National Tsing Hua University
Department of Electrical Engineering
EE429200 IC Design Laboratory, Fall 2021

Homework Assignment #1 **(9%)**
Three-input Raster Operations (ROP3) for Computer Graphics
Assigned on Sep 30, 2021
Due by **Oct 14, 2021**

Assignment Description

Raster operation (ROP) provide fundamental bit-level functions for computer graphics, which we are using all the time on the PC and mobile phone. Among them, ROP3 functions are designed for three 8-bit inputs: pattern *P [7:0],* source *S [7:0]*, and background *D [7:0]*. They consist of 256 different functions which are specified by an 8-bit input, *Mode [7:0]*. Table 1. Shows examples of fifteen such ROP3 functions.
(More details for ROP3 can be found in http://www.graphicalweb.net/2003/papers/RasterOperationsUsingFilterElements/)

Pattern (P)	1	1	1	1	0	0	0	0	Mode (Hex)	Function Name	Boolean Equation
Source (S)	1	1	0	0	1	1	0	0			
Background (D)	1	0	1	0	1	0	1	0			
Result	0	0	0	0	0	0	0	0	8'h00	BLACKNESS	0
	0	0	0	1	0	0	0	1	8'h11	NOTSRCERASE	~(D|S)
	0	0	1	1	0	0	1	1	8'h33	NOTSRCCOPY	~S
	0	1	0	0	0	1	0	0	8'h44	SRCERASE	S&~D
	0	1	0	1	0	1	0	1	8'h55	DSTINVERT	~D
	0	1	0	1	1	0	1	0	8'h5A	PATINVERT	D^P
	0	1	1	0	0	1	1	0	8'h66	SRCINVERT	D^S
	1	0	0	0	1	0	0	0	8'h88	SRCAND	D&S
	1	0	1	1	1	0	1	1	8'hBB	MERGEPAINT	D|~S
	1	1	0	0	0	0	0	0	8'hC0	MERGECOPY	P&S
	1	1	0	0	1	1	0	0	8'hCC	SRCCOPY	S
	1	1	1	0	1	1	1	0	8'hEE	SRCPAINT	D|S
	1	1	1	1	0	0	0	0	8'hF0	PATCOPY	P
	1	1	1	1	1	0	1	1	8'hFB	PATPAINT	D|P|~S
	1	1	1	1	1	1	1	1	8'hFF	WHITENESS	1

Table 1: Fifteen ROP3 functions with their modes and Boolean equations. The middle columns show bit-level examples for P [7:0] = 8'hF0, S [7:0] = 8'hCC, and D [7:0] = 8'hAA.

ROP3 works on each bit position i independently, and the formulation is given as follows:

```
temp1 [7:0] = 8'h1 << {P[i], S[i], D[i]};
temp2 [7:0] = temp1 [7:0] & Mode [7:0];
Result [i] = | temp2 [7:0];
```

It is encouraged to verify this simple formulation with the Boolean functions in Table 1. Note that if the 256

functions are implemented in brute-force ways and selected by a large multiplexer, it could consume much more hardware resource.

(We will verify this statement after introducing synthesis tools in **Homework 3**. Please look forward to it.)

In this assignment, you need to design this ROP3 module in Verilog RTL (Figure 1) using different implementation methods. After writing the RTL code, you need to test it with the testbench either provided by TA (Part 1) or written by yourself (Part2, 3).

Details are as follows:

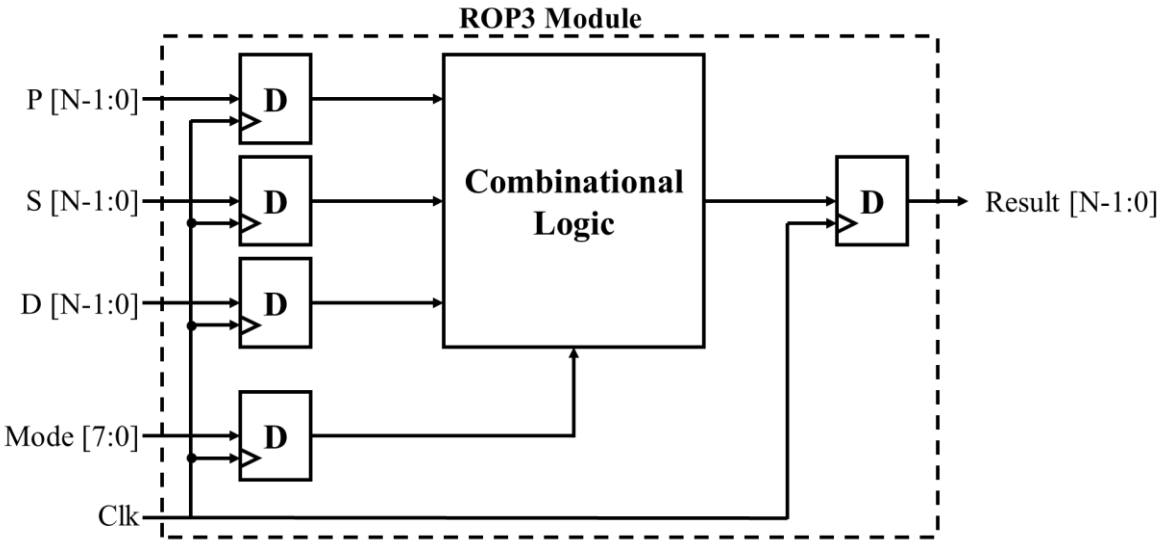

Figure 1: Module rop3 schematic view

Part 1 (1%)

For the first part, design a Verilog module (\HW1\hdl\rop3_lut16.v) that implements those fifteen functions listed in Table1. Note that this module should be implemented using multiplexer (i.e. look-up table). For those modes which do not appear in Table 1, set the computation result to zero.

After finishing the RTL, test it with the testbench provided by TA.

1. Complete \HW1\hdl\rop3_lut16.v
2. Change directory to \HW1\sim\part1\
3. Run ***$ ncverilog -f test_rop3_lut16.f***
4. Make sure you pass the test

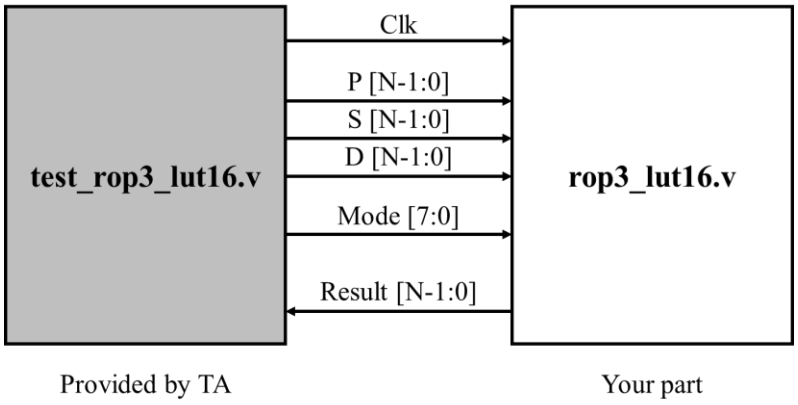

Figure 2: Part 1 Architecture

Part 2 (3%)

In this part, you need to implement the ROP3 function using the magical formulation mentioned above. After finishing the RTL (\HW1\hdl\rop3_smart.v), you need to test it by comparing its behavior with the RTL you wrote in Part 1(rop3_lut16.v).

1. Complete \HW1\hdl\rop3_smart.v
2. Change directory to \HW1\sim\part2\, write a testbench test_rop3_smart.v to test your RTL.
   *This testbench should send identical inputs to both modules(rop3_lut16&rop3_smart), and compare their computation results. A reference testing architecture is shown in Figure 3.
   *This testbench should generate all the modes listed in Table 1 and all possible combinations of P, S and D. Try to use for-loop to generate these input stimuli.
3. Write a README.txt to describe how you organize your testbench to test the RTL.
4. You can dump the waveform for debugging in the testbench, but remember to delete them (*.fsdb and statements in testbench) when submitting.

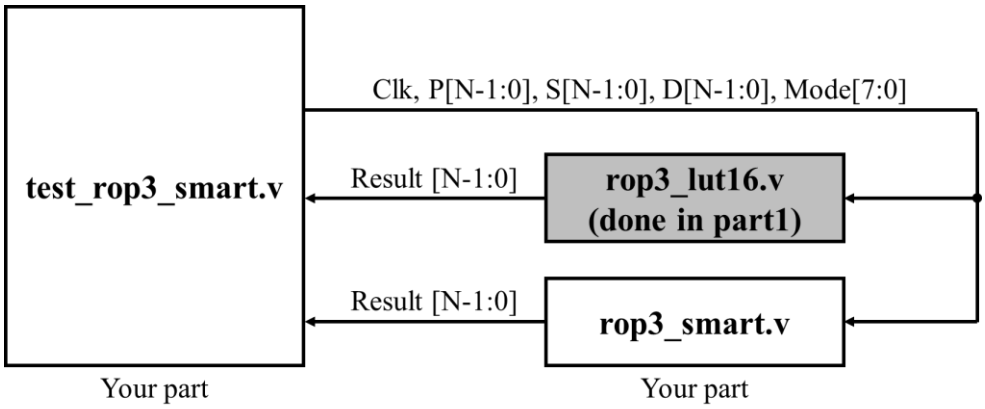

Figure 3: Part 2 Architecture

Part 3 (3%)

There are actually 256 possible functions for ROP3, but we only list 15 of them in Table 1. Find out all the 256 Boolean functions for ROP3, and use a large multiplexer to implement it. After finishing the RTL, try to compare the results with rop3_smart.v you implemented in Part2.

(Note: The overall simulation may take more than ten minutes. You may use $display to inform some internal status, e.g. display a message whenever a new function mode is tested.)

(Hint: Try to observe Table 1 carefully, the bit example of {P, S, D} we show in Table 1 is meaningful)

1. Complete \HW1\hdl\rop3_lut256.v
2. Change directory to \HW1\sim\part3\, write a testbench test_rop3.v to test your RTL.
   *This testbench should send identical inputs to both modules (rop3_smart&rop3_lut256), and compare their computation results. A reference testing architecture is shown in Figure 4.
   *This testbench should generate all of the 256 modes and all possible combinations of P, S and D. Try to use for-loop to generate these input stimuli.
3. Write a README.txt to describe how you organize your testbench to test the RTL.
4. You can dump the waveform for debugging in the testbench, but remember to delete them (*.fsdb and statements in testbench) when submitting.

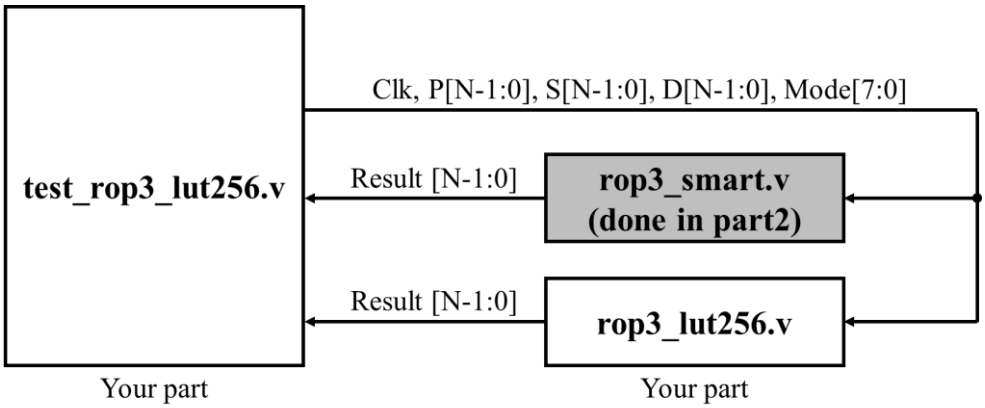

Figure 4: Part 3 Architecture

Part 4 (2%)

In this part, you need to integrate the following two features in your testbench \HW1\sim\part4\test_rop3.v and \HW1\sim\part4\rop3.task for better readability and scalability. Note that you should verify all 256 possible modes.

1. Use a text substitution macro (`define)

   *Since running all possible functions is time-consuming, sometimes we will just run certain test data for quick verification at the very first development stage. Try to use two substitution macros (e.g. MODE_L, MODE_U) to define the lower and upper bound of the mode respectively, and directly assign them when compiling the simulation.

2. Use tasks

   *Use tasks to make your code more readable. Complete the two tasks in \HW1\sim\part4\rop3.task for input data preparation and output data comparison.

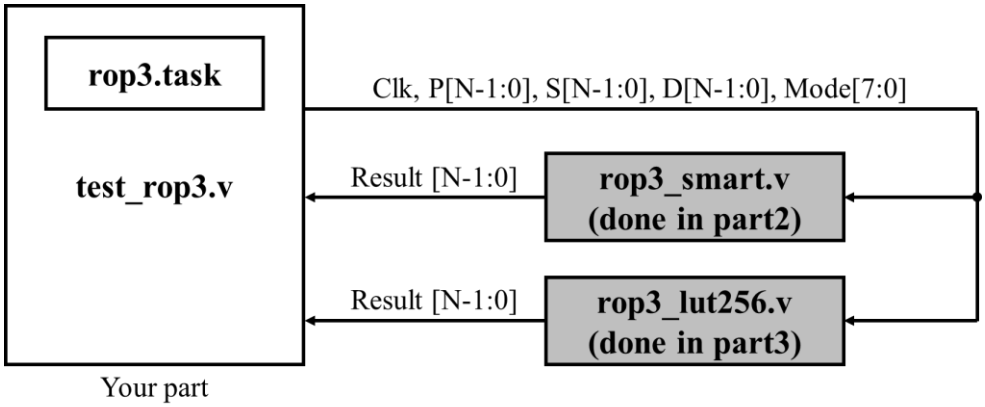

Figure 5: Part 4 Architecture

RTL I/O name definition

I/O name	Number of bits	Description
clk	1	Clock input
P	N	Pattern input
S	N	Source input
D	N	Background input
Mode	8	Function mode input
Result	N	Your computation result output

Deliverable

1. Synthesizable Verilog **rop3_lut16.v**, **rop3_lut256.v** and **rop3_smart.v**.
2. Spyglass report **rop3_lut16.rpt**, **rop3_lut256.rpt** and **rop3_smart.rpt**.
3. Testbench **test_rop3_smart.v** (Part 2), **test_rop3_lut256.v** (Part 3) and **test_rop3.v** (Part 4) .
4. Task file **rop3.task** (Part4).
5. One text file README.txt to describe:
    - How you organize the testbench in Part 2, Part 3 and Part4.
    - How you find out all the 256 functions.
    - Corresponding commands for mode=0-63, 64-127 and 128-255 in **test_rop3.v** (Part4).

      *For example:

      $ ncverilog test_rop3.v rop3_lut256.v rop3_smart.v +define+ MODE_L=0+MODE_U=63

Note: 1% punishment for wrong file delivery.

File Organization

Directory	Filename	Description
hdl/	rop3_lut16.v	Part 1 module
hdl/	rop3_smart.v	Part 2 module
hdl/	rop3_lut256.v	Part 3 module
sim/part_1/	test_rop3_lut16.f	Part 1 RTL pre-simulation file list
sim/part_1/	test_rop3_lut16.v	Part 1 testbench
sim/part_2/	test_rop3_smart.v	Part 2 testbench
sim/part_3/	test_rop3_lut256.v	Part 3 testbench
sim/part_4/	test_rop3.v	Part 4 testbench
sim/part_4/	rop3.task	Part4 task file

Note

1. The word length N should be parameterizable in both RTL and testbench files. You should test them with at least two settings, e.g. 4 and 8, before delivering your homework. We will run simulation with different settings to test your codes.
2. All RTLs should be **synthesizable**. i.e. rop3_lut16, rop3_lut256 and rop3_smart should pass Spyglass examination before submitting.