hECE 260A – Lab 3

4-tap 16-bit unsigned averaging FIR filter

Due: 2024/12/14 @ 11:59pm

Please make sure that you close existing terminals and open a fresh terminal and execute "ee260afa21". This will set path to the Design Compiler tool

(Group Project – max of 2 per group)

Project Overview

The objective is first to design a FIR filter, at register-transfer-level abstraction (in System Verilog) and then synthesize from RTL to a gate-level netlist (in 65nm CMOS). This filter is a **4-tap unsigned magnitude FIR filter with registered I/O,** that needs to be optimized for fast performance. Here, 4-tap indicates that the filter averages the input signal from four different time stamps. One way to implement this is to use 4 flops to capture the signal at different time steps.

The following figure shows an illustration of N-tapped delay line with rectangular boxes representing flops. In our project, we discard the first input before the first delay element and start from the output of the first flop. So, we will have 4 flops and the output of flops is used for averaging (coefficient multiplication and addition operations).

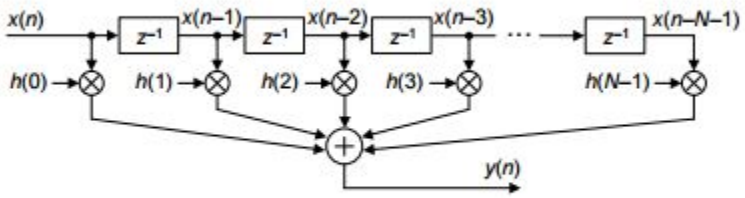


Figure 2: N-tap tapped-delay line FIR filter.

For simplicity, assume that all the coefficients h[k] to be 1. This makes it a much simpler problem, by not worrying about the intermediate multipliers, represented by X.

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k]$$

With the above assumption of unit coefficients, the key component in this circuit is design "adder" combinational logic and synthesize this Verilog to a gate-level netlist.

1. Design your added logic in System Verilog. Use the testbench provided, to validate the functionality in Modelsim, Questa or any other simulator that you are comfortable with. Free FPGA simulation (Mentor Questa) and synthesizer (Mentor Precision) tool access is also available at EDAPlayground.com.

2a. Synthesize from RTL to a lower-abstraction (gate-level) using "Design-Compiler (DC)" tool. This is the start of physical design flow.

2b. Alternatively, synthezie from RTL to an FPGA implementation. I suggest using Mentor Precision on the EDAPlaygorund, but Xilinx Vivado and Intel Quartus (freeware student version, if you like) are also good.

We can implement the four-input adder logic in several ways. One possible way is to use a ripple carry adder. Once you design the adder logic in system verilog, you should synthesize the circuit to a gate-level in 65nm CMOS process or an FPGA of your choosing. The output of synthesis is a gate-level netlist where the components are standard cells such as NAND, NOR, MUX, Full Adders etc.

This project is a great opportunity to design a complex circuit at RTL abstraction and synthesize to gate-level netlist, that will solidify your knowledge of arithmetic architecture and circuit design and your ability to creatively design circuits that have better performance, (that could manifest as increase in power and area) than the reference design. By using the state-of-the art compiler from Synopsys ("Design Compiler", also referred as DC), it also helps in connecting your understanding of a schematic to actual gates that get placed and routed on the silicon. This is a good starting point for ECE 260B (physical design), that starts from a gate-level netlist, and takes you through the physical design flow.

Design Criteria

The adder shall have clk and reset inputs, along with two's comp. data input A[15:0] and output S[17:0]. The output is driven by the Q outputs of positive-edge triggered flip-flops, and the input drives the D inputs of positive-edge triggered flip-flops. We provide a Verilog testbench for those wishing to try a hand at logic simulation/verification, but the main objective will be to compare a few candidate architectures by synthesizing them using Design Compiler or an FPGA synthesizer.

The critical timing paths are the flop-to-flop paths, between the flops in the input stream and flop generating sum. So, if the adder combinational logic involves heavy series circuitry, the tool tries to meet the given clock frequency by using low-vt and high drive-strength cells. This results in increase of power or area, to meet the given (fixed) frequency. If you try to parallelize the adder logic, by reducing the cascading nature of the adder, this increases the amount of combinational logic between these flops. Eventually, this leads to an increase in power and/or area, but a potentially significant reduction in the delay of the timing paths.

A faster design that you propose should try to avoid making significant trade-offs with power consumption or layout area. Proposed designs that have low power consumption and a lesser area at a given clock period of 0.5ns will be awarded more points than designs that have high power consumption and large area.

You are free to try CSAs, as in W&H Fig. 11.42(c), a three-adder "tree" architecture, or any other topology that gets the job done, with improvements from reference architecture in terms of performance, power and area.

Project Report Requirements

The report should include your names and student ID numbers. Please submit this "report" to Canvas by the due date noted above. "Only one" partner should submit a copy – multiple submissions for the same group could lead up to -10 points. Also, mention contribution of each group member (one line per group member). The total project report should not exceed 5 pages.

Reporting Contents

The final project report can consist of figures with text annotation. Suggestions on topics to cover in your report are:

- Briefly describe (max of 2 pages) your chosen architecture of "adder" or architectures that your tried and did not work, and why you chose the final single "proposed" architecture.
    - If something did not work as you expected, explain why
    - Provide suggestions for future iterations of your design
    - Figure that might help explain your design and/or results.
- Results:
    - Table of power, performance and area comparison (w.r.t reference)
    - Conclusions based on the results
    - Pointer on the server to your proposed System Verilog file and Design Compiler power, timing and qor reports.

A sample table of results is shown below. Feel free to expand if necessary.

Metrics	Ripple Carry Adder (Reference)	Proposed Architecture
Worst negative slack in ps (setup analysis)		
Total power consumed		
Leakage power		
Area of combinational logic		
# of combinational cells		
Total area		

Evaluation

You will be evaluated on the quality of your final report and design choice reasoning (50%), and experimental results (40%). Quality of results; power-performance and area improvements from the vanilla ripple carry adder) (10%).

Running the flow:

- You are provided a reference System Verilog file (naive implementation using ripple carry adder)
    - `/home/linux/ieng6/ee260afa23/public/lab3_setup/run_dc_setup_reference/rtl/fir4rca.sv`
- We are using 65nm CMOS process w/ DC.
- Clock frequency of 2 GHz
- Please open the following input files (to Design Compiler) to understand the contents of these files. Do not change the content of these files.
    - `/home/linux/ieng6/ee260afa23/public/lab3_setup/run_dc_setup_reference/run_dc.tcl` is the setup file for executing Design Compiler
    - `/home/linux/ieng6/ee260afa23/public/lab3_setup/run_dc_setup_reference/*sdc` has the constraints for your design (including clock period and I/O constraints)
    - `/home/linux/ieng6/ee260afa23/public/lab3_setup/run_dc_setup_reference/rtl/*sv` is the system verilog file that is an input to Design Compiler tool.

- Running Design Compiler on reference design
    - **mkdir <your course specific home dir>/lab3**
    - **cd <your course specific home dir>/lab3**
    - **cp -r /home/linux/ieng6/ee260afa23/public/lab3_setup/run_dc_setup_reference/ .**
    - **cd run_dc_setup_reference**
    - **dc_shell -f run_dc.tcl // Command to launch DESIGN COMPILER**
        - **report_power > power.rpt**
        - **report_qor > qor.rpt**
        - **report_timing > timing.rpt**

Sample screenshot for timing report

```
U709/ZN (XNR2D4)                         0.09       0.43 f
U256/ZN (ND2D2)                          0.02       0.46 r
U255/ZN (INVD2)                          0.02       0.48 f
U253/ZN (NR2D4)                          0.03       0.51 r
U457/ZN (NR2XD3)                         0.03       0.54 f
U454/ZN (ND2D4)                          0.03       0.56 r
U198/ZN (ND2D2)                          0.03       0.59 f
U251/ZN (ND3D2)                          0.02       0.61 r
U197/ZN (CKND2D2)                        0.02       0.63 f
U194/ZN (NR2XD1)                         0.02       0.65 r
s_reg_13_/D (DFD2)                       0.00       0.65 r
data arrival time                                   0.65

clock clk (rise edge)                    0.50       0.50
clock network delay (ideal)              0.00       0.50
s_reg_13_/CP (DFD2)                      0.00       0.50 r
library setup time                      -0.03       0.47
data required time                                  0.47
-----------------------------------------------------------
data required time                                  0.47
data arrival time                                  -0.65
-----------------------------------------------------------
slack (VIOLATED)                                   -0.18
```

    - Go through these reports and get a feel of your gate-level circuit, and make use of the shell to query for various attribute of your design
    - **Modify the Path in "run_dc.tcl" File**
        - **Change the "ee260afa19" part to "ee260afa23" in rtlPath and target library before launching DESIGN COMPILER**

```
set top_module fir4rca
set rtlPath "/home/linux/ieng6/ee260afa19/public/lab3_setup/run_dc_setup_reference/rtl/"

# Target library
set target_library /home/linux/ieng6/ee260afa19/public/lab3_setup/libraries/db/tcbn65gpluswc.db
set link_library $target_library
set symbol_library {}
set wire_load_mode enclosed
set timing_use_enhanced_capacitance_modeling true

set search_path [concat $rtlPath $search_path]
set link_library [concat * $link_library ]
```

- Running Design Compiler on proposed design
    - **cd <your course specific home dir>/lab3**
    - **mkdir dc_proposed_design**
    - **cp -r <>/run_dc_setup_reference/* .**

      // Copying the contents including rtl/ directory
    - **cd dc_proposed_design/rtl**
    - **change the system Verilog file (specified as to what part of the file needs to be modified)**
    - **Update run_dc.tcl in your new run dir, with the correct rtl path to your new location**
        - **set rtlPath "<PATH TO YOUR RTL FILES>"**
    - **Change the "ee260afa19" part to "ee260afa23" in target library before launching DESIGN COMPILER**
    - **Do not change any other file (content or name).**
    - **cd ..**
    - **dc_shell -f run_dc.tcl // this needs to be run in the dir where run_dc.tcl is present**
        - **report_power > power_new.rpt**
        - **report_qor > qor_new.rpt**
        - **report_timing > timing_new.rpt**
    - Open these reports and observe the changes from your reference implementation. For any command, you execute "-help" or "man" on dc_shell, to know more about the command options.
    - Eg:
        - **dc_shell>report_timing -help**

- **dc_shell>man report_power**

- Useful link for Linux operation

https://www.javatpoint.com/linux-edit-file

Appendix:

The following timing path indicates a setup violation (negative sign) of 100ps. This is generated by report_timing command executed on dc_shell. By default, only the worst timing path is reported, and unit of time is nanoseconds. Just visualize the number of flop-4flop timing paths exist in your design, by exploring "-slack_greater_than" and "max_paths" options of "report_timing" command. report_qor and report_power commands are self-explanatory.

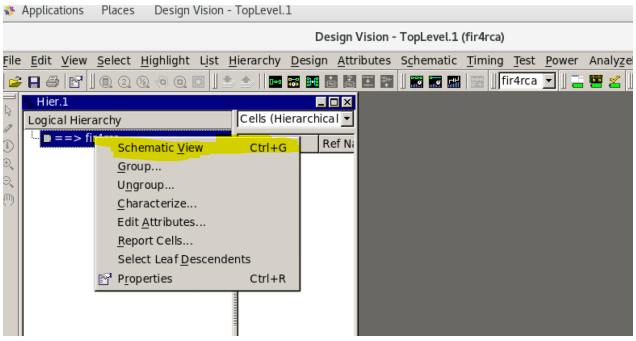
```
U547/ZN (ND2D2)                          0.02       0.35 r
U409/ZN (ND2D2)                          0.03       0.39 f
U563/ZN (CKND2D1)                        0.02       0.41 r
U501/ZN (ND2D0)                          0.04       0.45 f
U921/ZN (XNR2D1)                         0.08       0.53 f
U922/ZN (NR2D1)                          0.03       0.56 r
s_reg_3_/D (DFQD1)                       0.00       0.56 r
data arrival time                                   0.56

clock clk (rise edge)                    0.50       0.50
clock network delay (ideal)              0.00       0.50
s_reg_3_/CP (DFQD1)                      0.00       0.50 r
library setup time                      -0.03       0.47
data required time                                  0.47
-----------------------------------------------------------
data required time                                  0.47
data arrival time                                  -0.56
-----------------------------------------------------------
slack (VIOLATED)                                   -0.10
```

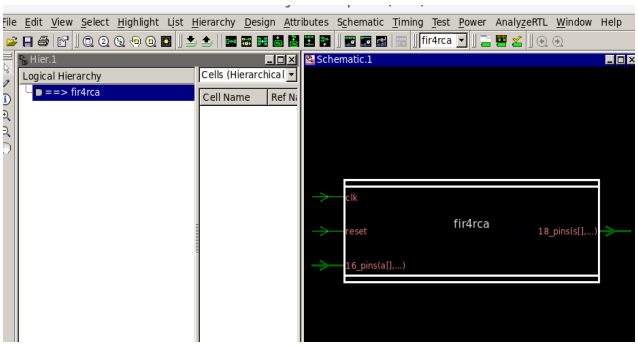
Visualizing generated gate-level netlist:

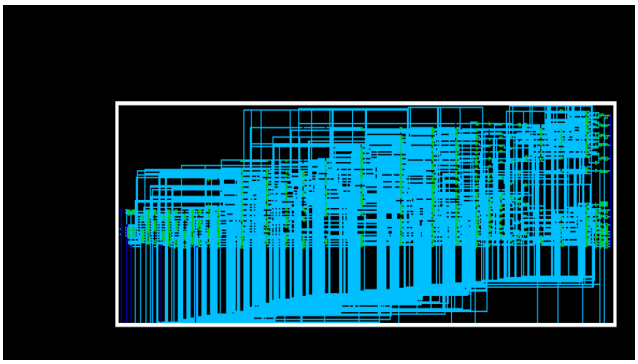
1. Output gate-level netlist is generated in verilog format (<design name>. out.v)

2. To view schematic, execute "start_gui" on dc_shell

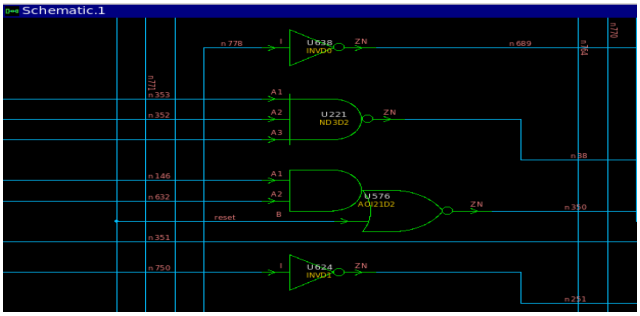


Click on the rectangular module

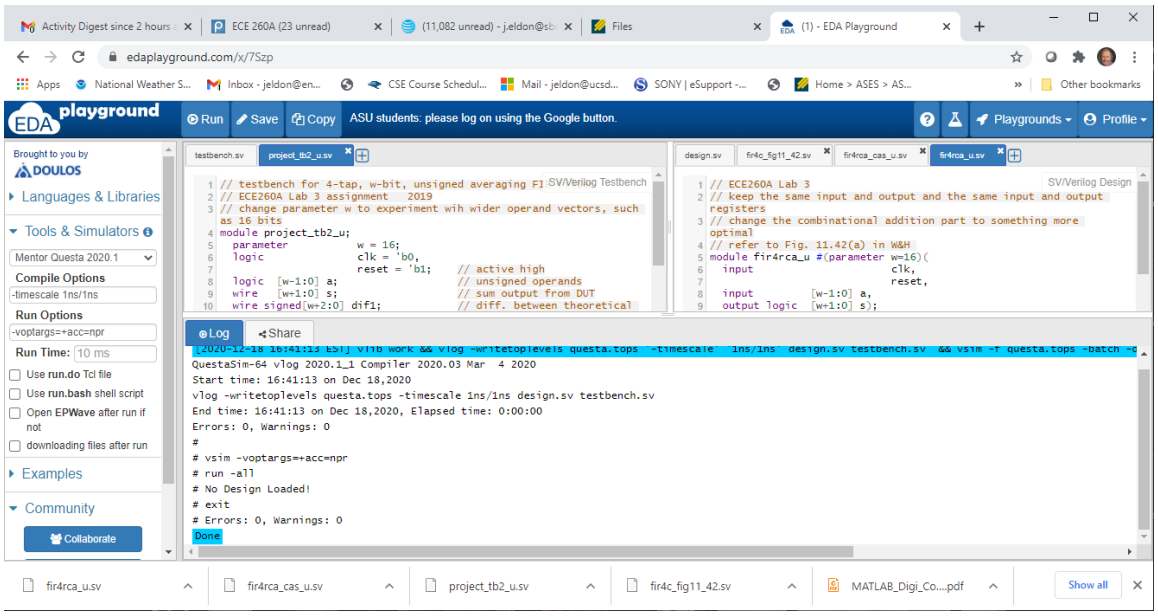




Zoom-in and out using mouse and move scroll using arrow keys



- Optional: You can verify *functionality* of system Verilog, using the test bench that we provided using modelsim. Remember to set the 'fclk' term in your testbench. When doing this, note that to HSPICE, '100M' means 100 mHz, not 100 MHz. To use mega, make sure to type '100MEGA'.
    - `/home/linux/ieng6/ee260afa23/public/lab3_setup/test_bench/fir4.tb`


Activity Digest since 2 hours
ECE 260A (23 unread)
(11,082 unread) - jeldon@sb
Files
(1) - EDA Playground
edaplayground.com/x/7Szp
Apps
National Weather S...
Inbox - jeldon@en...
CSE Course Schedul...
Mail - jeldon@ucsd...
SONY | eSupport -...
Home > ASES > AS...
Other bookmarks
EDA playground
Run
Save
Copy
ASU students: please log on using the Google button.
Playgrounds
Profile
Brought to you by
DOULOS
Languages & Libraries
Tools & Simulators
Mentor Questa 2020.1
Compile Options
-timescale 1ns/1ns
Run Options
-voptargs=+acc=npr
Run Time:
10 ms
Use run.do Tcl file
Use run.bash shell script
Open EPWave after run if not
downloading files after run
Examples
Community
Collaborate
testbench.sv
project_tb2_u.sv
SV/Verilog Testbench
// testbench for 4-tap, w-bit, unsigned averaging FI
// ECE260A Lab 3 assignment 2019
// change parameter w to experiment wih wider operand vectors, such as 16 bits
module project_tb2_u;
parameter w = 16;
logic clk = 'b0,
reset = 'b1; // active high
logic [w-1:0] a; // unsigned operands
wire [w+1:0] s; // sum output from DUT
wire signed[w+2:0] dif1; // diff. between theoretical
design.sv
fir4c_fig11_42.sv
fir4rca_cas_u.sv
fir4rca_u.sv
SV/Verilog Design
// ECE260A Lab 3
// keep the same input and output and the same input and output registers
// change the combinational addition part to something more optimal
// refer to Fig. 11.42(a) in W&H
module fir4rca_u #(parameter w=16)(
input clk,
reset,
input [w-1:0] a,
output logic [w+1:0] s);
Log
Share
[2020-12-18 16:41:13 EST] vlib work && vlog -writetoplevels questa.tops -timescale 1ns/1ns design.sv testbench.sv && vsim -f questa.tops -batch -d
QuestaSim-64 vlog 2020.1_1 Compiler 2020.03 Mar 4 2020
Start time: 16:41:13 on Dec 18,2020
vlog -writetoplevels questa.tops -timescale 1ns/1ns design.sv testbench.sv
End time: 16:41:13 on Dec 18,2020, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
#
vsim -voptargs=+acc=npr
run -all
No Design Loaded!
exit
Errors: 0, Warnings: 0
Done
fir4rca_u.sv
fir4rca_cas_u.sv
project_tb2_u.sv
fir4c_fig11_42.sv
MATLAB_Digi_Co....pdf
Show all