IL2225-Lab 2

Logic Synthesis-II

Power Estimation and Optimization

Name:

_____

Personal Number:

_____

Assistant:

_____

Date:

_____

Logic Synthesis-II

1. Objective

The purpose of this lab is to introduce following concepts:

1. Power Estimation using Value Change Dump File (VCD File)
2. Bottom Up Hierarchal Synthesis
3. Following Power Saving Methodologies during Synthesis:
    a. Clock Gating
    b. Operand Isolation

2. Preparation Tasks

Synthesize the given FIR code in $ IL2225_LAB/LAB2. You should also generate the following files during synthesis. You should perform the elaboration step with the parameter values (width and number of taps) which are given in the RTL code.

Report Following:

Area: _____ Slack: _____ Power (Total):_____
OperatingCondition: _____ WireLoadModel: _____
WireLoadModel Mode: _____

1. Constraints.sdc file, it will be used in the next lab. You can generate it by using the following command at the end of the lab

    write_sdc $ home/IL2225_LAB/LAB2/ MAPPED/constraints.sdc

2. Delays.sdf file: This file will be used to back annotate the delays of the circuit in our gate-level simulation. You can generate it by using the following command

    write_sdf $ home/IL2225_LAB/LAB2 /MAPPED/delays.sdf

3. Power Estimation

In lab1, we use report_power to get the power estimation for our circuit. This estimation is based on statistical switching activities of all the wires in our circuit. By default this switching activity is 0.2. In reality the power consumption of an electronic circuit highly depend on the input data. In this section we will do gate level simulation of the FIR filter. ***Use the netlist which is synthesized in the preparation task***.

Compile the sdf file in Model SIM while doing the gate level simulation. You can also use GUI to add the sdf file into your simulation in modelsim. For that, click simulate->start simulation-> select sdf tab and select the location of the sdf file as well as its scope which will be /tb_fir/u1. This will back annotate all the delays in the circuit during our gate level simulation. During this gate-level simulation, we will record the activities inside the circuit in a Value Change Dump file. This VCD file will then be used to accurately estimate the power in our FIR Filter.

1. Create a testbench for FIR filter with sufficient amount of samples. A good testbench can have a random number generator to generate random samples at every sample clock. No. of samples in the testbench should be selected or specified by a constant. You can change number of samples by simply changing this constant. Use the following process to edit your test bench for random sample generation.

Fig1: *Code for Random Number Genration*

```
random_generator:process
    VARIABLE seed1:positive:=1;
    VARIABLE seed2:positive:=15;
    VARIABLE a_real: real;
    VARIABLE count :INTEGER:=0;
begin

 for j in 1 to number_of_samples loop
  wait until sample_clk'event and sample_clk = '1';
  unIFORm(seed1,seed2,a_real);
  a_real := a_real * 16.0;
  if(count < number_of_samples) then
  sample_wire <= conv_std_logic_vector(INTEGER(a_real), 4) ;
  count := count+1;
  end if;
 end loop;
end process random_generator;
```

2. Check the slack of the synthesized netlist by looking into the timing report. This will tell you the frequency at which the FIR filter will work. This will be the clock frequency which will be used in the test bench.
3. Now use the following commands to generate the VCD file
    c. Run the simulation just before the first sample is generated by the test bench by using the command
    d. At this point, create a VCD file by using the command
    e. Add the signals which are to be recorded in the VCD file. '-r' switch add everything in the design
    f. Run the simulation.
    g. Quit the simulation.

Note: Repeat steps for each of the cases given in Table-III.

a. run xxx ns
b. vcd file PATH/filename.vcd
c. vcd add -r tb_fir/*
d. run xxx ns
e. quit –f

Table I: Script to create a VCD file in ModelSim

4. Synopsys do not support VCD file format and instead supports Switching Activity Information Format. To use this technique in Synopsys, either we need to generate saif file or we can convert a VCD file into a saif format file using the following command. Use the following command on unix/linux prompt to generate saif file.

vcd2saif -i myvcdfile.vcd -o mysaiffile

5. Now you have to use this saif file to do the power estimation in Synopsys.

```
read_file -format verilog {$ home/ IL2225_LAB/LAB2/MAPPED/LAB3_FIRToplevel.v}
elaborate FIR_Toplevel -lib WORK -update -param "width = 4" -param "filter_taps=5"
set_wire_load_mode segmented
set_wire_load_model  - name TSMC8K_Lowk_Conservative
read_saif -input $ home/ IL2225_LAB/LAB2/mysaiffile -instance_name tb_fir
report_power
```

Table II: Commands to estimate power in Synopsys using Saif File

Fill the following table:

No. of Sample	Simulation Cycles/Time	Power
5		
50		

Table-IIIA: Power Vs No. of Samples

6. Change the width of FIR filter to 16-bits by changing the generics in the VHDL Code. Don't forget to change the width of the coefficients in the ROM_process. Keep the values in ROM, just increase its width by adding zeros to the left. Synthesize the design in same condition as above. (Note: Keep a backup of the source code. The original source code with 4-bits will be used further in the lab)

Fill the following table:

No. of Sample	Simulation Cycles/Time	Power
5		
50000		

Table-IIIB: Power Vs No. of Samples

Question: Explain the difference in the average power for different no. of sample in Table-IIIA and IIIB : _____
_____
_____

7. Use the saif file, used for table above, for 500 samples to fill the Table-IIIC. Explain the difference in power results:_____

wire_load_mode	Wire_load_model	Samples	Power
Top	TSMC8K_Lowk_Conservative	500	
enclosed	TSMC8K_Lowk_Conservative	500	
segmented	TSMC8K_Lowk_Conservative	500	

Table-IIIC: Power Vs No. of Samples

4. Power Saving During Synthesis

(Note: Use the original 4-bit code for this exercise)

There are two basic ways to implement power saving techniques during synthesis. These are:

1. Clock Gating
2. Operand Isolation

The theory behind these techniques will be discussed during the lectures. In this lab we will use the Synopsys synthesis tool to implement these in our design and see their effect. Before implementing these on our design please answer the following questions;

Question: What do you understand by clock gating? _____
_____
_____.

Question: What do you understand by Operand Isolation? _____
_____
_____.

(Hint: Please refer to the lecture notes if you do not know about Clock gating and Operand Isolation.)

4.1. Clock Gating:

Enabling clock gating in Synopsys design vision requires slight modification to the synthesis script that we used in the LAB1. These are:

1. Using set_clock_gating_style to set the parameters to determine when to use clock gating and the type of the clock gating used. The clock gating can be latch based or latch less based. Read the online documentation to find the difference between these two styles.

_____
_____
_____
_____

2. In this lab task, we will use latch based clock gating style. Use the following command before analyzing the source files: set_clock_gating_style -sequential_cell latch
3. While compiling the design, you have to add the parameter -gate_clock to implement the clock gating function.
4. Report Following:
   Area: _____ Slack: _____ Power (Total):_____
   OperatingCondition: _____ WireLoadModel: _____
   WireLoadModel Mode: _____

4.2. Operand Isolation:

In order to enable operand isolation, You need to set the following variable in your synthesis script.

1. set do_operand_isolation "true"

Question: Re-synthesise your design using clock Gating and Operand Isolation. Fill the following tales

Design Name	Power Saving Technique	Area	Timing	Power
	None			
	Clockgating and Operand Isolation			

Table-IV: Effect of Clock Gating and operand isolation (Report power with Saif)

Design Name	Power Saving Technique	Area	Timing	Power
	None			
	Clockgating and Operand Isolation			

Table-V: Effect of Clock Gating and operand isolation (Report power without Saif)

5. Hierarchal Bottom Up Synthesis

In this section we will look into a bottom up synthesis technique. Bottom up synthesis has the following advantage over the top down synthesis;

1. Large designs are compiled with divide and conquer approach.
2. Bottom up synthesis is not limited by the size of available memory. In top down synthesis, available memory is a major limiting factor for the size of the design.

Disadvantages include;

1. It is cumbersome to do bottom up synthesis since it may involve much iteration to get it done.
2. Careful revision control is required to keep track of changes in the design.

The bottom-up compile strategy requires these steps:

1. Develop both a default constraint file and subdesign-specific constraint files. The default constraint file includes global constraints, such as the clock information and the drive and load estimates. The subdesign-specific constraint files reflect the time budget allocated to the subblocks.
2. Compile the subdesigns independently.
3. Read in the top-level design and any compiled subdesigns not already in memory.
4. Set the current design to the top-level design, link the design, and apply the top-level constraints. If the design meets its constraints, you are finished. Otherwise, continue with the following steps.
5. Apply the characterize command to the cell instance with the worst violations.
6. Use write_script to save the characterized information for the cell. You use this script to re-create the new attribute values when you are recompiling the cell's referenced subdesign.
7. Use remove_design -all to remove all designs from memory.
8. Read in the RTL design of the previously characterized cell. Recompiling the RTL design instead of the cell's mapped design usually leads to better optimization.
9. Set current_design to the characterized cell's subdesign and recompile, using the saved script of characterization data.
10. Read in all other compiled subdesigns.
11. Link the current subdesign.
12. Choose another subdesign, and repeat steps 3 through 9 until you have recompiled all subdesigns, using their actual environments.

A pseudo code of the compile script is given in Appendix A.

Task: A basic bottom up synthesis script, named bottomup_outline.scr is given in the LAB3 directory under SCRIPT directory. This script is not automatic, so for every pass you need to execute it using File->Execute Script in Design Vision. Your task is to complete the script, and execute it to perform bottom up synthesis of your code. Set the clock period to 2ns.

Report Following:

Area: _____ Slack: _____ Power (Total):_____
OperatingCondition: _____ WireLoadModel: _____
WireLoadModel Mode: _____

Appendix- A

script

```
all_blocks = {E,D,C,B,A}
/* compile each subblock independently */
foreach (block, all_blocks) {
/* read in block */
block_source = block + ".v"
read_file -format verilog block_source
current_design block
link
uniquify
/* apply global attributes and constraints */
include defaults.con
/* apply block attributes and constraints */
block_script = block + ".con"
include block_script
/* compile the block */
compile
}
/* read in entire compiled design */
read_file -format verilog TOP.v
current_design TOP
link
write -hierarchy -output first_pass.db
/* apply top-level constraints */
include defaults.con
include top_level.con
/* check for violations */
report_constraint
/* characterize all instances in the design */
all_instances = {U1,U2,U2/U3,U2/U4,U2/U5}
characterize -constraint all_instances
/* save characterize information */
foreach (block, all_blocks) {
current_design block
char_block_script = block + ".wscr"
write_script > char_block_script
}
```

```
/* recompile each block */
foreach (block, all_blocks) {
/* clear memory */
remove_design -all
/* read in previously characterized subblock */
block_source = block + ".v"
read -format verilog block_source
/* recompile subblock */
current_design block
link
uniquify
/* apply global attributes and constraints */
include defaults.con
/* apply characterization constraints */
char_block_script = block + ".wscr"
include char_block_script
/* apply block attributes and constraints */
block_script = block + ".con"
include block_script
/* recompile the block */
compile
}
```