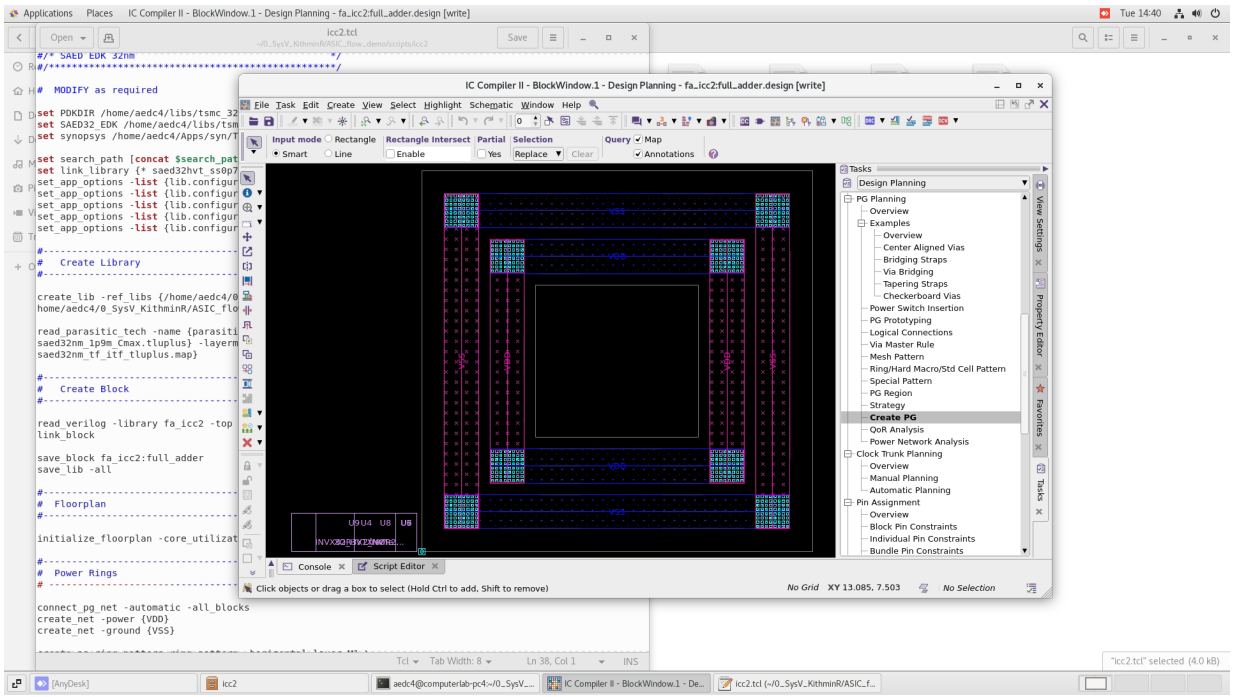

Applications Places IC Compiler II - BlockWindow.1 - Design Planning - fa_icc2:full_adder.design [write]
Tue 14:38
Task Assistant - Design Planning
PG Planning->Ring/Hard Macro/Std Cell Pattern
36 of 70
Ring
HM Scattered Pin
HM Long Pin
HM Ring Pin
Std Cell
Pattern name
ring_pattern
Nets
VDD VSS
Horizontal
Vertical
Layer name
M1
M2
Width list
1.5
1.5
Spacing list
0.5
0.5
Track alignment
No alignment
Track
Half track
Auto
Corner bridge
Side width
side width : width of each side in the PG ring pattern
Side spacing
side spacing : spacing of each side in the PG ring pattern
Side layer
side layer : layer of each side in the PG ring pattern
Via rule
via_rule : via rule within this PG ring pattern
1 patterns
ring_pattern
Apply
Script
Default
Help
Report
Remove
Close
Design Planning
PG Planning
Overview
Examples
Center Aligned Vias
Bridging Straps
Via Bridging
Tapering Straps
Checkerboard Vias
Power Switch Insertion
PG Prototyping
Logical Connections
Via Master Rule
Mesh Pattern
Ring/Hard Macro/Std Cell Patt...
Special Pattern
PG Region
Strategy
Create PG
QoR Analysis
Power Network Analysis
Clock Trunk Planning
Overview
Manual Planning
Automatic Planning
Pin Assignment
Overview
Block Pin Constraints
Individual Pin Constraints
Bundle Pin Constraints
Console
Script Editor
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)
No Grid
XY 12.665, 10.611
No Selection
Power Rings
connect_pg_net -automatic -all_blocks
create_net -power {VDD}
create_net -ground {VSS}
Tcl
Tab Width: 8
Ln 38, Col 1
INS
"icc2.tcl" selected (4.0 kB)


Applications Places IC Compiler II - BlockWindow.1 - Design Planning - fa_icc2:full_adder.design [write]
Tue 14:39
Task Assistant - Design Planning
PG Planning->Strategy
39 of 70
Strategy name
core_ring
Routing area
Core
Design boundary
PG region
Hard macro
Polygon
Block
Voltage area
Enter pattern(s)
Pattern
{name : ring_pattern}{nets : {VDD VSS}}{offset : {0.5 0.5}}
Blockage
blockage : blockage option of this strategy
Extension
extension : extension option of this strategy
set_pg_strategy core_ring -core -pattern {{name : ring_pattern}{nets : {VDD VSS}}{offset : {0.5 0.5}}}
Apply
Script
Default
Help
0 PG strategies
Remove
Remove All
Close
Design Planning
PG Planning
Overview
Examples
Center Aligned Vias
Bridging Straps
Via Bridging
Tapering Straps
Checkerboard Vias
Power Switch Insertion
PG Prototyping
Logical Connections
Via Master Rule
Mesh Pattern
Ring/Hard Macro/Std Cell Pattern
Special Pattern
PG Region
Strategy
Create PG
QoR Analysis
Power Network Analysis
Clock Trunk Planning
Overview
Manual Planning
Automatic Planning
Pin Assignment
Overview
Block Pin Constraints
Individual Pin Constraints
Bundle Pin Constraints
Console
Script Editor
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)
No Grid
XY 10.761, 11.675
No Selection
Power Rings
connect_pg_net -automatic -all_blocks
create_net -power {VDD}
create_net -ground {VSS}
Tcl
Tab Width: 8
Ln 38, Col 1
INS
"icc2.tcl" selected (4.0 kB)

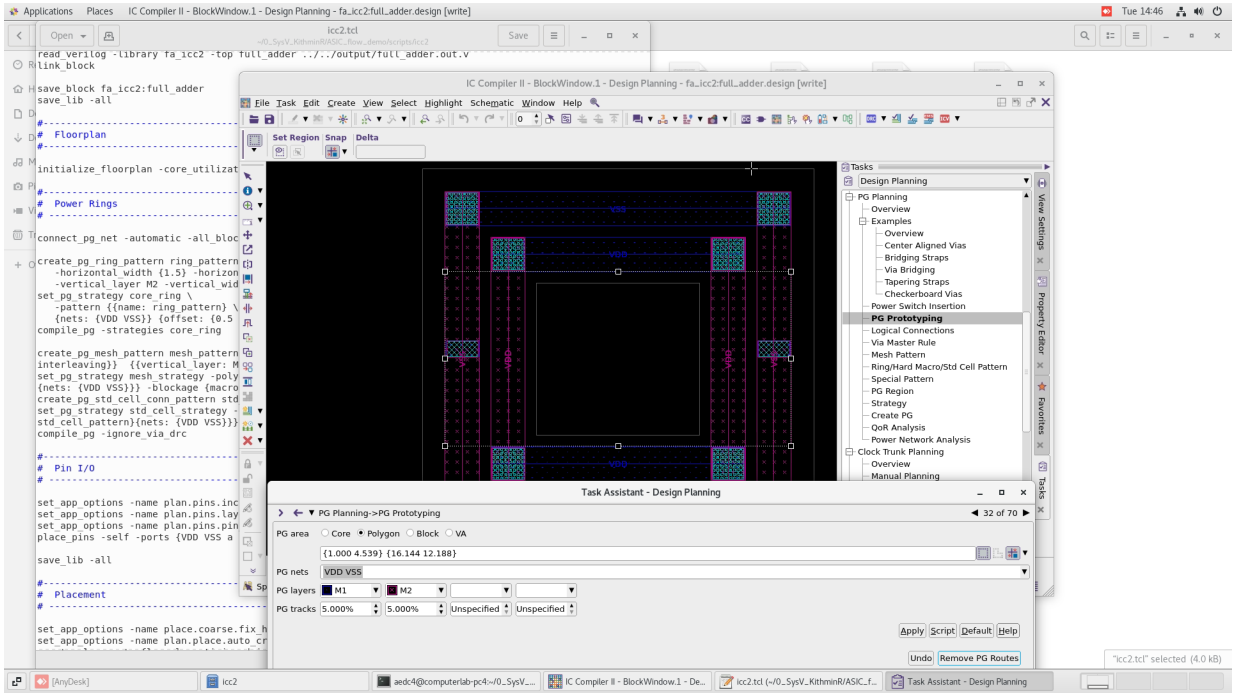


After successfully completing these above steps, you should be able to see the floor plan with the power rings as shown below. The cells have not been placed yet.

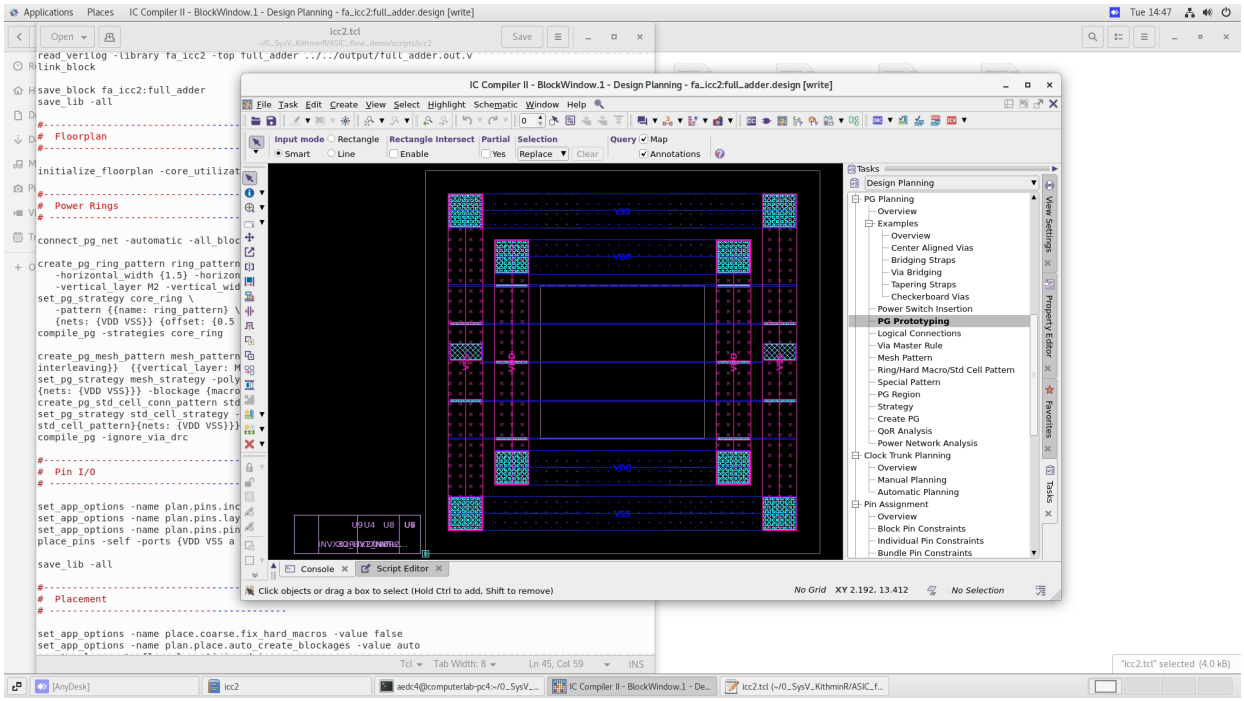
Note: Feel free to explore the different options in the settings given.



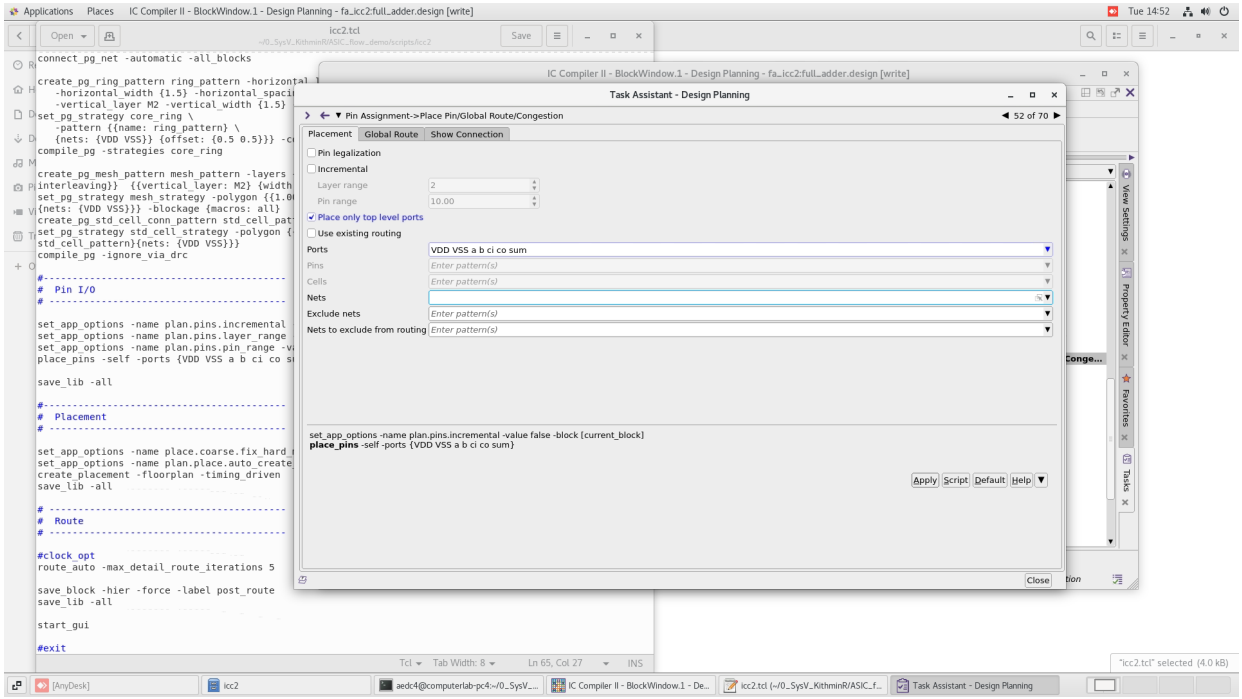
Next, you must start using the PG Prototyping option to do some special routing in between the VDD and VSS rings. Here we use the polygon option, so choose to route using this option. It will select the coordinate of the polygon you draw on the layout screen as shown below.



You should get something like this.

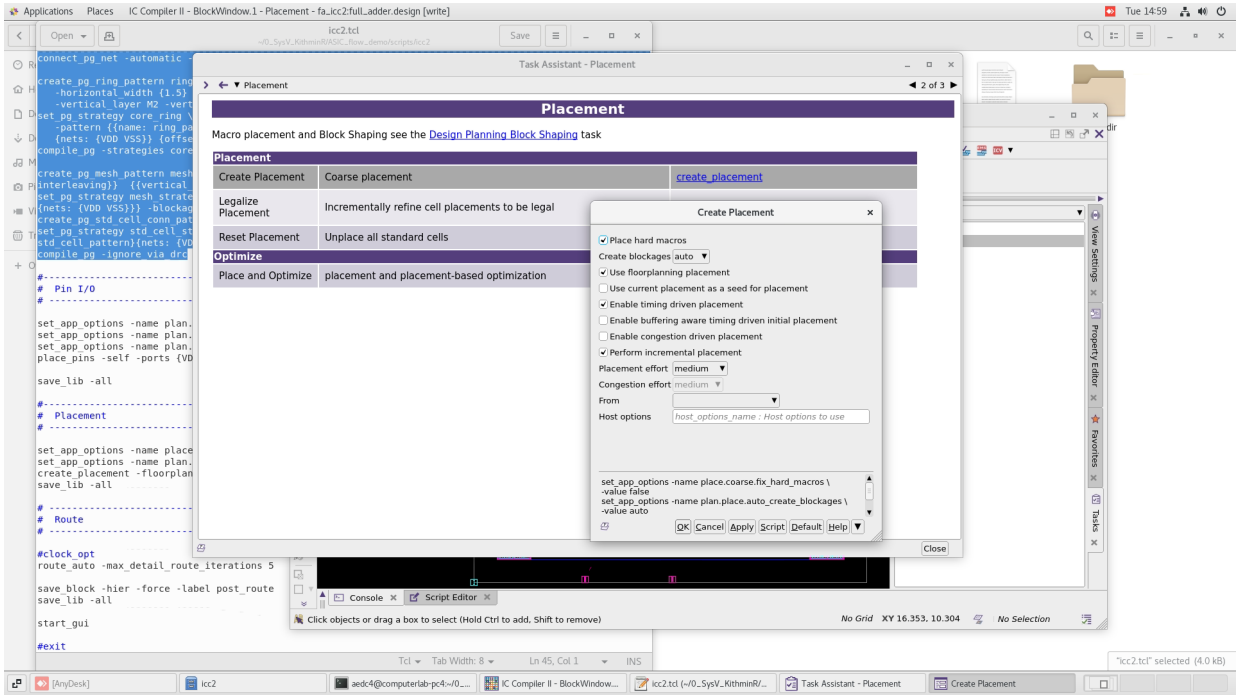


Finally, you can start the Pin Assignment.

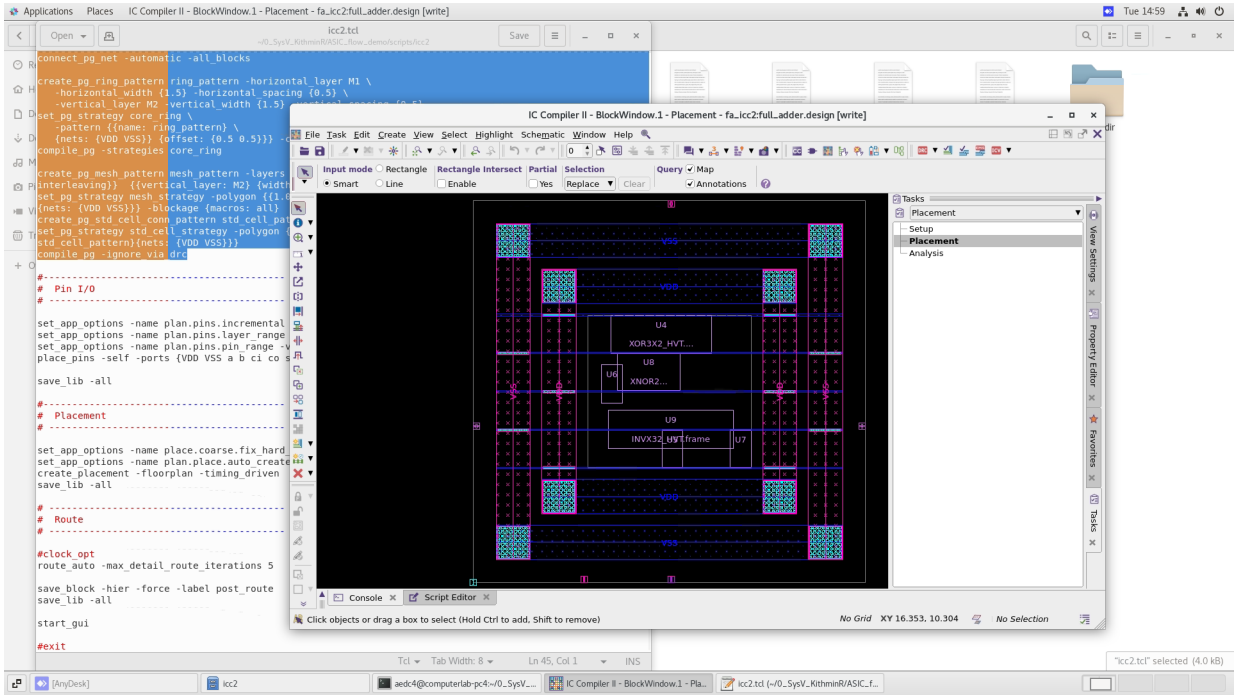


Placement Task

20. The last task before starting routing, is the placement task. You should select Placement from the Task menu in order to use this.

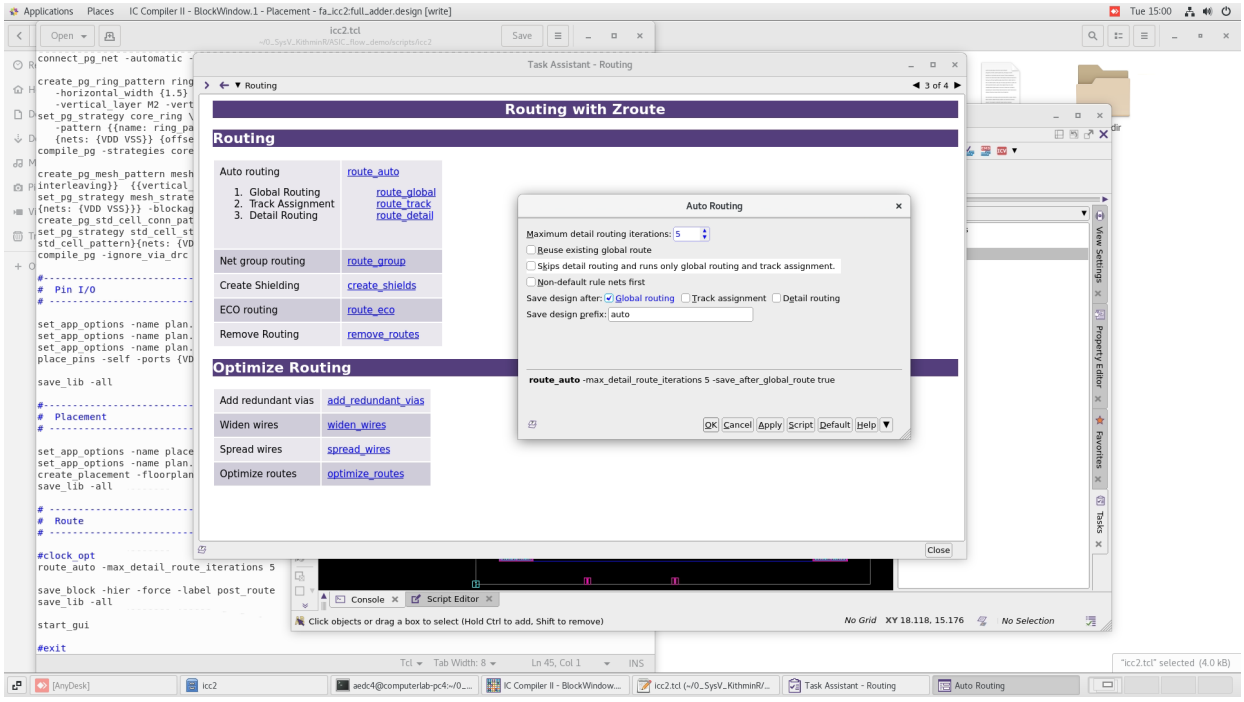


You will bee the cells get placed inside the core after you successfully complete this step.

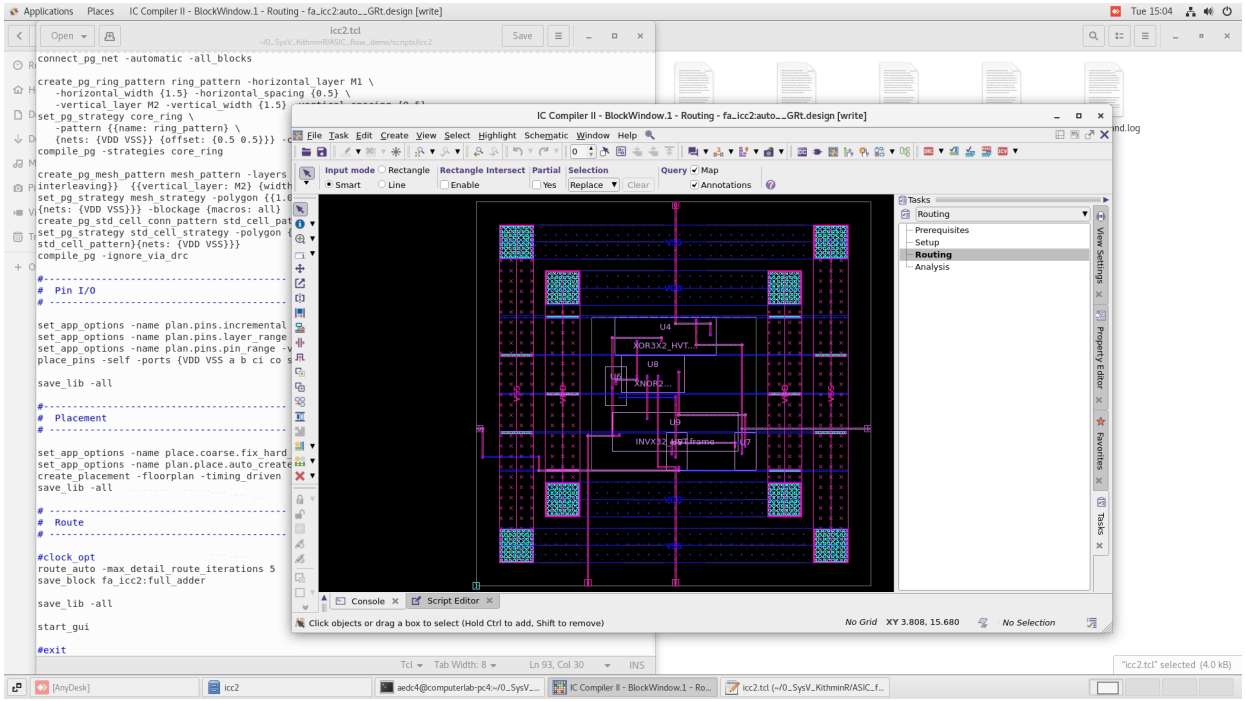


Routing Task

21. Last part is to do the routing task. We will use the auto routing option.



You will see the final layout look like this for our full_adder place and route flow.



The TCL Instruction File

To be filled (continue to the next section).

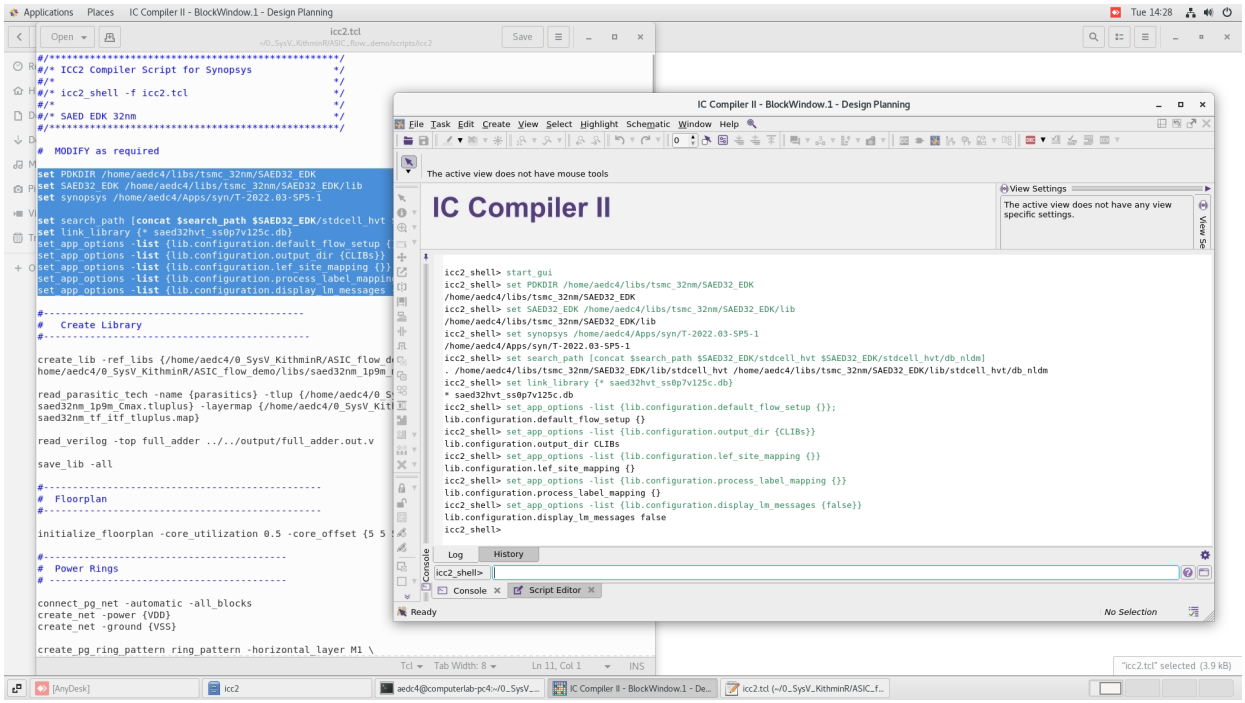
icc2.tcl

```
#/*****************************************************/
#/* ICC2 Compiler Script for Synopsys                 */
#/*                                                   */
#/* icc2_shell -f icc2.tcl                            */
#/*                                                   */
#/* SAED EDK 32nm                                     */
#/*****************************************************/

#  MODIFY as required

#/* Top-level Module                                  */
set top_module full_adder

#/* Library Name                                      */
set library_name fa_icc2

set PDKDIR /home/aedc4/libs/tsmc_32nm/SAED32_EDK
set SAED32_EDK /home/aedc4/libs/tsmc_32nm/SAED32_EDK/lib
set synopsys /home/aedc4/Apps/syn/T-2022.03-SP5-1

set search_path [concat $search_path $SAED32_EDK/stdcell_hvt
$SAED32_EDK/stdcell_hvt/db_nldm]
set link_library {* saed32hvt_ss0p7v125c.db}
set_app_options -list {lib.configuration.default_flow_setup {}};
set_app_options -list {lib.configuration.output_dir {CLIBs}}
set_app_options -list {lib.configuration.lef_site_mapping {}}
set_app_options -list {lib.configuration.process_label_mapping {}}
set_app_options -list {lib.configuration.display_lm_messages {false}}

#-----------------------------------------------
#   Create Library
#-----------------------------------------------
```

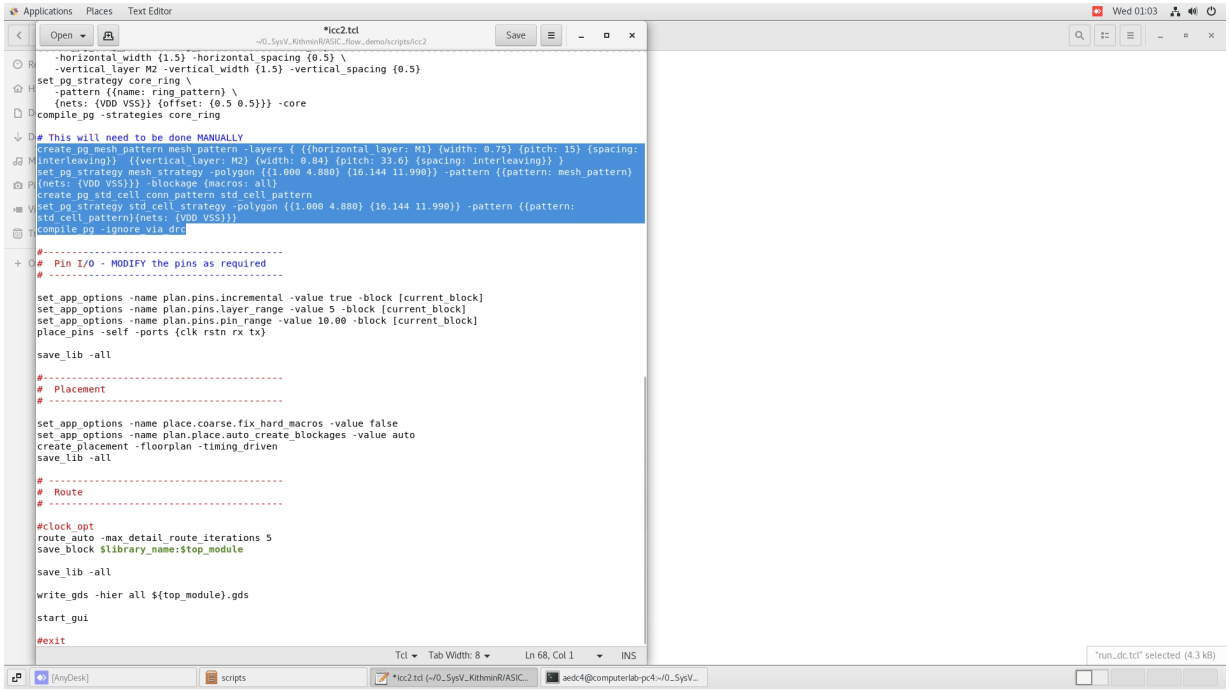
Running the TCL Instruction File from the Terminal (Line by Line)

22. You can copy cells line by line and execute them on the icc2_shell terminal as shown below.

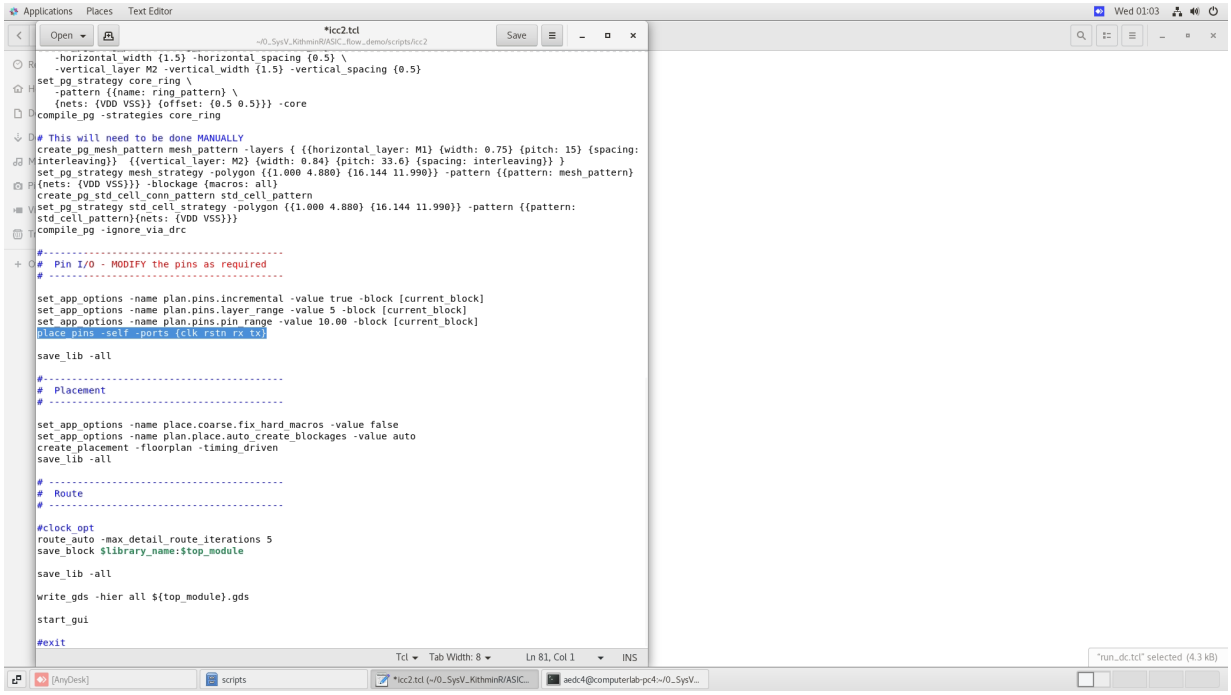
icc2_shell >



23. When you get to running the PG special routing part, make sure to complete this section manually for different cells, since it uses the polygon coordinate you draw on the layout screen. You can also change the polygon coordinate on the icc2.tcl script as well, if you plan to keep using the script to build the IC.



24. Place the 5 pins of the , either manually or modify the highlighted line appropriately.



25. Once you have made the above adjustments based on the machine you are using, you should be able to also run the entire icc2.tcl file from the terminal as well. You can use the command;

    $ icc2_shell -f icc2.tcl

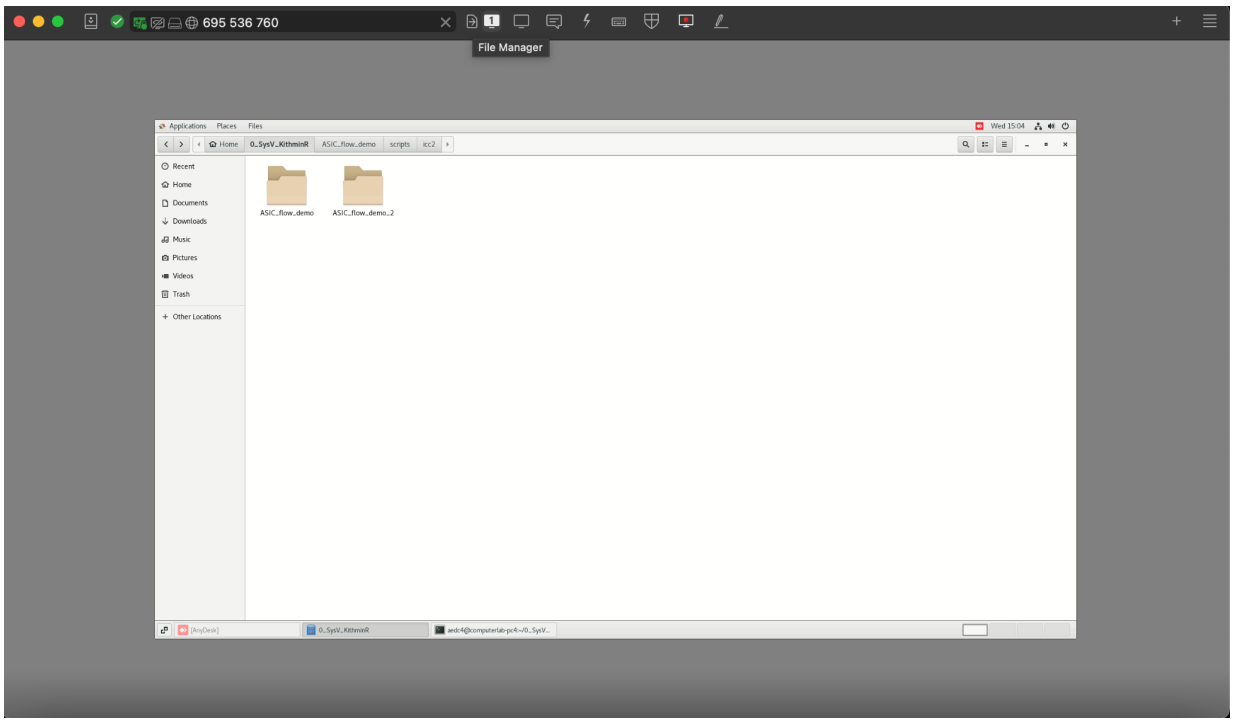
GDSII file export and visualization using kLayout (or GDS3D)

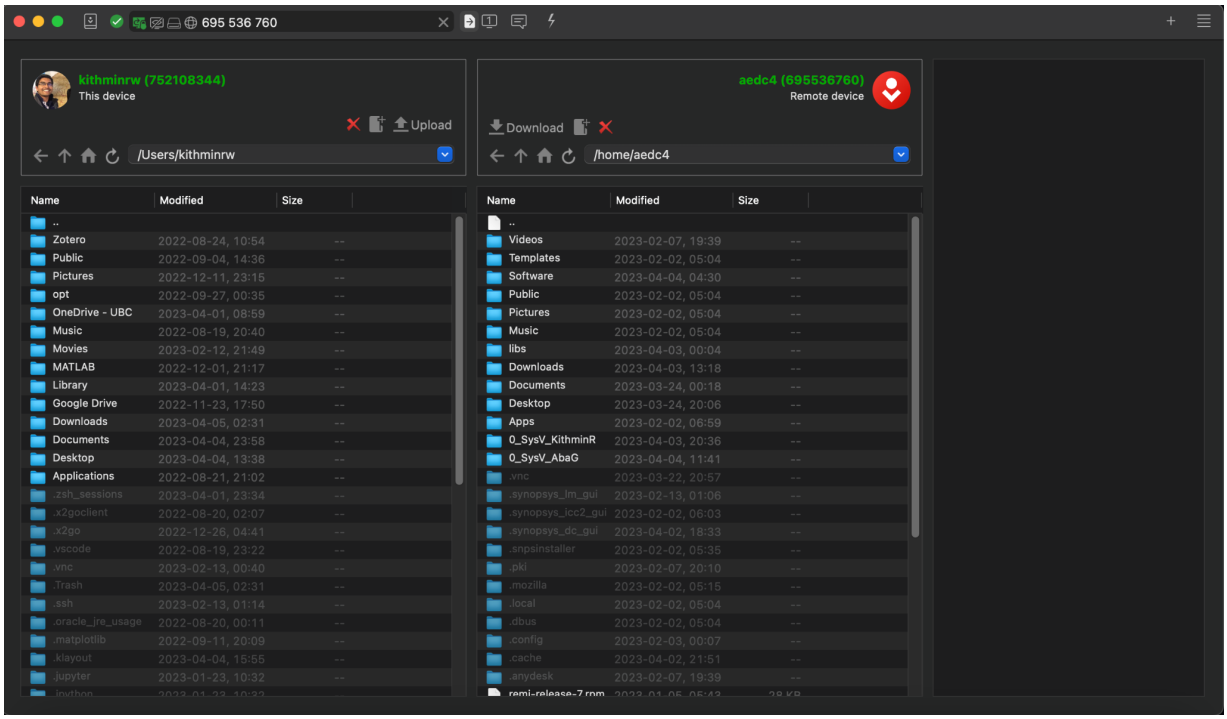
You can install the necessary software using the links given below.

kLayout - Download or Build Yourself

GDS3D - An application used for rendering IC (chip) layouts in 3D.

For users of AnyDesk, you can download the files back into your own computer as follows.





Although a comparatively simple piece of software, a layout **viewer** is not only just a tool for the chip design engineer. Today's design's complexity requires not only a simple "viewer". Rather, a **viewer** is the microscope through which the engineer looks at the design.

There are numerous viewers available, but sadly there are not many which satisfy a few basic requirements. Most of them are commercial and expensive. If there is a need for a simple, yet powerful viewer - here it is.

The main objective was to focus on the basic functionality but adding some useful features that many, even commercial viewers don't have.

First, rarely any tool allows two or even more layout files over each other. It often happens that you receive some layers in one file, the other layers in another. Some tools allow you to load multiple layouts and switch between the windows. Well, this may help - but still the possibility of overlaying two layouts offers much more comfort.

Sadly, almost no viewer is really precise. There is not much more annoying than a layout that changes when you zoom into it. Or placeholder shapes appearing at some zoom level and disappearing at the next, cell labels that cannot be caught because they jump around when you try to zoom them into view, and many other surprising ways or creative interpretation and optimization. This viewer shows the design as it is.

Only some viewers are allowed to make layers "transparent". Only this way, a stack of layers can be visualized effectively. In addition, this viewer can animate layers to make them blink or scroll the fill pattern. Animation is a good tool to highlight certain layers.

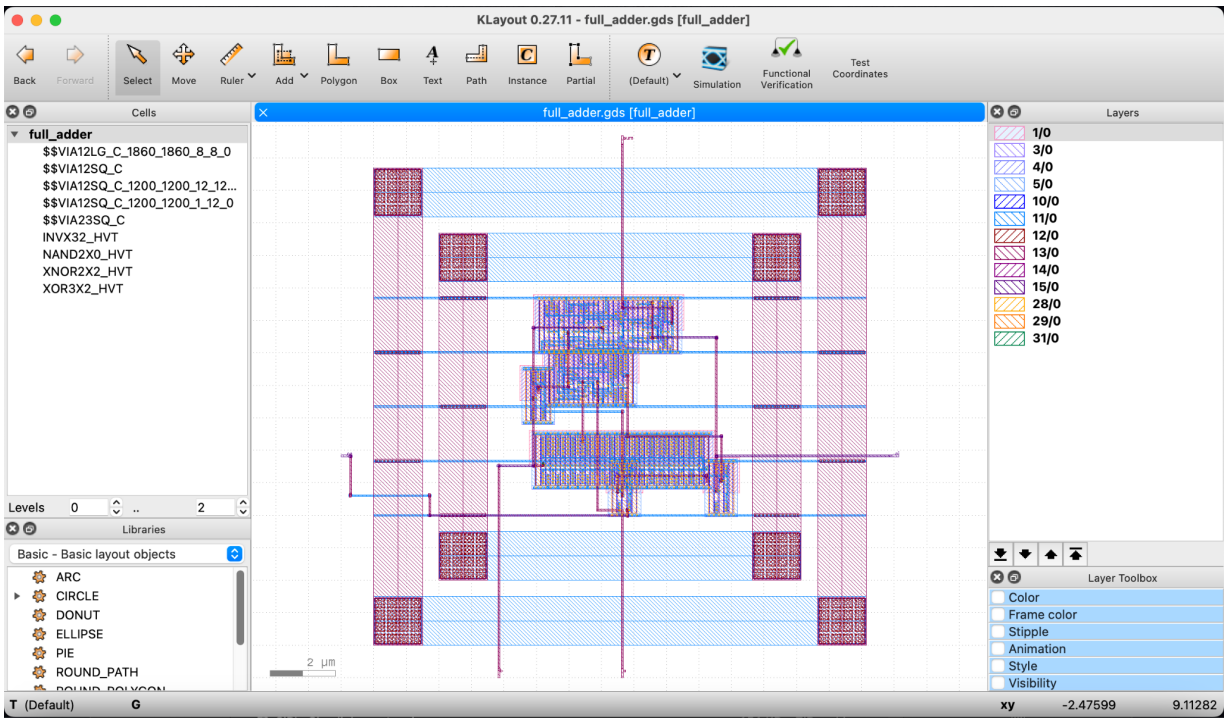
This viewer allows the user to display a layer "marked" by drawing a small cross on all shapes. There is no better way to visualize the distribution of a set of sparse error markers on a dense layout!

All comes wrapped in a nice, Qt based state of the art GUI. Usage of the viewer is simple and is similar to that of other tools.

Starting with version 0.15, KLayout is also an **editor** that allows users to change GDS and OASIS files and create them from scratch. See the release notes and the editor mode quickstart manual for a more detailed description. With version 0.22, parameterized cells became available in KLayout. See About PCell's for details about that feature.

In kLayout, you can follow these steps to change the scaling factor after copying and pasting the Layout views of the cells. First of all, deselect the first 4-5 layers which correspond to the metal interconnects and power rings etc. Next, select all the cells you want and press 'Q' for properties. In the properties window, you will see an option to increase the scaling factor from 1 to 10. Keep doing this for each of the cells, where the right arrow on the bottom will allow you to move from cell to cell.

The final GDSII file will look like this.



Sec 3 - Assignment 3

In this Assignment, you have to take the Matrix Vector Multiplier UART System final verilog code (https://github.com/SkillSurf/systemverilog/tree/master/rtl), and synthesize the RTL using the SAED 32nm EDK and Synopsys Design Compiler. Next, you must get the final layout after doing PnR using the Synopsys IC Compiler II. You are free to vary the 4 parameters R, C, W_X, W_K, but include the final selection of parameters in the submission (suggested to use {2,2,2,2}). Finally, you have to export the final .gds file and visualize it using kLayout (https://www.klayout.de/build.html).

Hints:

1. Copy all the .sv and .v files inside above github link into the "***input/rtl/***" folder
2. In, **run_dc.tcl**, change name of top module and uncomment to this line below during synthesis

```
define_design_lib WORK -path .template

# read RTL
#analyze -format sverilog [glob ${rtlPath}*.sv] > ../log/1.${top_module}_analyse.log
analyze -format verilog [glob ${rtlPath}*.v] > ../log/1.${top_module}_analyse.log
elaborate $top_module > ../log/2.${top_module}_elaborate.log
current_design $top_module
check_design > ../log/3.${top_module}_check_design.rpt

# Link Design
link
uniquify

# Default SDC Constraints (can be an sdc file)
set my_period 0.7
```

3. Open the mvm_uart_system.out.v file inside "***output/***" folder to make sure the file was synthesized correctly. You can check this on the DC Design Vision gui as well.

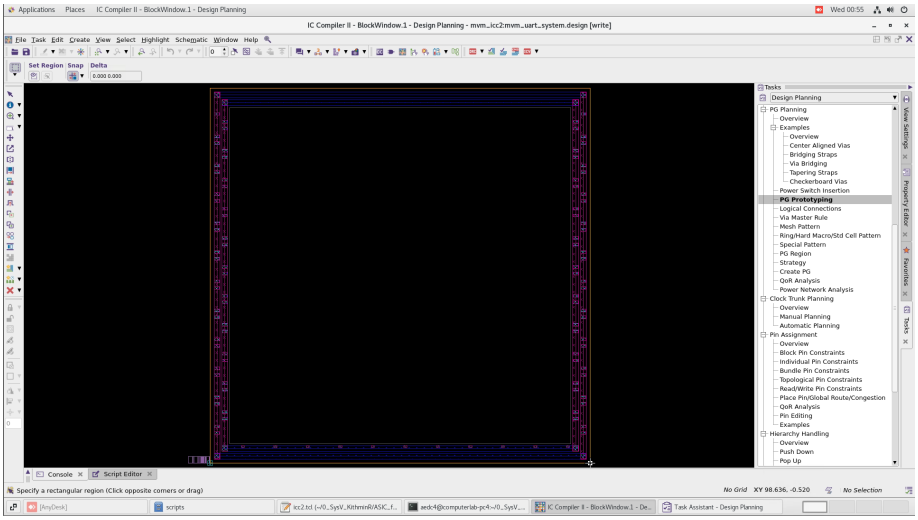
mvm_uart_system.out.v

```
////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : T-2022.03-SP5-1
// Date      : Wed Apr  5 00:46:48 2023
////////////////////////////////////////////////////////////


module mvm_uart_system ( clk, rstn, rx, tx );
  input clk, rstn, rx;
  output tx;
  wire   s_valid, s_ready, m_ready, m_valid, UART_RX_n140, UART_RX_n139,
         UART_RX_n138, UART_RX_n137, UART_RX_n136, UART_RX_n135, UART_RX_n134,
         UART_RX_n133, UART_RX_n132, UART_RX_n131, UART_RX_n130, UART_RX_n129,
         UART_RX_n128, UART_RX_n127, UART_RX_n126, UART_RX_n125, UART_RX_n124,
         UART_RX_n123, UART_RX_n122, UART_RX_n121, UART_RX_n120, UART_RX_n119,
         UART_RX_n118, UART_RX_n117, UART_RX_n116, UART_RX_n115, UART_RX_n114,
         UART_RX_n113, UART_RX_n112, UART_RX_n111, UART_RX_n110, UART_RX_n109,
         UART_RX_n108, UART_RX_n107, UART_RX_n106, UART_RX_n105, UART_RX_n104,
         UART_RX_n103, UART_RX_n102, UART_RX_n101, UART_RX_n87, UART_RX_n86,
         UART_RX_n85, UART_RX_n84, UART_RX_n83, UART_RX_n82, UART_RX_n81,
         UART_RX_n80, UART_RX_n79, UART_RX_n78, UART_RX_n77, UART_RX_n76,
         UART_RX_n75, UART_RX_n74, UART_RX_n73, UART_RX_n72, UART_RX_n71,
         UART_RX_n70, UART_RX_n69, UART_RX_n68, UART_RX_n67, UART_RX_n65,
         UART_RX_n64, UART_RX_n63, UART_RX_n62, UART_RX_n61, UART_RX_n60,
         UART_RX_n59, UART_RX_n58, UART_RX_n57, UART_RX_n56, UART_RX_n55,
         UART_RX_n54, UART_RX_n53, UART_RX_n52, UART_RX_n51, UART_RX_n50,
         UART_RX_n49, UART_RX_n48, UART_RX_n47, UART_RX_n46, UART_RX_n45,
         UART_RX_n44, UART_RX_n43, UART_RX_n42, UART_RX_n41, UART_RX_n40,
         UART_RX_n39, UART_RX_n38, UART_RX_n37, UART_RX_n27, UART_RX_n25,
         UART_RX_n24, UART_RX_n23, UART_RX_n22, UART_RX_n21, UART_RX_n20,
         UART_RX_n19, UART_RX_n6, UART_RX_n3, UART_RX_n2, UART_RX_n1,
         UART_RX_N216, UART_RX_N99, UART_RX_N98, UART_RX_N97, UART_RX_N96,
         UART_RX_N95, UART_RX_N94, UART_RX_N93, UART_RX_N92, UART_RX_N91,
         UART_RX_N90, UART_RX_N89, UART_RX_N88, UART_RX_N87, UART_RX_N86,
```

4. In **icc2.tcl**, change name of top module and the library
5. In **icc2.tcl**, this section of the script needs to be done manually

```
# This will need to be done MANUALLY
create_pg_mesh_pattern mesh_pattern -layers { {{horizontal_layer: M1} {width: 0.75} {pitch: 15} {spacing:
interleaving}}  {{vertical_layer: M2} {width: 0.84} {pitch: 33.6} {spacing: interleaving}} }
set_pg_strategy mesh_strategy -polygon {{1.000 4.880} {16.144 11.990}} -pattern {{pattern: mesh_pattern}
{nets: {VDD VSS}}} -blockage {macros: all}
create_pg_std_cell_conn_pattern std_cell_pattern
set_pg_strategy std_cell_strategy -polygon {{1.000 4.880} {16.144 11.990}} -pattern {{pattern:
std_cell_pattern}{nets: {VDD VSS}}}
compile_pg -ignore_via_drc

#-----------------------------------------
```

   a. Set the two PG tracks (M1, M2) to 20%, and chose the entire layout area for the polygon size

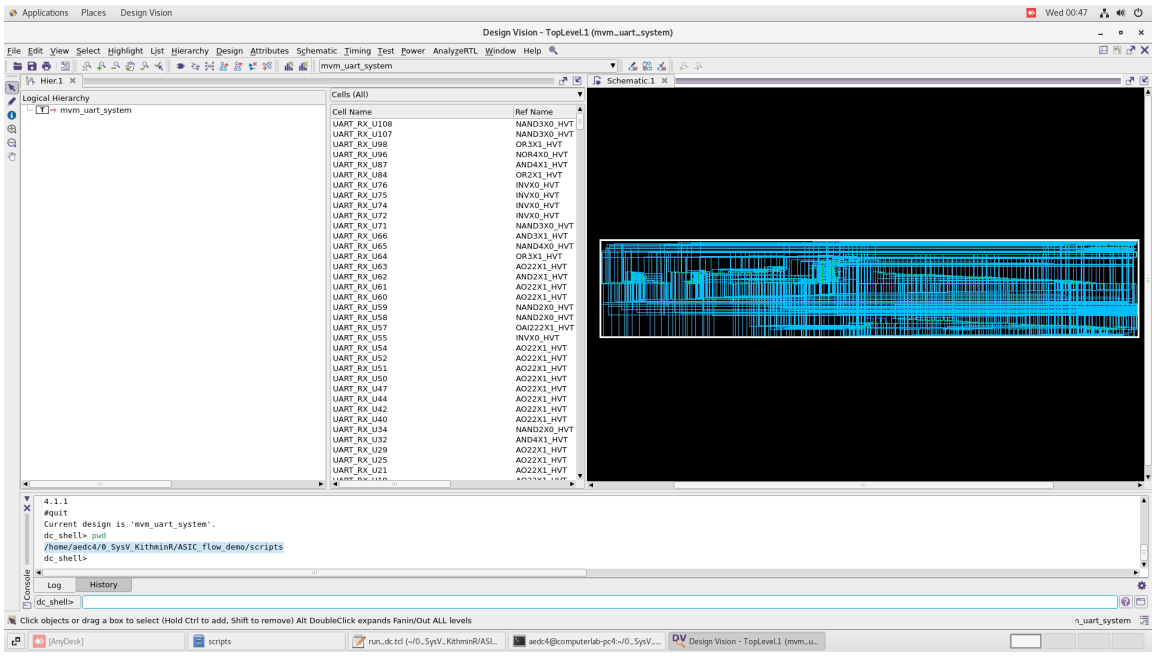


6. In **icc2.tcl**, adjust the pin names during PnR

```
#----------------------------------------
#  Pin I/O - MODIFY the pins as required
# ----------------------------------------

set_app_options -name plan.pins.incremental -value true -block [current_block]
set_app_options -name plan.pins.layer_range -value 5 -block [current_block]
set_app_options -name plan.pins.pin_range -value 10.00 -block [current_block]
place_pins -self -ports {clk rstn rx tx}

save_lib -all

#----------------------------------------
```

Submit a PDF report with 3 screenshots of the different sections in ASIC flow covering;

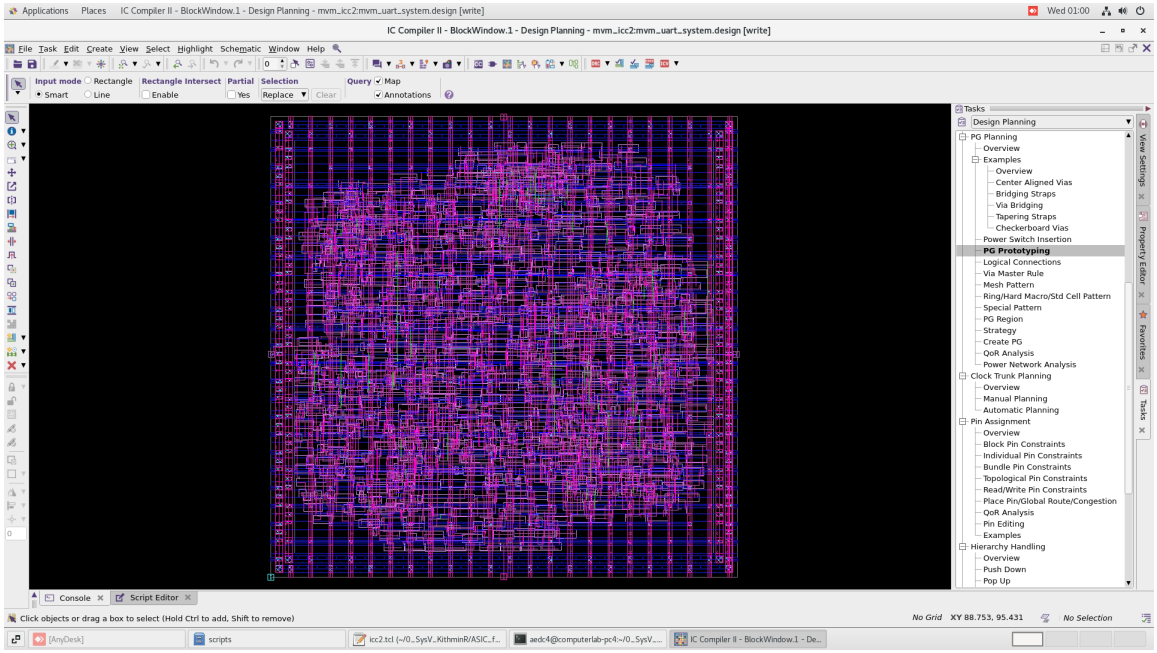
1. The synthesized RTL design schematic visualized in Design Vision of DC (example shown below) - use the 'pwd' command on the terminal to show the directory you are inside, which should have **0_SysV_<YourName>** in the name.



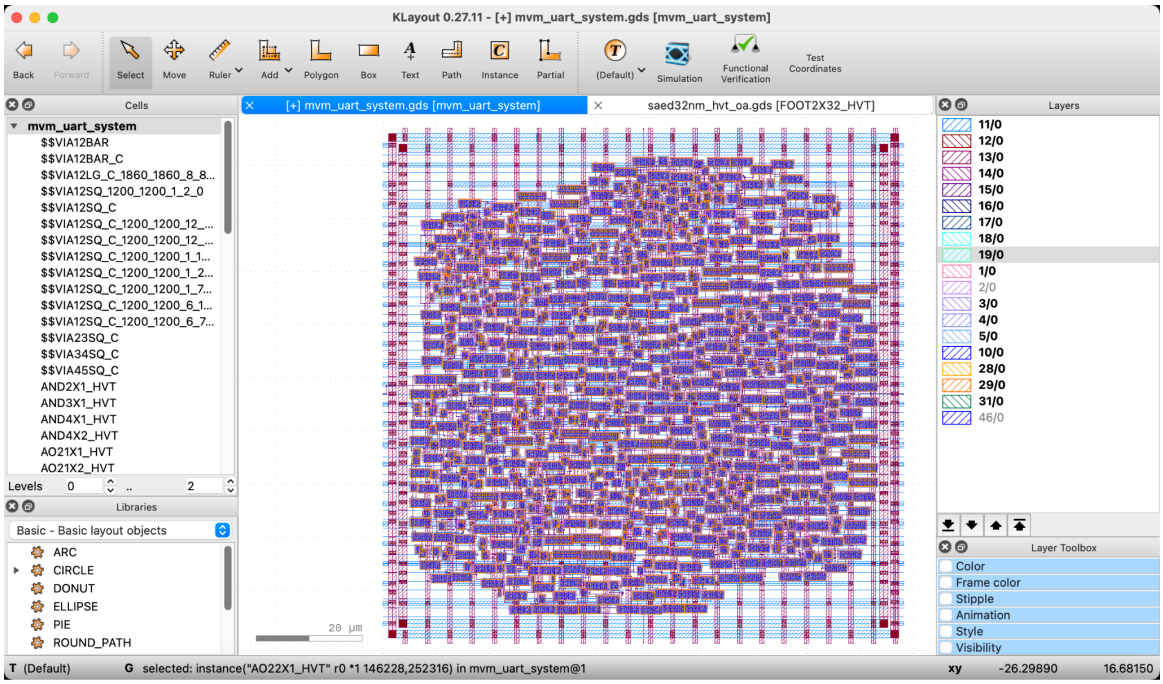
2. Go inside the **report**/ folder and find the different reports generated during the synthesis process of the mvm_uart_system. Each of the reports contain information about
    a. Ports
    b. Area
    c. Power
    d. Timing
    e. No. of Cells

    Include the information from these reports in your report as well.

3. The completed PnR layout visualized inside ICC2 (example shown below) - use the 'pwd' command on the terminal to show the directory you are inside, which should have **0_SysV_<YourName>** in the name.



4. The final exported GDSII file visualized using kLayout (example shown below) - this is an optional part of the assignment.



Useful Resources:

https://github.com/kunalg123/icc2_workshop_collaterals