Power and Area

Hardware for Artificial Intelligence Fundamentals - Lab

Hardware for Artificial Intelligence Group

TECHNISCHE UNIVERSITÄT DARMSTADT

Winter 2024
Sheet 6

The most important metrics for an ASIC are Power, Performance, and Area (PPA), which we will evaluate in this exercise. First we examine the Power and Area of a multiplier with two inputs. After that we fix the weight into the multiplier design and observe the optimizations done during synthesis.

Task 6.1: Ordinary Multiplier

For this part we will reuse the multiplier design from the previous exercise. Two signed 8 bit inputs and one 16 bit output.

6.1a) Synthesis

Unfortunately we need to synthesize the circuit again. Besides the netlist we also want to create an area report, which tells us how much area the synthesized circuit is actually requiring. For our power analysis we will need to know the delays of the gates and wires, which we save as a Standard Delay Format (SDF) file.

Listing 1: synthesis.tcl

```
set target_library NangateOpenCellLibrary_typical.db ; # tell synopsis which library to use
set link_library NangateOpenCellLibrary_typical.db

read_file add.v ; # read specified input file
link ; # link design with library
compile_ultra ; # compile design with extra effort
write -f verilog -hierarchy -output netlist_add.v ; # save result as verilog netlist
write_sdf sdf_full.sdf  ; # create SDF file
report_area > area-report.txt  ; # create area report
quit
```

Inspect the area report. We are interested in total cell area, which is measured in μm^2.

6.1b) Simulation

In order to determine the power usage we need to simulate the design to obtain switching activity of the gates. To do this we use modelsim and a testbench, just like in the last exercise.

Listing 2: simulation.tcl

```
vlib work
vlog testbench_mul.v
vlog netlist_mul.v
vlog NangateOpenCellLibrary.v
vsim -sdftype /testbench_mul/dut=sdf_full.sdf work.testbench  ; # annotate multiplier with sdf file
power add /testbench_mul/dut/*
run 10 us
```

Power and Area

```
power report -all -bsaif power_full.saif  ; # create power report
quit -f
```

For simulation we specify our SDF file to annotate delays and export a Switching Activity Interchange Format (SAIF) file which contains the switching information from simulation.

6.1c) Power Analysis

Now that we have obtained the switching information from simulation, we are able to determine the actual power usage. For analysis we use design compiler, as shown in the following script.

Listing 3: analysis.tcl

```
set target_library NangateOpenCellLibrary_typical.db  ; # specify library
set link_library NangateOpenCellLibrary_typical.db

read_verilog netlist_full.v  ; # read netlist
read_saif -input power_full.saif -instance_name testbench/mul  ; # read SAIF
report_power -analysis_effort high > report.txt  ; # create power report
quit
```

Open the report. The last row of the contained table tells us the relevant information.

Task 6.2: Fixed Multiplier

In this part of the exercise we will evaluate what happens when the weight of a multiplication is embedded and therefore synthesized into the multiplier design.

Our goal is to create an area/power report for every weight in the range $[-127, 127]$. To acheive this we create a parametrized multiplier design, then set the parameter. This design then gets synthesized, simulated and analyzed. We repeat this for every integer in the desired range.

6.2a) Multiplier design

The new multiplier will have only one 8 bit input but the 16 bit output remains unchanged. Extend the verilog module listed below.

Listing 4: mul_fixed.v

```
module MUL (
    input wire signed [7:0] a,
    output wire signed [16:0] y
);
    parameter WEIGHT = 0;  // parameters require a default value in verilog
    wire signed [7:0] b = WEIGHT;

    // your code here
endmodule
```

To instantiate a module with a parameter you would do it as in the following code snippet.

```
MUL #(.WEIGHT(7)) dut (/* connections here */);
```

6.2b) Synthesis, Simulation, and Analysis

In previous tasks we have already used scripts to automate design compiler and modelsim. Those scripts were written in the Tool Command Language (TCL), which is also capable of loops, variables and running external commands. We will make use of those features to automate the steps for our parameter.

First let us create the script for modelsim. Use the testbench provided on moodle, testvector is the same as in the previous task.

Listing 5: sim_fixed.tcl

```
vlib work
vlog testbench_fixed.v
vlog $1
vlog NangateOpenCellLibrary.v
vsim -sdftype /mul=$2 work.testbench
power add /testbench/mul/*
run 10 us
power report -all -bsaif $3
quit
```

You may notice the `$1, $2, $3` in the code. Those are arguments which are passed on the commandline to the script when invoking modelsim.

Next we need to create a script for design compiler. It will read the library and multiplier design. And then in a loop it will synthesize the design with a given parameter, execute our modelsim script, and then read back the SAIF file for analysis.

Create directories “logs” and “gen” before running the script.

Listing 6: synth_fixed.tcl

```
set target_library NangateOpenCellLibrary_typical.db
set link_library NangateOpenCellLibrary_typical.db

analyze -f verilog mul_fixed.v  ; # read file but do nothing else with it
for {set factor -127} {$factor <= 127} {incr factor} {  ; # loop over factor range
    set fSdf "gen/sdf_$factor.sdf"  ; # generate variables for filenames
    set fNetlist "gen/netlist_$factor.v"
    set fSaif "gen/power_$factor.saif"

    # create netlist and sdf
    elaborate -parameters WEIGHT=$factor MUL  ; # create design with parameter weight set
    rename_design $current_design "MUL_FIXED"  ; # rename design to MUL_FIXED (otherwise it's new name
    contains the parameter value. e.g., MUL_n127)

    compile_ultra
    report_area > "logs/area_$factor.txt"
    write -f verilog -hierarchy -output $fNetlist  ; # create netlist
    write_sdf $fSdf  ; # create sdf

    if { [ catch {
        # execute modelsim with arguments (filenames)
        exec vsim -- -c -do "do sim_fixed.tcl $fNetlist $fSdf $fSaif"

        # analyze result
        read_saif -input $fSaif -instance_name testbench/dut
    } err ] } {
        puts "No cells for weight $factor"
    }
    report_power -analysis_effort high > "logs/power_$factor.txt"

    # remove design
    remove_design "MUL_FIXED"
}
quit
```

You may wonder why there is a `catch` statement around the modelsim execution and reading the SAIF back in. For certain parameters (e.g., 0, 2) the design does not include any cells from the library because everything gets optimized away. In those cases modelsim tries to annotate cells but none exist, additionally there will be no switching to simulate.

6.2c) Plotting

Use Matplotlib[1] to create power and area plots. The X-Axis should contain the used factor and the Y-Axis the required total cell area/total power. You may want to use the following python functions to automatically parse the reports.

```
import glob
import re

def matchArea(data, keys):
    return [re.search(f"{key}:\\s+(.+)", data).group(1) for key in keys]

for file in glob.glob('logs/area_*.txt'):
    with open(file) as f:
        values = matchArea(f.read(), [
            'Combinational area', 'Buf/Inv area',
            'Noncombinational area', 'Total cell area'
        ])
        print(values)

def matchPower(data):
    vp = "([0-9\\.+-e]+)\\s[mnu]W"
    r = re.search(f"Total\\s+{vp}\\s+{vp}\\s+{vp}\\s+{vp}", data)
    return r.groups()

for file in glob.glob('logs/power_*.txt'):
    with open(file) as f:
        values = matchPower(f.read())
```

[1] https://matplotlib.org/stable/tutorials/pyplot.html