Logic Synthesis

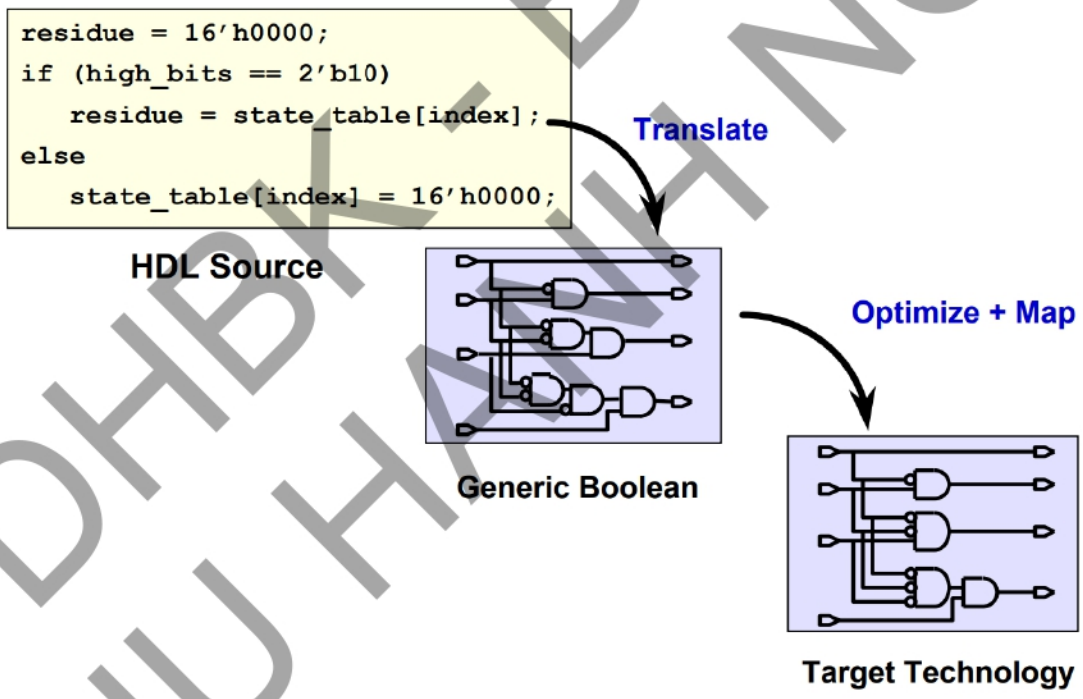
Khai Pham

1. *Introduction:*

What is Synthesis?

Synthesis is the process of transforming the HDL high level design code (synthesizable Verilog/Systemverilog) into a technology-specific gate-level netlist (includes nets, sequential and combinational cells, and their connectivity), optimized for a set of pre-defined constraints and optimization settings.

In other words, synthesis is a process of combining pre-existing elements - "Technologies library" which has all the physical and timing information of standard cells, to form the functional circuitry and meet the design specifications from the specified HDL design code.



By doing this, we can translate ideal RTL into an implementation of the targeted technology.

The synthesis inputs:

- RTL, Technology libraries, constraints (Environment, clocks, IO external delays, etc).

The synthesis outputs:

- Netlist, SDF - Standard Delay Format files, Reports, etc.

The ***Design Compiler (DC)*** from Synopsys and ***Genus synthesis*** or ***RTL compiler*** from Cadence are the tools widely used for Synthesis.

Logic Synthesis is described as translation with the addition of logic optimization and mapping. In-terms, if the tools translation is performed during reading and elaborating the design then Logic optimization and mapping are performed by the synthesis command.

2. *Synthesis flow*

Logic Synthesis Goals:

- To get a Gate-level Netlist
- Logic optimization that's fit the constraints.
- The logical equivalent between RTL and Netlist should be maintained after synthesis.

Synthesis Flow in genus synthesis:

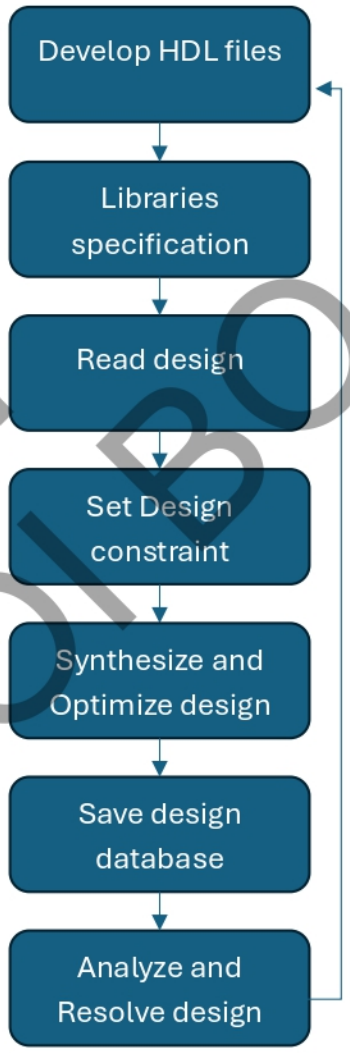
Below is the simple script for synthesis in genus synthesis:

```
################################################
#
#Khai Pham
#HCMUT LAB 203
#
################################################
#file setup variables#
set top_module synth_wrapper
#set_db auto_ungroup none
################################################
#Libraries setup#
set_db library [list /opt/PDKs/skywater130/timing/sky130_fd_sc_hd__tt_025C_1v80.lib ]
set_db lef_library [ list /opt/PDKs/skywater130/tech/sky130_fd_sc_hd__nom.tlef ]
################################################
#HDL files read#
read_hdl -sv -f 00_src/flist.f
################################################
elaborate
set_top_module ${top_module}
write_hdl > 03_synth/${top_module}_elab.v
check_design -all
################################################
#Timing constraint variables#
set FREQ_GHz 1.1
```

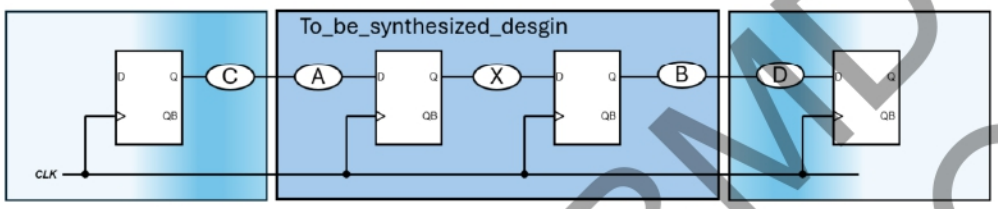
```
set FREQ [ expr ${FREQ_GHz} * 1000000000.0 ]
set PERIOD_tmp [ expr (1.0/${FREQ}) ]
set PERIOD [ expr ${PERIOD_tmp} * 1000000000000.0 ]
puts "Clock Period = ${PERIOD} ps"
set IN_DLY [ expr $PERIOD/2.0 ]
set OUT_DLY [ expr $PERIOD/2.0 ]
set UNCER [expr ${PERIOD}/100000.0]
puts "Clock Uncertainty = ${UNCER} ns"
set TRANS 1.5
puts "max transition = ${TRANS} ns"
################################################
#set constraint for clock and input with output#
set clock [define_clock -period $PERIOD -name clk [clock_ports] ]
set_dont_touch_network [get_clocks clk]
set_clock_uncertainty ${UNCER} -setup -hold clk
set_max_transition ${TRANS} [get_design ${top_module}]
external_delay -clock clk -input $IN_DLY  -name delay_in [all_inputs]
external_delay -clock clk -output $OUT_DLY -name delay_out [all_outputs]
################################################
#Synthesize RTL code to generic#
syn_generic
write_hdl > 03_synth/${top_module}_generic.v
#Mapping technology to the generic#
syn_map
write_hdl > 03_synth/${top_module}_tech_map.v
#remove assignment statement
remove_assigns_without_optimization -verbose
#Optimizing the mapped technology netlist#
syn_opt

report timing -lint
################################################
#design post-synthesis export#
#Export the netlist
write_hdl > 03_synth/${top_module}_gate.v
#Export the standard delay format of the synthesized design #
write_sdf -edges check_edge -setuphold "split" -recrem split > 03_synth/${top_module}.sdf

################################################
#Reports export#
report timing -max_paths 10 > 04_reports/${top_module}.timing.rpt
report hierarchy > 04_reports/${top_module}.hier.rpt
report gates > 04_reports/${top_module}.gates.rpt
report datapath > 04_reports/${top_module}.datapath.rpt
report qor > 04_reports/${top_module}.qor.rpt
report area > 04_reports/${top_module}.area.rpt
report power > 04_reports/${top_module}.power.rpt
################################################
#Open genus GUI
gui_show
```

Stage Develop HDL files:

- This stage is to have the input design files ready for design compiler, usually written in Hardware Description Language (HDL) such as Verilog/Systemverilog or VHDL.
- Designs should be checked and linted for ***synthesizability*** in this stage.



Note: Before synthesis we want to put the register after/before the input / output of the top design so that we can apply constraint on the **reg-2-reg** path of the design.



Stage Libraries specification:

- The genus synthesis uses technology process libraries (PDKs-Process Design Kit provided by foundries) to Implement the design that meet the requirement of the specification of the architect. The PDK library will contain all the physical properties of the required devices that will be implemented.

```
set_db library [list
/opt/PDKs/skywater130/timing/sky130_fd_sc_hd__tt_025C_1v80.lib
   ]
```

- We use the command set_db library to point the design to the liberty file we want to implement to our HDL design.

Stage Read design:

- Analyze the RTL design by reading the design and organize the design hierarchy and then elaborate the design with specified top design.

```
read_hdl -sv -f 00_src/flist.f
```

- For other HDL file format such as Verilog we can use option ***-v2001*** and ***-vhdl*** for VHDL.
- ***flist.f*** is the file that contains all the path to the RTL files, we also can list all the RTL files to the read_hdl command.

```
read_hdl -sv { 00_src/module_a.v 00_src/module_b.v 00_src/module_c.v }
```

- But using ***-f flist.f*** option is easier to manage and maintain.

Stage Set Design constraint:

- In this stage, we are trying to implement the design with specified criteria that the designer wanted to constraints aspects of the synthesized design, in this case will be timing constraints.
- For timing we specify clock period for clock port and uncertainty.
- We specified the external input delay and external output delay, we also specify the maximum transition the design.

```
################################################
#set constraint for clock and input with output#
set clock [define_clock -period $PERIOD -name clk [clock_ports] ]
set_dont_touch_network [get_clocks clk]
set_clock_uncertainty ${UNCER} -setup -hold clk
set_max_transition ${TRANS} [get_design ${top_module}]
external_delay -clock clk -input $IN_DLY  -name delay_in [all_inputs]
external_delay -clock clk -output $OUT_DLY -name delay_out [all_outputs]
################################################
```

- The final constraint is area, if the timing is met but the design area of the block is too large then it would not meet the designer requirement, the design will need to reduce the area by alternating the RTL architecture and still retain the function of the design.

Stage Synthesize and Optimize design:

- Design optimization will be carried out by synthesis-tools. After mapping the technology process on to the generic netlist the synthesis tools will try to optimize the netlist with mapped gates from the PDK library so that it can meet the expectations of the designer both timing and area.
- Also, we have an option to eliminate assign statement after we get the netlist by adding ***remove_assigns_without_optimization*** to the scripts so that netlist will only contain instantiated cell and no assign statement.

```
################################################
#Synthesize RTL code to generic#
syn_generic
write_hdl > 03_synth/${top_module}_generic.v
#Mapping technology to the generic#
syn_map
write_hdl > 03_synth/${top_module}_tech_map.v
#remove assignment statement
remove_assigns_without_optimization -verbose
#Optimizing the mapped technology netlist#
syn_opt

report timing -lint
```

```
################################################
```

Stage Save design database:

- In this stage, we will save the synthesized netlist of the design, also the SDF file which contains the delay information of each instance inside the netlist.

```
#design post-synthesis export#
#Export the netlist
write_hdl > 03_synth/${top_module}_gate.v
#Export the standard delay format of the synthesized design #
write_sdf -edges check_edge -setuphold "split" -recrem split > 03_synth/${top_module}.sdf
```

Stage Analyze and Resolve design problems:

- We would get all the reports from the synthesis process of the design in this stage.

```
#Reports export#
report timing -max_paths 10 > 04_reports/${top_module}.timing.rpt
report hierarchy > 04_reports/${top_module}.hier.rpt
report gates > 04_reports/${top_module}.gates.rpt
report datapath > 04_reports/${top_module}.datapath.rpt
report qor > 04_reports/${top_module}.qor.rpt
report area > 04_reports/${top_module}.area.rpt
report power > 04_reports/${top_module}.power.rpt
```

- Check if any timing paths of the design violate the timing resources specified by the designer constraint.
- In this stage we will try to avoid VIOLATE on the 10 worst timing-paths reports by synthesis-tools.

Below is an example of a simple synthesis of the 4-bit and operator:

Verilog Code for simple 4-bit "OR" operator

```
module simple_lab
(
   input  wire clk,
   input  wire rstn,
   output reg [3:0] out,
   input  wire [3:0]  ina, inb
);
   reg [3:0] ina_reg;
   reg [3:0] inb_reg;
   reg [3:0] out_reg;
   always @(posedge clk, negedge rstn) begin
     if(!rstn) begin
        ina_reg <= 4'h0;
        inb_reg <= 4'h0;
        out <= 4'h0;
     end
```

```
    else begin
       ina_reg <= ina;
       inb_reg <= inb;
       out <= out_reg;
    end
  end
  always@* begin
     out_reg = ina_reg | inb_reg ;
  end

endmodule
```

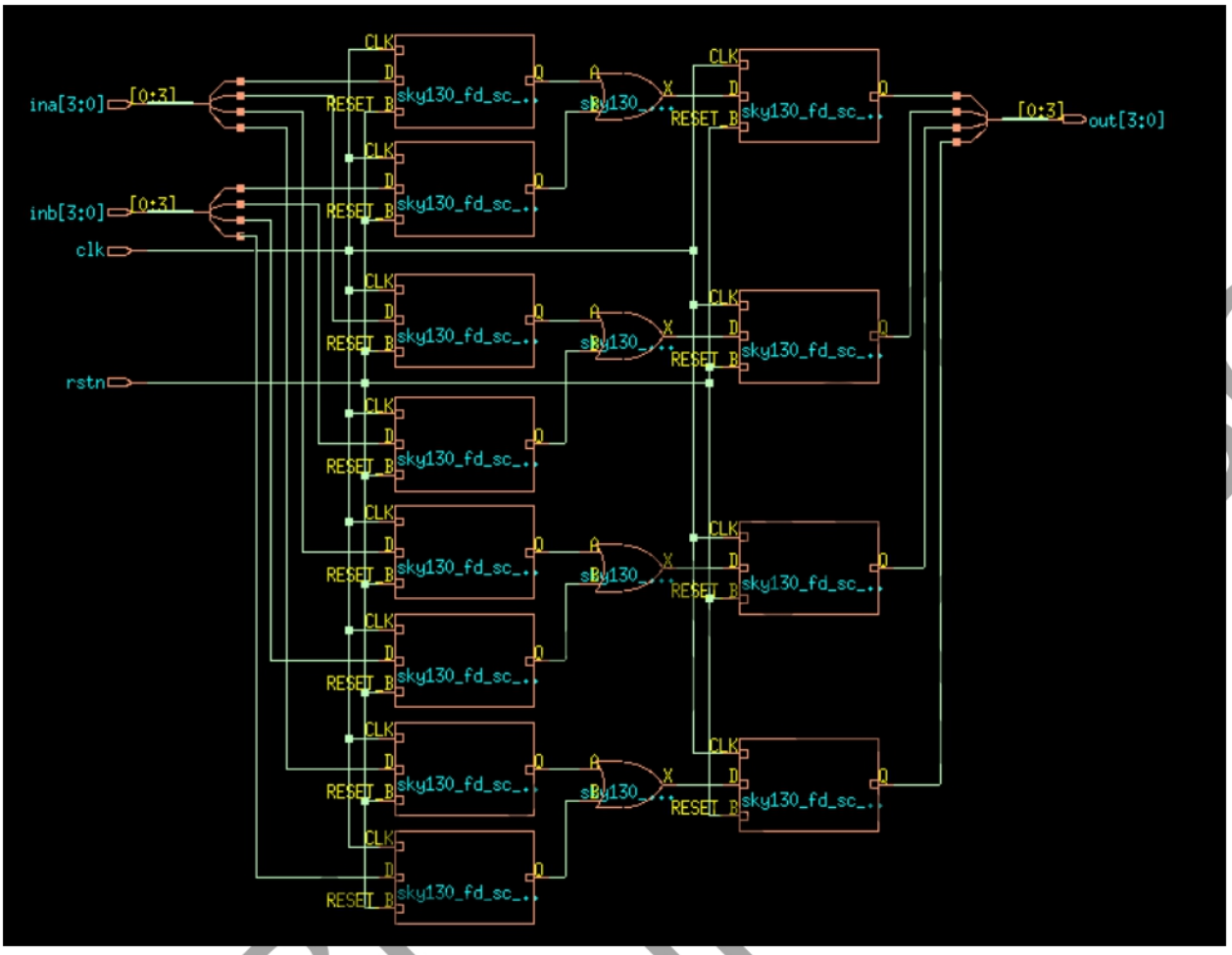
The RTL code of “or” function above is described in behavioral coding-style, this design file will be one of the inputs in the synthesis process.

The synthesized RTL design (Netlist)

```
module simple_lab(clk, rstn, out, ina, inb);

  input clk, rstn;

  input [3:0] ina, inb;

  output [3:0] out;

  wire clk, rstn;

  wire [3:0] ina, inb;

  wire [3:0] out;

  wire [3:0] ina_reg;

  wire [3:0] inb_reg;

  wire n_0, n_1, n_2, n_3;

  sky130_fd_sc_hd__dfrtp_1 \out_reg[3] (.RESET_B (rstn), .CLK (clk), .D

       (n_3), .Q (out[3]));

  sky130_fd_sc_hd__dfrtp_1 \out_reg[2] (.RESET_B (rstn), .CLK (clk), .D

       (n_0), .Q (out[2]));

  sky130_fd_sc_hd__dfrtp_1 \out_reg[0] (.RESET_B (rstn), .CLK (clk), .D

       (n_1), .Q (out[0]));

  sky130_fd_sc_hd__dfrtp_1 \out_reg[1] (.RESET_B (rstn), .CLK (clk), .D

       (n_2), .Q (out[1]));

  sky130_fd_sc_hd__or2_0 g41(.A (ina_reg[3]), .B (inb_reg[3]), .X

       (n_3));

  sky130_fd_sc_hd__or2_0 g43(.A (ina_reg[1]), .B (inb_reg[1]), .X

       (n_2));
```

```
  sky130_fd_sc_hd__or2_0 g42(.A (ina_reg[0]), .B (inb_reg[0]), .X
       (n_1));
  sky130_fd_sc_hd__or2_0 g44(.A (ina_reg[2]), .B (inb_reg[2]), .X
       (n_0));
  sky130_fd_sc_hd__dfrtp_1 \ina_reg_reg[0] (.RESET_B (rstn), .CLK
       (clk), .D (ina[0]), .Q (ina_reg[0]));
  sky130_fd_sc_hd__dfrtp_1 \ina_reg_reg[1] (.RESET_B (rstn), .CLK
       (clk), .D (ina[1]), .Q (ina_reg[1]));
  sky130_fd_sc_hd__dfrtp_1 \ina_reg_reg[2] (.RESET_B (rstn), .CLK
       (clk), .D (ina[2]), .Q (ina_reg[2]));
  sky130_fd_sc_hd__dfrtp_1 \ina_reg_reg[3] (.RESET_B (rstn), .CLK
       (clk), .D (ina[3]), .Q (ina_reg[3]));
  sky130_fd_sc_hd__dfrtp_1 \inb_reg_reg[0] (.RESET_B (rstn), .CLK
       (clk), .D (inb[0]), .Q (inb_reg[0]));
  sky130_fd_sc_hd__dfrtp_1 \inb_reg_reg[1] (.RESET_B (rstn), .CLK
       (clk), .D (inb[1]), .Q (inb_reg[1]));
  sky130_fd_sc_hd__dfrtp_1 \inb_reg_reg[2] (.RESET_B (rstn), .CLK
       (clk), .D (inb[2]), .Q (inb_reg[2]));
  sky130_fd_sc_hd__dfrtp_1 \inb_reg_reg[3] (.RESET_B (rstn), .CLK
       (clk), .D (inb[3]), .Q (inb_reg[3]));
endmodule
```

Graphical view of the netlist after synthesis



Timing report from the design

```
with Frequency at:
set FREQ_GHz 1.0
04_reports/simple_lab.timing.rpt:Path 1: MET (184 ps) Late External Delay Assertion at pin out[0]
04_reports/simple_lab.timing.rpt:Path 2: MET (184 ps) Late External Delay Assertion at pin out[1]
04_reports/simple_lab.timing.rpt:Path 3: MET (184 ps) Late External Delay Assertion at pin out[2]
04_reports/simple_lab.timing.rpt:Path 4: MET (184 ps) Late External Delay Assertion at pin out[3]
04_reports/simple_lab.timing.rpt:Path 5: MET (334 ps) Setup Check with Pin out_reg[1]/CLK->D
04_reports/simple_lab.timing.rpt:Path 6: MET (334 ps) Setup Check with Pin out_reg[0]/CLK->D
04_reports/simple_lab.timing.rpt:Path 7: MET (334 ps) Setup Check with Pin out_reg[2]/CLK->D
04_reports/simple_lab.timing.rpt:Path 8: MET (334 ps) Setup Check with Pin out_reg[3]/CLK->D
04_reports/simple_lab.timing.rpt:Path 9: MET (391 ps) Setup Check with Pin inb_reg_reg[3]/CLK->D
04_reports/simple_lab.timing.rpt:Path 10: MET (391 ps) Setup Check with Pin inb_reg_reg[2]/CLK->D
```

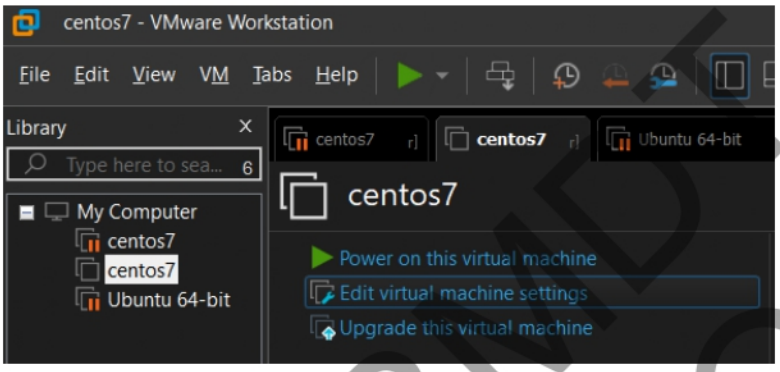
3. *Prerequisite knowledge to do labs.*

Before we can use the ***synthesis.zip*** package, we need to create a link between your Window folder and the virtual machine folder through sharing-folder method.

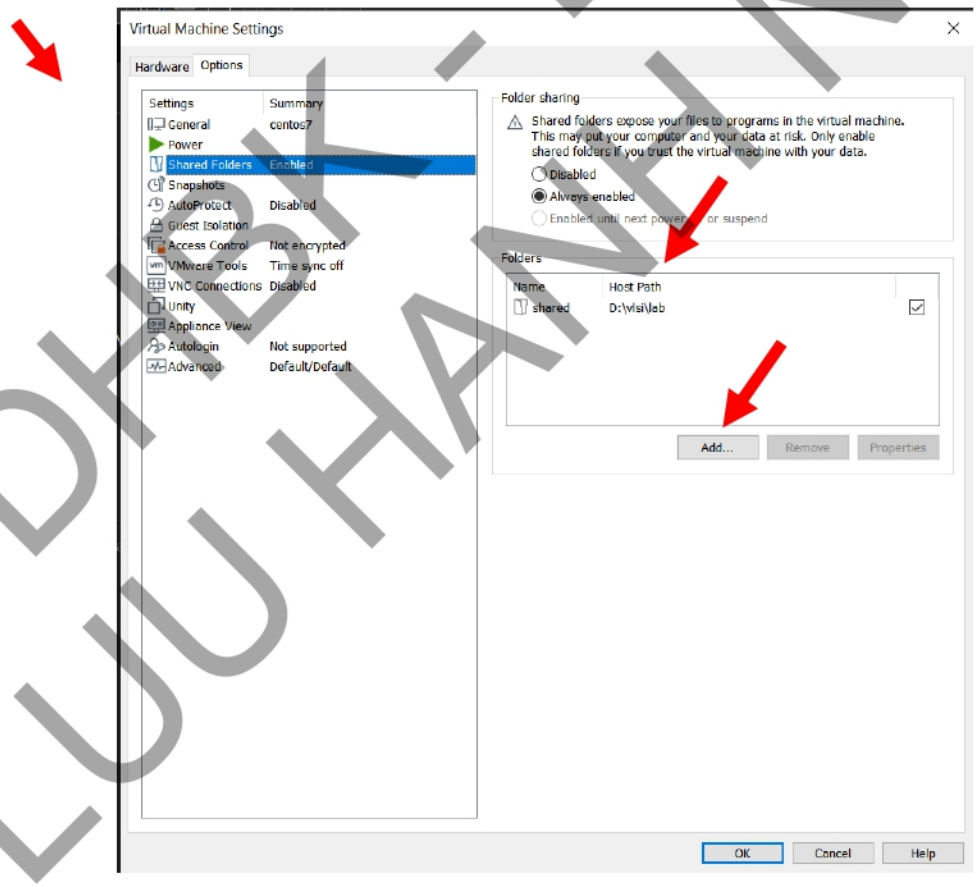
The reason to do this is to account for unsaved work to be lost if the virtual machine ran into problems. Virtual machines will be saved when it is power-off properly, sudden crash or unintentionally shutdown the vmware-player would cause all data lost.

Note: Some of knowledge below will be used in the future subjects also in the ASIC design industry. If you have time please learn these skill below to improve on using EDA tools.

a. ***To share a folder (Linking) from our PC to Vmware, first click on Edit virtual machine settings.***



Click on Options tab and make sure that Shared Folders is Always Enabled. Edit PATH of folder you want to connect with virtual machine. Click OK to finish setup.

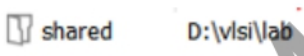


After configuring on the vmware player management of the virtual machine, we need to suspend and resume the machine so that the configuration can be properly applied to the virtual machine.

After the machine turned on, we then open command prompt type "cd" to get to the "~" directory and type **mntsh**.

```
[admin@centos7 ~]$ mntsh
```

Your folder will be connected to "**shared**" folder. Move the unzipped synthesis package into window folder we shared to the virtual machine.



And we can now access the synthesis package and start working on it.

b. ***To use the virtual machine with linux scripting skill is required***. Users need to get acquainted with some command line for creating/removing folders, accessing or modifying our design files and execute software through command or scripts.

cd	Access to target folder (directory navigation). To understand this, we need to know about Linux directory hierarchy: Linux directory always begin at root directory, or "/". Ex: cd /mnt/hgfs To access to folder in home dicrectory, which is default user directory, you need to add "~". When we login into our system, we're automatically placed in our home directory. Ex: cd ~/shared To go to the previous directory, or higher level folder, we add " ../" press or double "Tab" to discover the existing items in the current directory. Ex: cd ../
mkdir	Make new folder Ex: Make a new folder with name "lab3" mkdir lab3

pwd	Print current directory
cp	Copy file or folder A to dicrectory B Ex: cp -rf dirA dirB The -r option is recursive which help copy all nested content inside a folder and -f is force.
rm	Remove files or folder Ex: rm -rf dirA
ls	List all files and folders in current directory
mv	Move a file or a folder to another location Ex: mv ./location-source ./location-destination

For more information on scripting please view the link below:

Bash Scripting Tutorial for Beginners https://www.youtube.com/watch?v=tK9Oc6AEnR4

c. *Vim editor*

Using text editor such as vscode and notepad is not bad for developing codes, but vim is another method of editing the scripting and faster with combinational keys and limit the access to the usage of mouse, this will help programmers more convenient with the linux scripting style and file navigation since vim support a lot file editing and saving with touching the mouse.

To open a file:

Type "***vim*** document.txt".

To edit the file:

Type "***i***" which means insert. To exit insert-mode we press "**esc**".

To exit the file:
Type "***:q***" which mean quit. If you want to save after editing the file type "***:wq***".

Refer the link below for more guidance to use vim editor.

How to Use Vim – Tutorial for Beginners (freecodecamp.org)

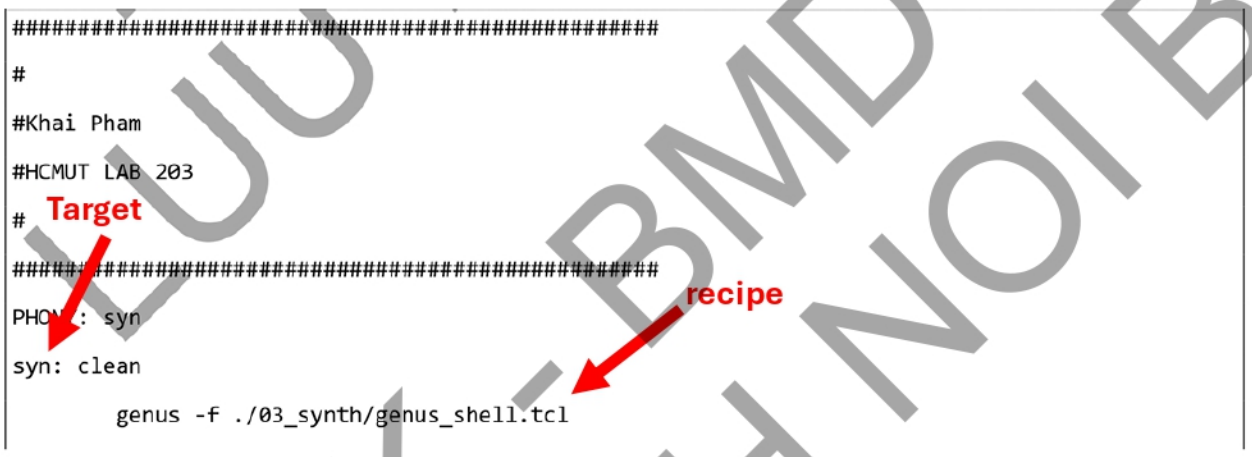
d. *Makefile scripting*

Using makefile we can make recipes that require only a single command line input to the terminal that can execute multiple predetermined commands.

In our example, Makefile contains a line that acts like function names in most of programming language, it's called ***Target***.

Under ***Target***, we have ***Recipe***, which is a combo of multiple command lines that run on terminal. To simplify and reduce the tedious task of inserting each command line into the terminal to achieve such opertions, we write all the commands into the ***Recipe*** region of the Makefile ***Target*** to run all the operation as single command line.

To run the ***Target***, we get into folder which contains makefile, type "**make" follow with the target which is "syn" in the example below** and observe the result.



Inside the Makefile written for lab_3, there are kinds of ***Target***:

syn: run genus to synthesize the target design and modified parameter contain in .tcl file

sim: run simulation tool to observe waveform from our HDL code with timescale 1ns/10ps.

verify: run simulation tool to observe waveform from our files generating after synthesis, only run this target after synthesis.

report: print report after checking.

clean, ***clean-all***, ***clean-rpt***, ***clean-syn***: remove corresponding from recipes.

To use Makefile to synthesize and simulate your design, access to systhesis folder using ***cd*** command and follow lab guide step by step to use each ***Target***.

4. Lab 1: synthesize a 4bit-adder.

*Note: This Lab is only guidance for on-hand exercises with scripting and tools usage, no report required.

Uncompress ***synthesis.zip*** package and type “cd synthesis” to get into the synthesis folder:

```
synthesis/
|-- 00_src
|-- 01_tb
|-- 02_sim
|-- 03_synth
|-- 04_reports
`-- Makefile
```

We will see this directory hierarchy:

+ ***00_src*** folder is for containing RTL code with ***flist.f*** file which is the list of all RTL design-files of the design.

```
synthesis/00_src/
|-- flist.f
|-- full_adder.v
|-- ripple_adder_4bit.v
`-- synth_wrapper.v
```

Inside the ***flist.f*** file, designs file path are listed in order from the most sub-module to the top module hierarchy.

```
00_src/full_adder.v
00_src/ripple_adder_4bit.v
00_src/synth_wrapper.v
```

File ***synth_wrapper.v*** is the top-module of the synthesized design which we will register all top-module input/output interface with instantiated design.

Note:

- if we want to add new designs to the synthesis flow, by putting the HDL files (.v, .sv) inside the folder ***00_src*** and list all the location or path of design files inside ***flist.f*** file (files should be list in-order from smallest design at the top -> the biggest at the bottom),

```
00_src/full_adder.v
00_src/ripple_adder_4bit.v
00_src/synth_wrapper.v
```

and instantiate the design inside the ***synth_wrapper.v*** file which we will need to puts register at both inputs and output ports (*except reset signal and clock*) of the designs.

```
module synth_wrapper
(
 input wire [3:0] X, Y,// Two 4-bit inputs
 input wire rst_n,clk,
 output reg [3:0] S,
 input wire Cin,
 output reg Co
);
  reg [3:0] X_reg;
  reg [3:0] Y_reg;
  reg [3:0] S_reg;
  wire  Co_reg;
  reg  Cin_reg;

always@(posedge clk, negedge rst_n) begin
   if(!rst_n) begin
      X_reg <= 4'h0;
      Y_reg <= 4'h0;
      S <= 4'h0;
      Co <= 1'b0;
      Cin_reg <= 1'h0;
   end
   else begin
      X_reg <= X;
      Y_reg <= Y;
      S <= S_reg;
      Co <= Co_reg;
      Cin_reg <= Cin;
   end
end

ripple_adder u1(X_reg, Y_reg, S_reg, Co_reg, Cin_reg);

endmodule
```

Registered inputs & outputs

Instantiated design

+ ***01_tb*** folder will hold the ***testbench.v*** file which will be used to test the RTL code and also verify the synthesized netlist. Modify ***testbench*** here to run the test as we want it to test the design.

```
synthesis/01_tb/
`-- testbench.v
```

+ ***02_sim*** folder contains library model of gates/cells instantiated inside the synthesized netlist, and ***flist.f*** for listing all models path for netlist verify.

```
synthesis/02_sim/
|-- cells_timing
|-- flist.f
`-- models
```

+ ***03_synth*** folder contains synthesis script ***genus_shell.tcl*** which controls the whole synthesis process, the ***sdf.cmd*** file would be the configuration to apply to the simulation if we need full timing simulation of the netlist.

```
synthesis/03_synth/
|-- genus_shell.tcl
`-- sdf.cmd
```

+ ***04_reports*** will be the folder that's hold all the reports related to the synthesized design, we can check all the necessary information that we need to evaluate and analyze the synthesized design using these reports.

```
[admin@centos7 synthesis]$ tree -L 1 04_reports/
04_reports/
|-- synth_wrapper.area.rpt
|-- synth_wrapper.datapath.rpt
|-- synth_wrapper.gates.rpt
|-- synth_wrapper.hier.rpt
|-- synth_wrapper.power.rpt
|-- synth_wrapper.qor.rpt
`-- synth_wrapper.timing.rpt

0 directories, 7 files
```

+ ***Makefile*** is the master script used to automate all operations of the workflow instead of typing to the terminal each line of commands.

Below is the Makefile script which hold full flow of the synthesis package operation:

```
################################################
#
#Khai Pham
#HCMUT LAB 203
#
################################################
PHONY: syn
syn: clean
	genus -f ./03_synth/genus_shell.tcl
################################################
#GUI = -gui
tb_file := 01_tb/testbench.v

PHONY: sim
sim:
	xrun $(GUI) +xm64bit -sv \
	-vlogext .sv \
	-f 00_src/flist.f \
```

```
       $(tb_file) \
       -timescale 1ns/10ps \
       +access+rcw
################################################
#GUI = -gui
delay_mode := -delay_mode punit
#sdf_file := -sdf_cmd_file 03_synth/sdf.cmd
netlist := 03_synth/*_gate.v
test_file := 01_tb/testbench.v

PHONY: verify
verify:
       xrun $(GUI) +xm64bit -sv \
       $(netlist) \
       $(test_file) \
       -vlogext .sv \
       -f 02_sim/flist.f \
       $(sdf_file) \
       $(delay_mode) \
       -timescale 1ns/10ps \
       +access+rcw

################################################
PHONY: report
report:
       @echo "with Frequency at: "
       @grep "set FREQ_GHz" 03_synth/genus_shell.tcl
       @grep "Path " 04_reports/*.timing.rpt

################################################
PHONY: clean
clean:
       @rm -rf genus.cmd genus.log xrun.history xrun.key xcelium.d *.shm

################################################
PHONY: clean-all
clean-all: clean-rpt clean-syn
       @rm -rf fv genus.cmd genus.log xrun.history xrun.log xrun.key synth_wrapper.sdf.X
xcelium.d *.shm

################################################
PHONY: clean-rpt
clean-rpt:
       @rm -rf 04_reports/*.rpt

################################################
PHONY: clean-syn
clean-syn:
       @rm -rf 03_synth/*.v 03_synth/*.sdf

################################################
```

Note:

Please review the Makefile script to see what the make command will execute.

Steps to run the first lab:

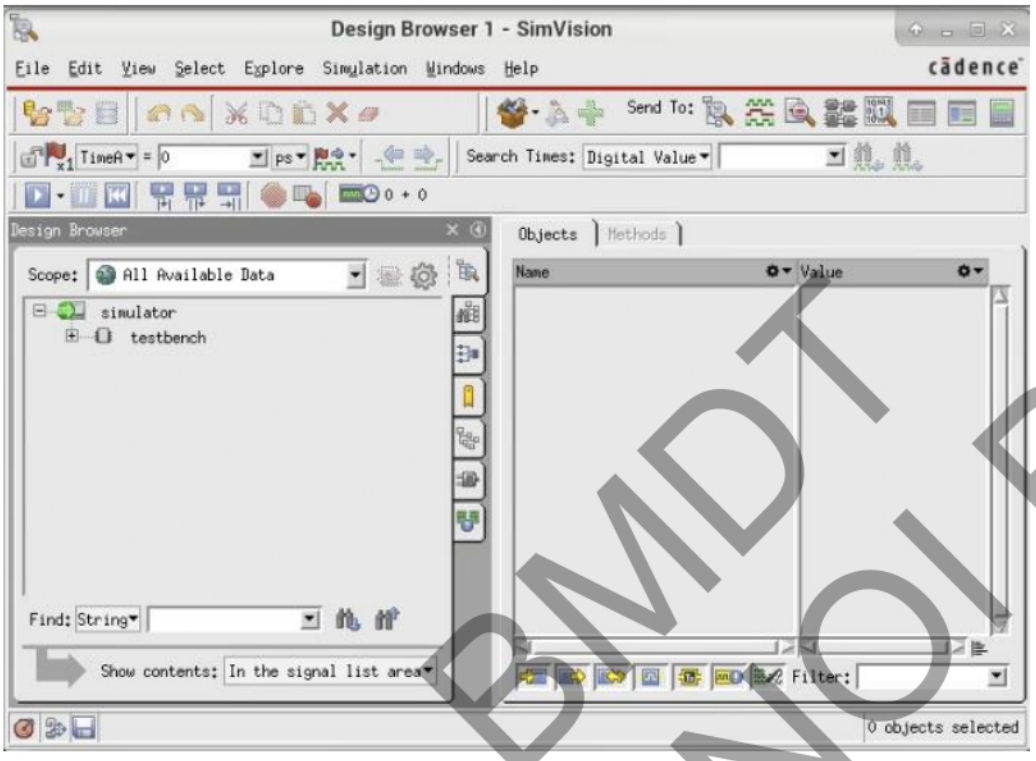
First, we will try to simulate the adder example:

- Type ***make sim GUI="-gui"*** and press ***enter*** to simulate the RTL code of the 4bit-adder design.

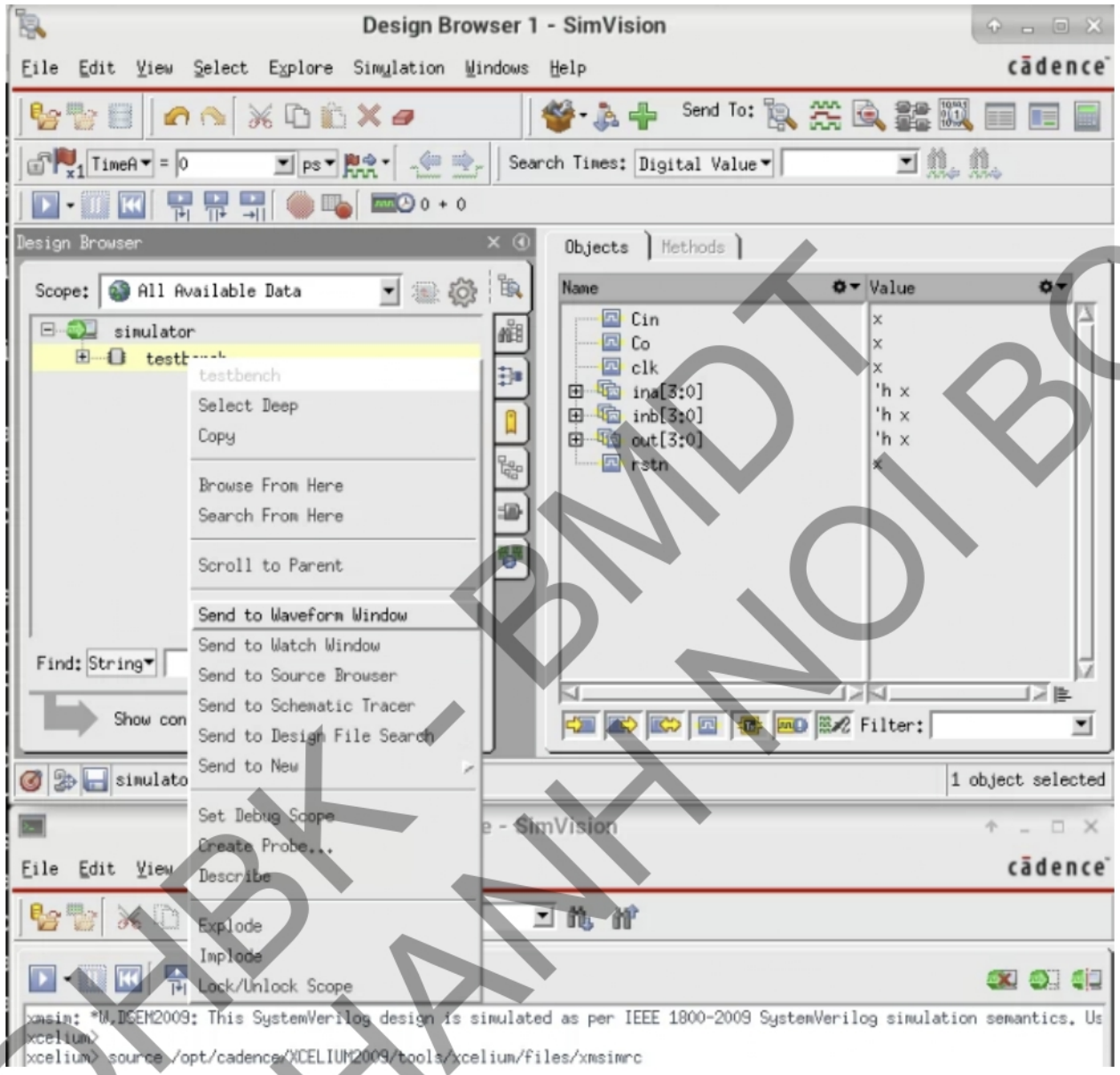
```
[admin@centos7 synthesis]$ make sim GUI="-gui"
```

```
PHONY: sim
sim:
        xrun $(GUI) +xm64bit -sv \
        -vlogext .sv \
        -f 00_src/flist.f \
        $(tb_file) \
        -timescale 1ns/100ps \
        +access+rcw
```

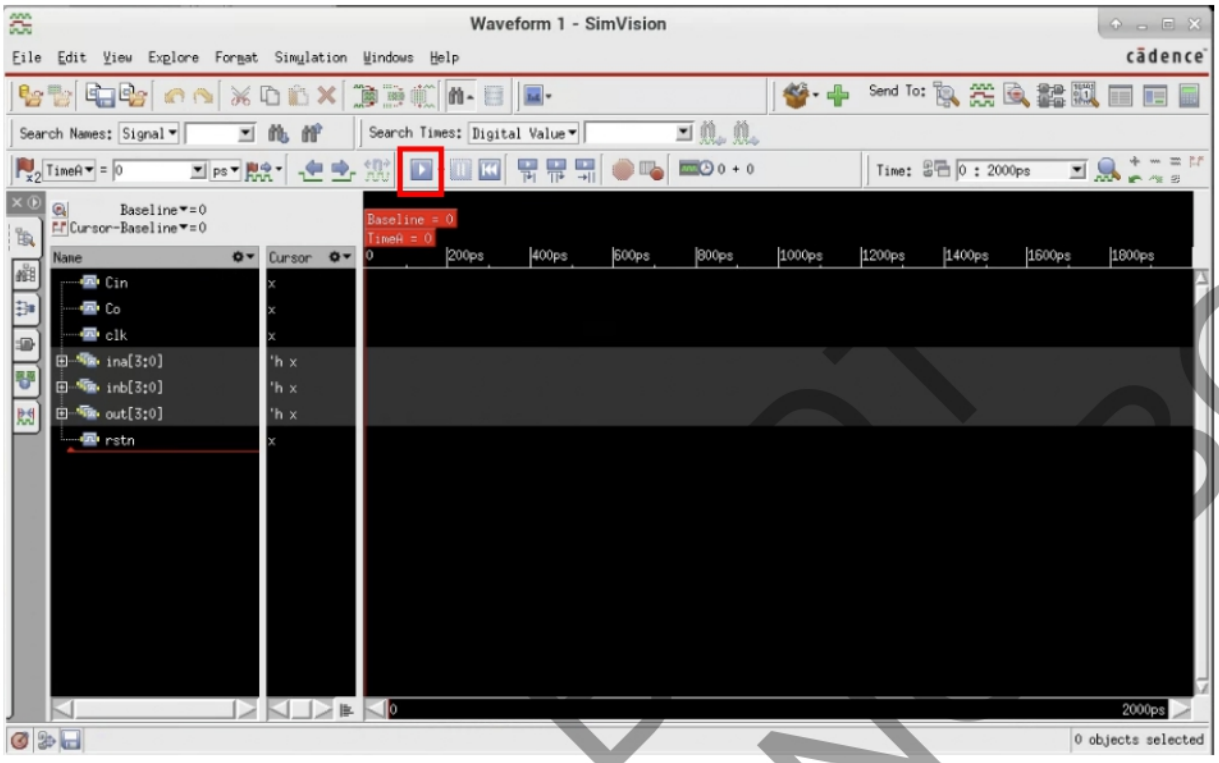
- xrun is the command to run the Verilog simulator tools called xcelium, and other beside "xrun" are options to run and simulate, importance option you need to know is "+access+rcw" is for waveform dumping so that we can observe and debug issue relate to RTL simulation.
- Option ***-f 00_src/flist.f*** is for using the list of RTL files so that the simulator will use paths in the ***flist.f*** file to find the RTL design files.
- We can also type ***make sim*** and simulate without tool GUI but to view wave form we need to type ***make sim GUI="-gui"***.
- Design browser window will pop-up.



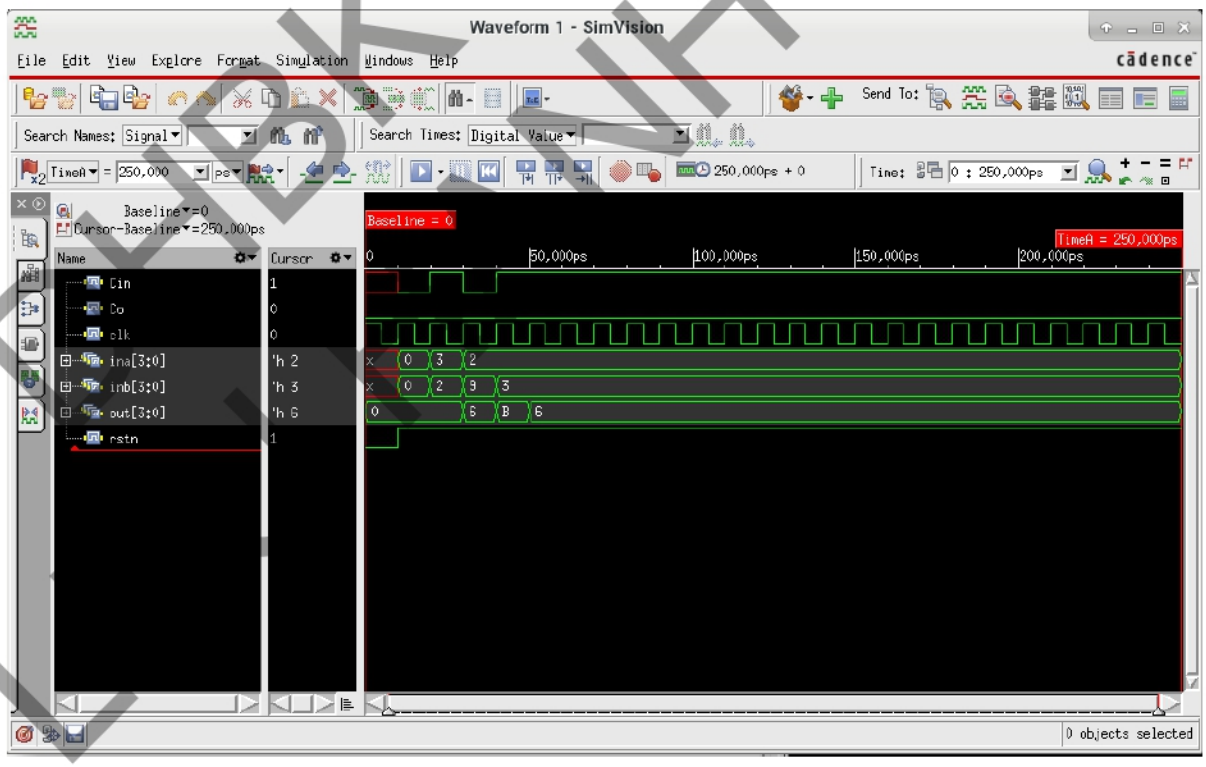
- Select the "***testbench***" with right-mouse and choose "***Send to Waveform Window***".



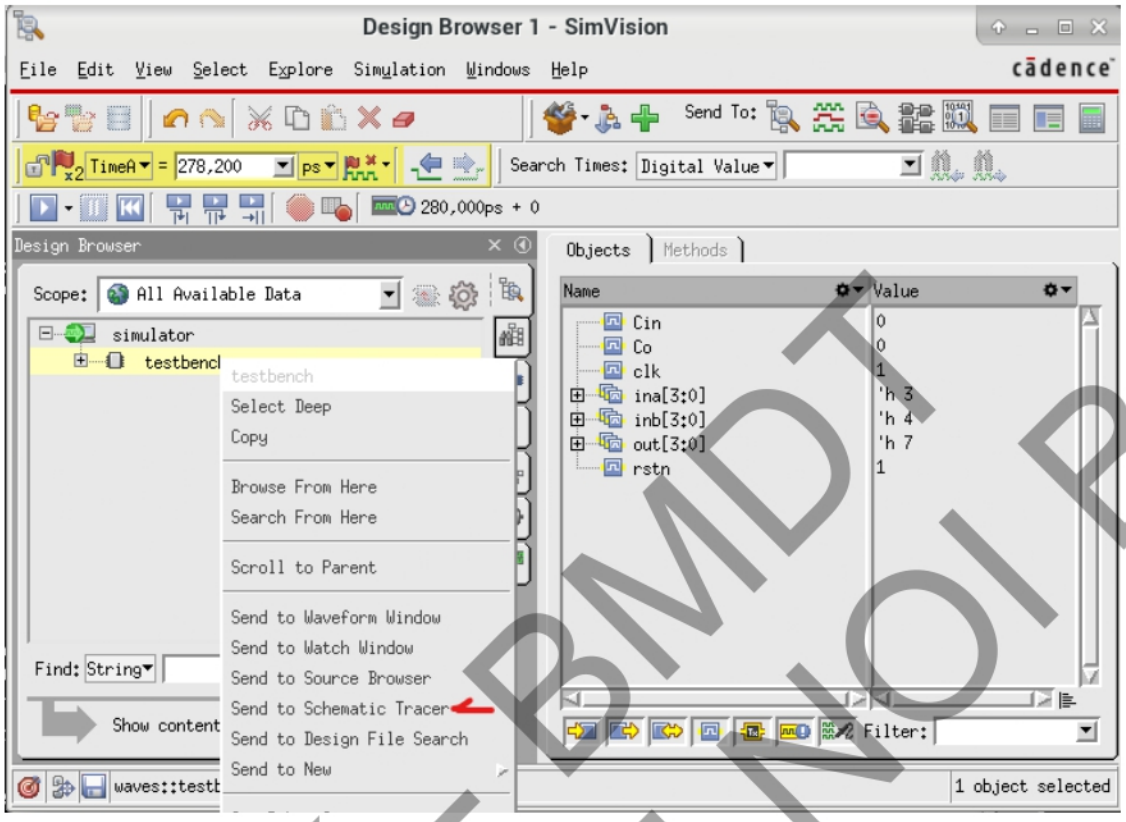
- The waveform viewer window will pop-up and we select the button in the red box below to run.



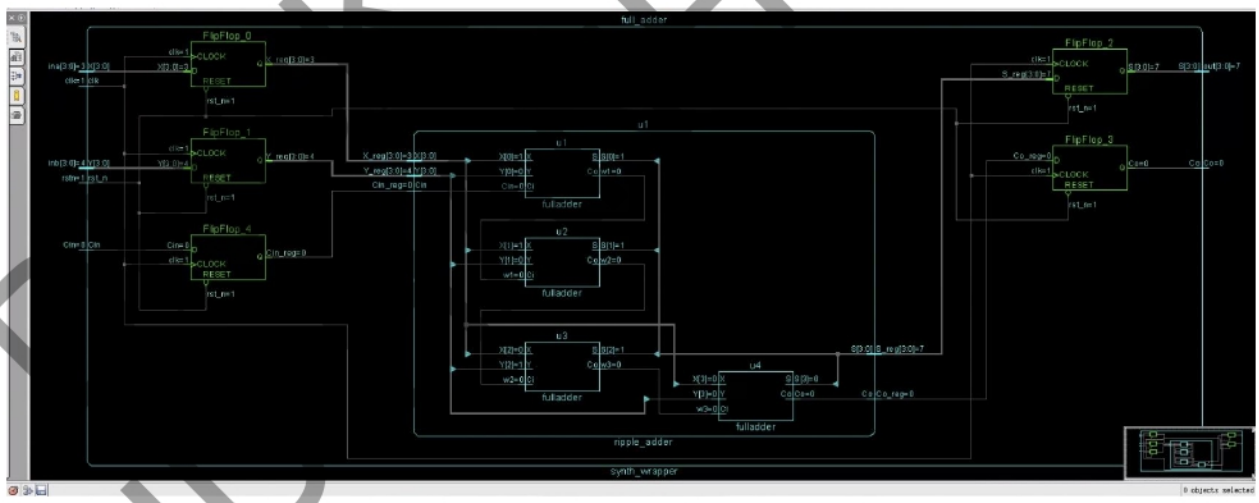
- And the waveform of RTL simulation will appear and press "**=**" key to zoom out fully.



- We can view the RTL schematic by sending the design to "***schematic tracer***".



- The Schematic trace window will appear.



- After developing the RTL design and verifying it, we will go to synthesize the RTL design which is in this case the 4bit-adder with ***genus synthesis*** tool by typing "***make syn***" into terminal.
- Before we type "***make syn***" to the terminal and start synthesizing, we need to specify the frequency for synthesizing the design which could be found in the ***genus_shell.tcl*** script.

```
#################################################
```

```
#Timing constraint variables#
set FREQ_GHz 1.1
set FREQ [ expr ${FREQ_GHz} * 1000000000.0 ]
set PERIOD_tmp [ expr (1.0/${FREQ}) ]
set PERIOD [ expr ${PERIOD_tmp} * 1000000000000.0 ]
puts "Clock Period = ${PERIOD} ps"
set IN_DLY [ expr $PERIOD/2.0 ]
set OUT_DLY [ expr $PERIOD/2.0 ]
set UNCER [expr ${PERIOD}/100000.0]
puts "Clock Uncertainty = ${UNCER} ns"
set TRANS 1.5
puts "max transition = ${TRANS} ns"
################################################
```

We can change the frequency (in GHz) at the highlighted part of the timing constraint variables inside genus_shell.tcl.

```
[admin@centos7 synthesis]$ make syn
```

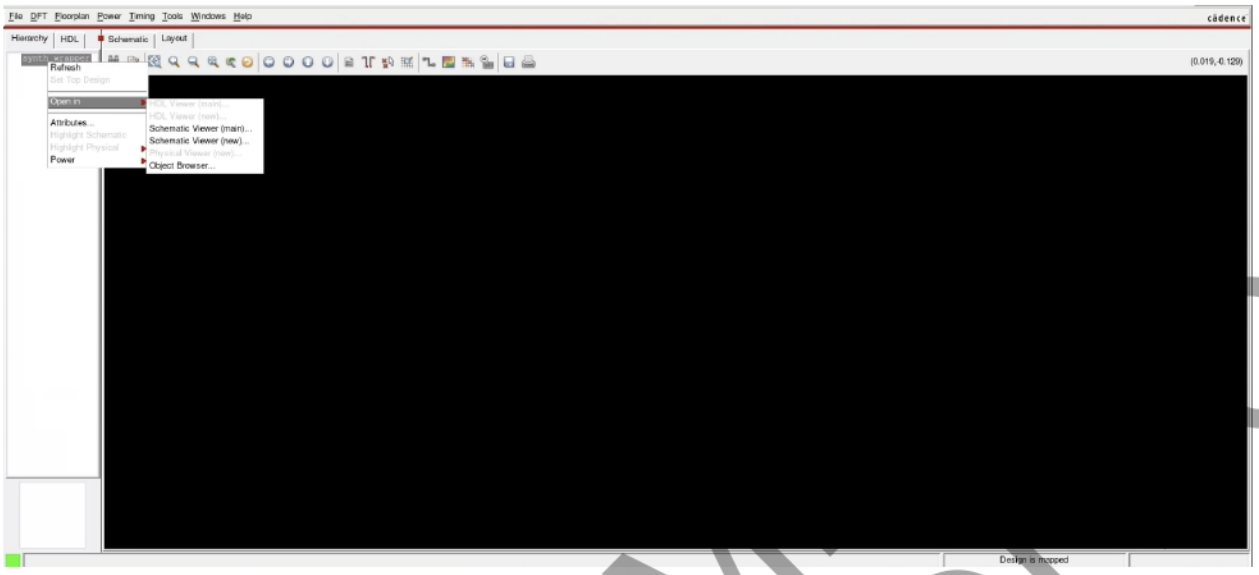
- The ***make syn*** will took the recipe syn which invoke the genus synthesis tool with the option ***-f*** to add the ***tcl*** file ***genus_shell.tcl*** to run synthesis.

```
################################################
PHONY: syn
syn: clean
        genus -f ./03_synth/genus_shell.tcl
################################################
```

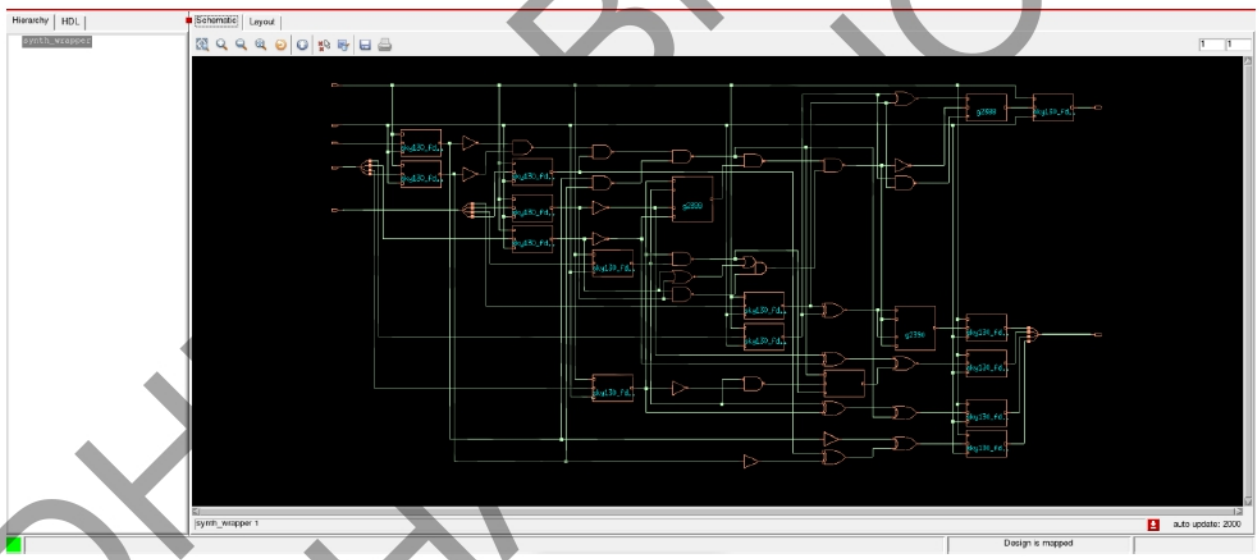
- The synthesis process will start running.

```
Version: GENUS15.20 - 15.20-p004_1, built Sat Nov 14 2015
Options: -files ./03_synth/genus_shell.tcl
Date:    Fri May 03 03:38:28 2024
Host:    centos7 (x86_64 w/Linux 3.10.0-1160.108.1.el7.x86_64) (1*AMD Ryzen 5 2600 Six-Core Processor 512KB)
OS:      CentOS Linux release 7.9.2009 (Core)

Checking out license: Genus_Synthesis

Loading tool scripts...
```

- Before doing synthesis, you might run into un-synthesizable Verilog warnings/errors on, please fix all the Error appear on the first run of the tool log file (***genus.log***) by checking your RTL code.
- GUI of genus synthesis will appear after synthesizing, we choose "***Schematic Viewer (main)***"



- And the schematic of netlist will appear.



- We then exit the GUI application, and type command "***make report***" to terminal to see the timing path resulted from the synthesized design. This will decide if your design passed the timing constraints or not, if a single path violate the timing requirements the RTL will be not good for sending this violated data to the Back-end stages.

```
with Frequency at:
set FREQ_GHz 1.0
Path 1: MET (1 ps) Setup Check with Pin S_reg[3]/CLK->D
Path 2: MET (6 ps) Setup Check with Pin Co_reg/CLK->D
Path 3: MET (19 ps) Setup Check with Pin S_reg[2]/CLK->D
Path 4: MET (55 ps) Setup Check with Pin S_reg[0]/CLK->D
Path 5: MET (99 ps) Setup Check with Pin S_reg[1]/CLK->D
Path 6: MET (165 ps) Late External Delay Assertion at pin Co
Path 7: MET (165 ps) Late External Delay Assertion at pin S[0]
Path 8: MET (165 ps) Late External Delay Assertion at pin S[1]
Path 9: MET (165 ps) Late External Delay Assertion at pin S[2]
Path 10: MET (165 ps) Late External Delay Assertion at pin S[3]
```

- The report can also be found in ***04_reports/synth_wrapper.timing.rpt*** as shown below.

```
[admin@centos7 synthesis]$ vi 04_reports/synth_wrapper.timing.rpt
```

```
Path 1: MET (1 ps) Setup Check with Pin S_reg[3]/CLK->D
     Startpoint: (R) X_reg_reg[0]/CLK
          Clock: (R) clk
       Endpoint: (F) S_reg[3]/D
          Clock: (R) clk

                    Capture       Launch
       Clock Edge:+    1000            0
       Src Latency:+      0            0
       Net Latency:+      0 (I)        0 (I)
           Arrival:=    1000            0

              Setup:-    131
        Uncertainty:-     10
      Required Time:=    859
       Launch Clock:-      0
          Data Path:-    858
              Slack:=      1

#-------------------------------------------------------------------------------------------------
#  Timing Point    Flags    Arc    Edge          Cell              Fanout Load Trans Delay Arrival
#                                                                        (fF)  (ps)  (ps)   (ps)
#-------------------------------------------------------------------------------------------------
  X_reg_reg[0]/CLK -        -      R     (arrival)                    14    -     0     -       0
  X_reg_reg[0]/Q   -       CLK->Q  F     sky130_fd_sc_hd__dfrtp_1      3 10.6    75   378     378
  g2425/Y          -       A->Y    R     sky130_fd_sc_hd__inv_2        1  5.9    41    63     441
  g2415/Y          -       B->Y    F     sky130_fd_sc_hd__nand2_2      1  5.6    34    50     491
  g2404/Y          -       A->Y    R     sky130_fd_sc_hd__nand2_2      1 10.1    73    69     559
  g2403/Y          -       A->Y    F     sky130_fd_sc_hd__nand2_4      3 15.3    50    62     621
  g2397/Y          -       A->Y    R     sky130_fd_sc_hd__nand2_2      1  5.9    52    60     682
  g2395/Y          -       A->Y    F     sky130_fd_sc_hd__nand2_2      3  9.3    49    58     740
  g2390/Y          -       A2_N->Y F     sky130_fd_sc_hd__o2bb2ai_1    1  3.2    59   118     858
  S_reg[3]/D       -       -       F     sky130_fd_sc_hd__dfrtp_1      1    -     -     0     858
#-------------------------------------------------------------------------------------------------
```

- We also can see other aspects of the synthesized design in other reports inside the ***04_reports*** folder.

```
[admin@centos7 synthesis]$ tree -L 1 04_reports/
04_reports/
|-- synth_wrapper.area.rpt
|-- synth_wrapper.datapath.rpt
|-- synth_wrapper.gates.rpt
|-- synth_wrapper.hier.rpt
|-- synth_wrapper.power.rpt
|-- synth_wrapper.qor.rpt
`-- synth_wrapper.timing.rpt

0 directories, 7 files
```

- Below is also the gates usage and area of the design.

```
Timing
--------

Clock Period
-------------
clk   1000.0


  Cost    Critical       Violating
 Group   Path Slack TNS    Paths
-----------------------------------
default         0.8   0          0
-----------------------------------
Total                 0          0

Instance Count
--------------
Leaf Instance Count             46
Sequential Instance Count       14
Combinational Instance Count    32
Hierarchical Instance Count      0

Area
----
Cell Area                                553.030
Physical Cell Area                       0.000
Total Cell Area (Cell+Physical)          553.030
Net Area                                 231.297
Total Area (Cell+Physical+Net)           784.327

Max Fanout                               14 (clk)
Min Fanout                               1 (n_0)
Average Fanout                           1.9
Terms to net ratio                       2.6
Terms to instance ratio                  3.3
Runtime                                  9.927 seconds
Elapsed Runtime                          11 seconds
RC peak memory usage:                    473.21
EDI peak memory usage:                   no_value
Hostname                                 centos7
```

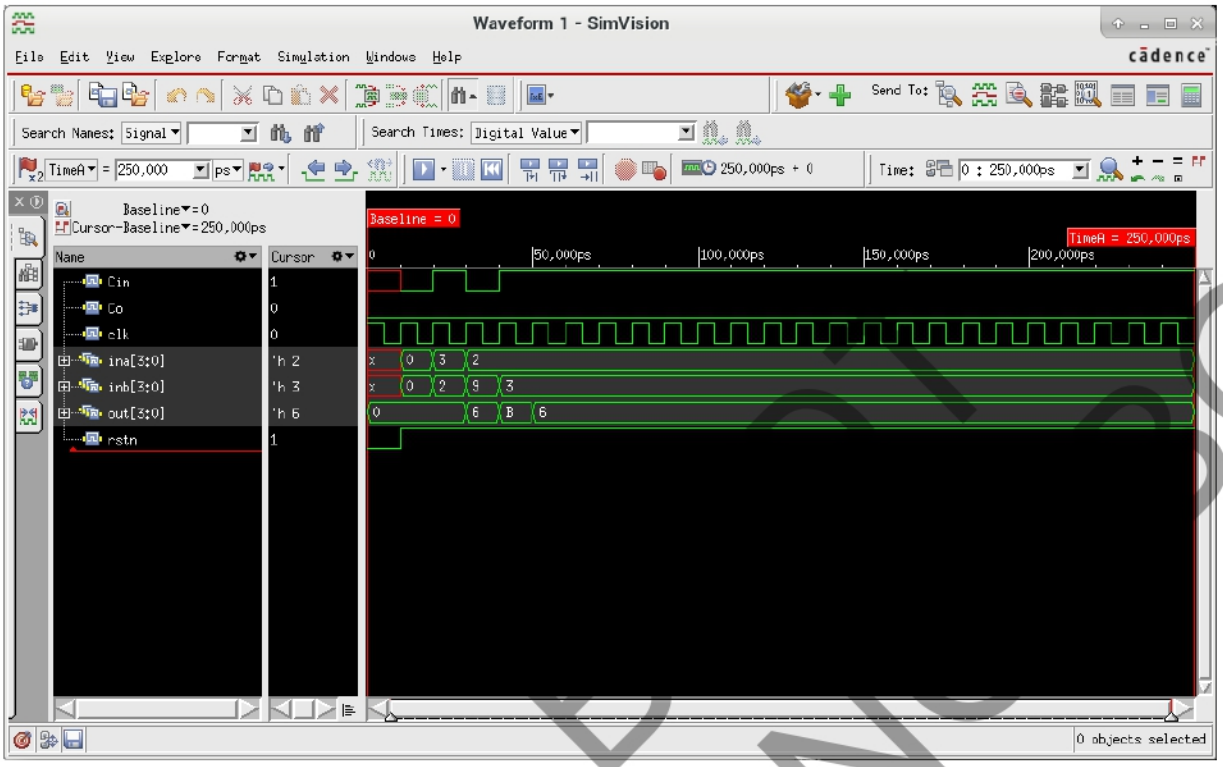
Quality Of Results report (QoR)

Gate	Instances	Area	Library
sky130_fd_sc_hd__a21boi_2	1	11.261	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__a2bb2oi_1	1	8.758	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__buf_1	1	3.754	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__buf_2	1	5.005	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__dfrtp_1	14	350.336	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__inv_1	2	7.507	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__inv_2	4	15.014	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__nand2_1	4	15.014	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__nand2_2	4	25.024	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__nand2_4	1	11.261	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__nand2b_1	1	6.256	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__nor2_1	2	7.507	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__o21a_1	1	7.507	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__o21ai_2	1	8.758	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__o2bb2ai_1	1	8.758	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__xnor2_1	2	17.517	sky130_fd_sc_hd__tt_025C_1v80
sky130_fd_sc_hd__xor2_1	5	43.792	sky130_fd_sc_hd__tt_025C_1v80
total	46	553.030	

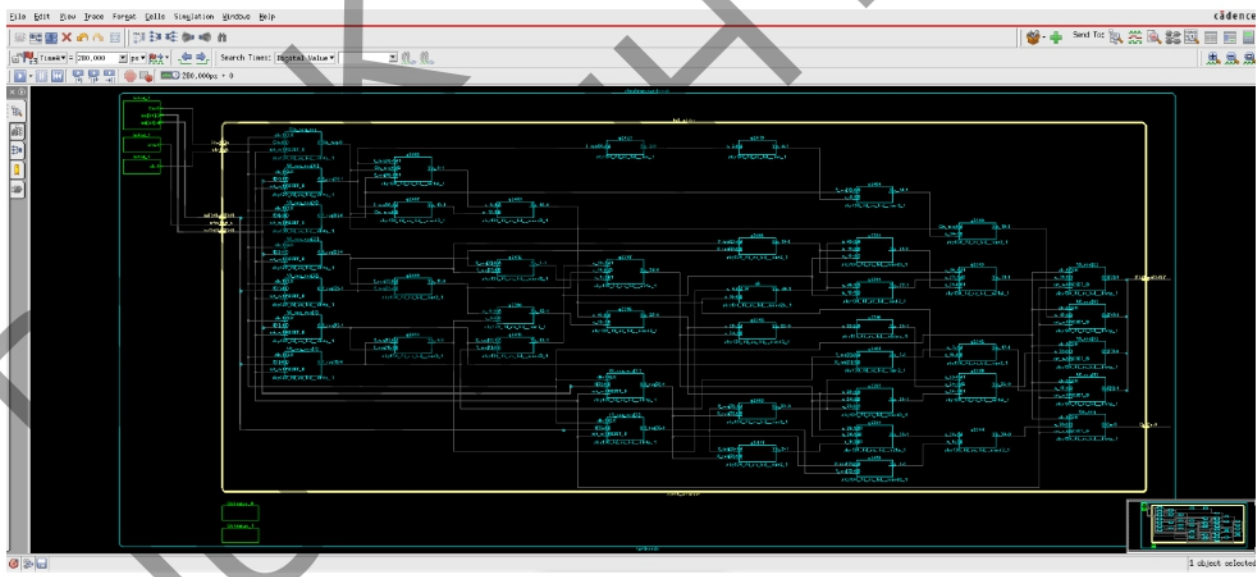
Type	Instances	Area	Area %
sequential	14	350.336	63.3
inverter	6	22.522	4.1
buffer	2	8.758	1.6
logic	24	171.414	31.0
total	46	553.030	100.0

Gate usage of the design

- After synthesis, we will test the netlist with the testbench we developed in the RTL stage, type ***make verify GUI=”-gui”***.
- And access the waveform exactly like what we did above for RTL, the result below will appear.



- We can also view the schematic of the netlist.



- We can see the hierarchy of the netlist is totally flattened.
- If we double-click into any cell-box it will show the model inside the standard cell model.

Note:

After this lab what you need to do for lab2 and lab3 are:

- Change ***00_src*** folder content (add your design file, modify ***flist.f*** file as described above on the beginning of the ***lab1*** tutorial).
- Modify ***testbench*** file in folder ***01_tb*** to test your design.

5. *Lab 2: Design and synthesize "4-bit counter" specified in the previous RTL and verification lab (lab-3_1), use verification plan in previous lab.*

- Synthesize at the maximum frequency that the design does not cause any violation in timing report.
- Do not use (+, -, <<, >>, *, /, …) operator in the design for arithmetic computation, each computational in the design must be structural, which mean arithmetic operator should not be used in the design (this will help the operator meet higher frequency).
- Report resources usage (gates, area report can be found in ***04_reports***).
- Post-synthesis netlist must meet all criteria in the verification plan.

6. *Mini project: Design and synthesize ALU specified in the previous RTL and verification lab(lab-3_1), use verification plan in previous lab.*

- Get the maximum frequency (no violate on timing reports) and report.
- Do not use (+, -, <<, >>, *, /, …) operator in the design for arithmetic computation, each computational in the design must be structural, which mean arithmetic operator should not be used in the design (this will help the compute module meet higher frequency).
- Report resources usage (gates, area report can be found in ***04_reports***).
- Post-synthesis netlist must meet all criteria in the verification plan.

Lab/Mini-project report requirements:

- Hand-over your compressed synthesis folder.
- Specify verification plan mentioned from the RTL and verification labs that will be used for verifying netlist.
- High level Architecture of the design (**Specification design mentioned in the RTL and verification lab**).
    - block diagrams of the design.
    - flow chart of how the design works.
    - input/output description.
      Ex: the description of the 4-bit counter

Signal	Width	Type	Description
clk	1	input	Clock signal
rst_n	1	input	Negative edge reset. If rst_n = 0, output will be set to 0. Else, it will start the normal operation.
sel	1	input	Mode selection signal. If sel = 1, the design will start counting up else, it will start counting down from the current output value.
out	4	output	Result of the counter.

    - Type or algorithm of arithmetic compute unit architecture used in the design (ripple adder, carry look ahead adder).
- Capture the reports of:
    - Highest frequency the design can be synthesized.
    - Timing report of the design like below (show that 10 paths printed from the command ***make report*** have no VIOLATE indication).
    - Ex:

Below is timing MET with frequency at 1.0GHz

```
with Frequency at:
set FREQ_GHz 1.0
Path 1: MET (1 ps) Setup Check with Pin S_reg[3]/CLK->D
Path 2: MET (8 ps) Setup Check with Pin S_reg[1]/CLK->D
Path 3: MET (10 ps) Setup Check with Pin S_reg[2]/CLK->D
Path 4: MET (16 ps) Setup Check with Pin Co_reg/CLK->D
Path 5: MET (61 ps) Setup Check with Pin S_reg[0]/CLK->D
Path 6: MET (165 ps) Late External Delay Assertion at pin Co
Path 7: MET (165 ps) Late External Delay Assertion at pin S[0]
Path 8: MET (165 ps) Late External Delay Assertion at pin S[1]
Path 9: MET (165 ps) Late External Delay Assertion at pin S[2]
Path 10: MET (165 ps) Late External Delay Assertion at pin S[3]
```

Below is timing VIOLATED with frequency increased to 1.1GHz

```
with Frequency at:
set FREQ_GHz 1.1
Path 1: VIOLATED (-64 ps) Setup Check with Pin S_reg[3]/CLK->D
Path 2: VIOLATED (-63 ps) Setup Check with Pin Co_reg/CLK->D
Path 3: VIOLATED (-60 ps) Setup Check with Pin S_reg[2]/CLK->D
Path 4: VIOLATED (-40 ps) Setup Check with Pin S_reg[0]/CLK->D
Path 5: MET (16 ps) Setup Check with Pin S_reg[1]/CLK->D
Path 6: MET (121 ps) Late External Delay Assertion at pin Co
Path 7: MET (121 ps) Late External Delay Assertion at pin S[0]
Path 8: MET (121 ps) Late External Delay Assertion at pin S[1]
Path 9: MET (121 ps) Late External Delay Assertion at pin S[2]
Path 10: MET (121 ps) Late External Delay Assertion at pin S[3]
```

    - Total area of the design and gates resources used in the design.
- Waveform of RTL and netlist simulation.
    - Capture the waveform from both RTL and netlist simulation and gives comments.

Contact Information:

email: ***hanspham.1970s@gmail.com***