IFFT

# Step1

* Make compile

This command utilizes the VCS Verilog compiler or simulator to compile Verilog source files with a testbench file. It activates linting checks for potential issues, enables comprehensive debugging information for the simulation, and sets the simulation timescale to 1 nanosecond with a precision of 10 picoseconds. Additionally, it incorporates all Verilog source files found in the "../src/" directory, and includes the testbench file located at "../tb/tb\_ifft64\_radix2\_top.v".

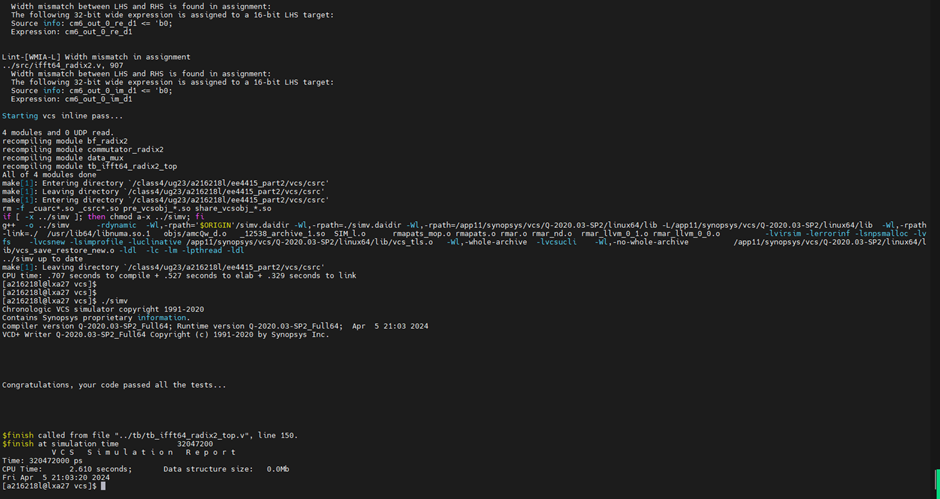
* Make plot

Generate a waveform plot using DVE.

* Make clean

Remove temporary files generated during the previous compilation.

* Pass all the 1000 test cases through VCS simulation.



* Show the waveforms in DVE.

A computer screen shot of a computer

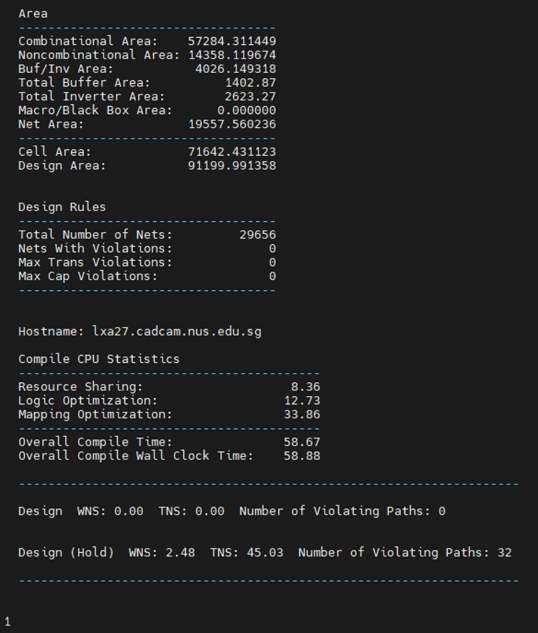
Description automatically generated

# Step2

* Make synthesis

This command initiates the use of DC\_shell with custom settings ("-xg-t") and directs it to execute a series of commands outlined in the "dc-syn.tcl" Tcl script.

* Show the timing and area reports. Check is there any violation? Point out which part of circuits cause the critical path.



There are no timing violations, and the area measures 91199 square micrometers.

* In real design cases, you are going to use the synthesized files for next step (the .ddc, .sdc, sdf, .v files in the /syn/output folder). Compare the synthesized netlist (.v file) with your Verilog code and show your comments.

In actual netlists, RTL (Register Transfer Level) descriptions are substituted with standard cells. For instance, registers described in RTL are represented by DFF\* in the netlist, and arithmetic operations like addition and subtraction are replaced with combinational logic composed of half-adders and other gate circuits, as illustrated in the diagram below.

A screen shot of a computer program

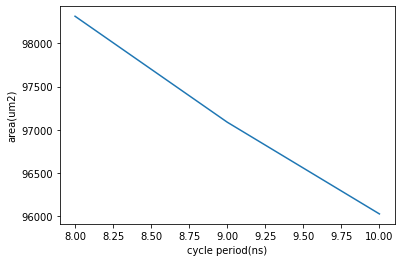
Description automatically generated

A black and white background with text

Description automatically generated

# Part 3

* Make changes to your constraint files and plot the timing/area trade-off curve.

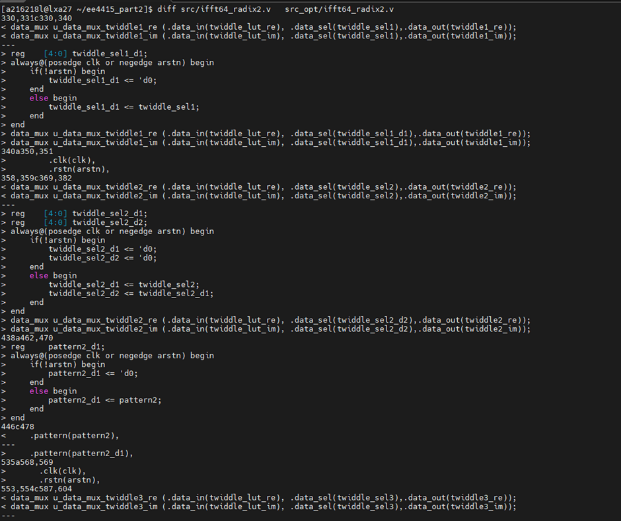


* Will the whole system still work properly after register insertion, retiming, etc… and why?

The system can still function correctly because even though I separate addition and multiplication operations within each bf\_radix2\* module using registers, I ensure the functionality by delaying twiddle\_sel, pattern, and start\_check signals corresponding cycles.

* Show your comments on the comparison between original design and your optimized design (in circuit, area, timing, etc.).

A screenshot of a computer program

Description automatically generated

As mentioned before, by incorporating registers into the combinational logic, timing can be improved to achieve faster operating frequencies, albeit at the expense of area.

# Part 4: Improvement (open ended) (2%)

* Can you further improve the whole design?

I believe it's possible to reduce area in synthesis by utilizing Multibit registers, or by employing LVT (Low Voltage Threshold) technology to enhance performance.