National Tsing Hua University
Department of Electrical Engineering
EE429200 IC Design Laboratory, Fall 2022

Homework Assignment #1 **(9%)**
Three-input Raster Operations (ROP3) for Computer Graphics
Assigned on Sep 29, 2022
Due by **Oct 13, 2022**

Assignment Description

Raster operation (ROP) provides fundamental bit-level functions for computer graphics, which we are using all the time on the PCs and mobile phones. Among them, ROP3 functions are designed for three 8-bit inputs: pattern P [7:0], source S [7:0], and destination 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.

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 second column shows bit-level results for P [7:0] = 8'hF0, S [7:0] = 8'hCC, and D [7:0] = 8'hAA under different Modes.

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 would 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 (Part 2, Part 3).

Details are as follows:

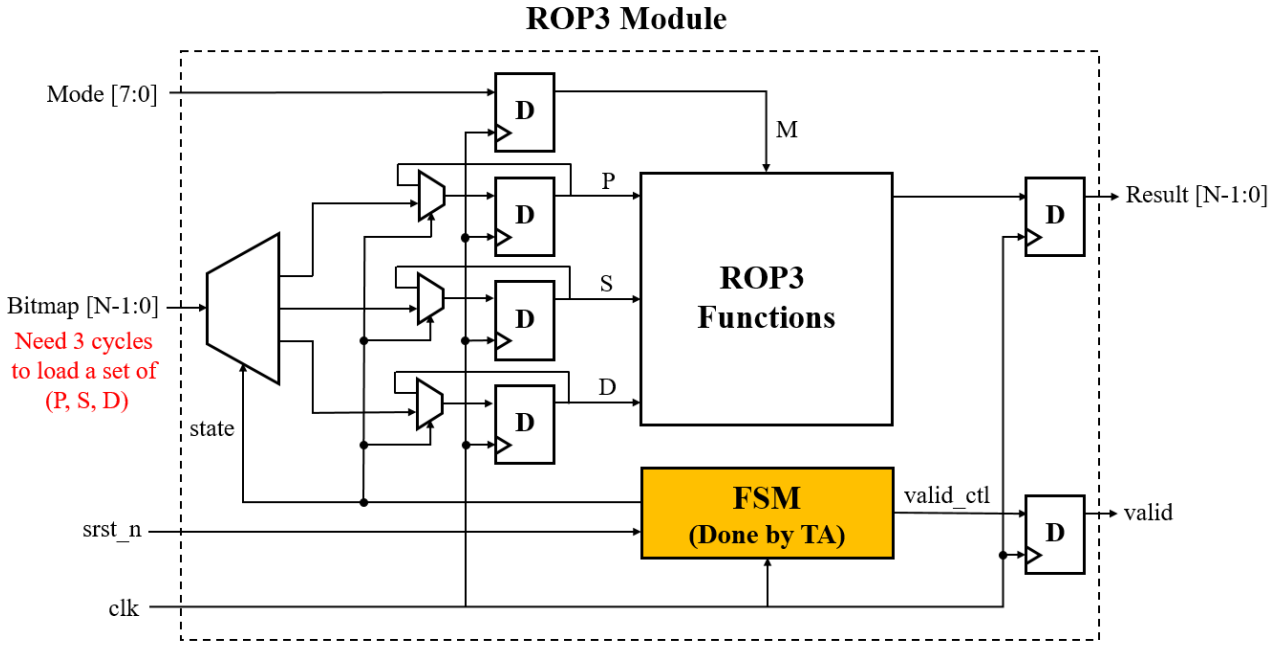

Figure 1: Schematic view of ROP3 Module

Type	Name	Number of bits	Description
Input	clk	1	Clock
Input	srst_n	1	System reset (synchronous, active low)
Input	Mode	8	ROP3 function mode
Input	Bitmap	N	Input to load P, S, D sequentially
Output	Result	N	ROP3 computation result
Output	valid	1	Indicate that the result is valid when it is high

Table 2: I/O Description of ROP3 Module

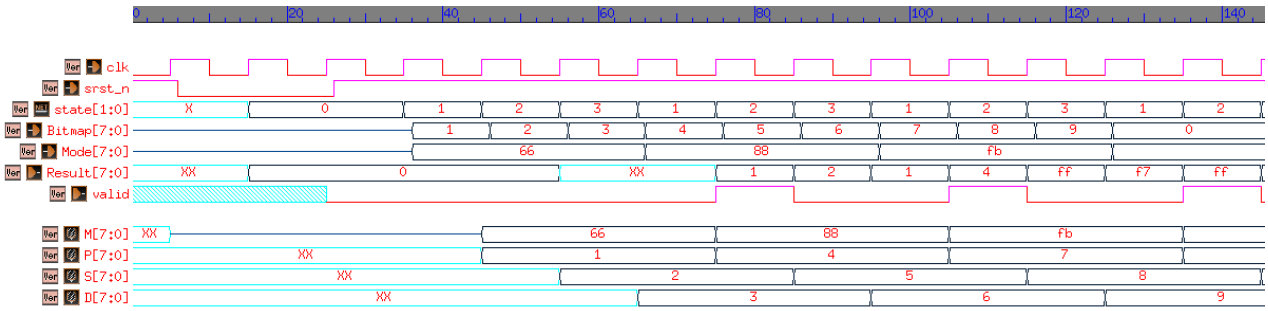

Figure 2: Timing Diagram of ROP3 Module

Figure 1 shows the schematic view of the ROP3 module. Note that in the module, the FSM (Finite State Machine) part has been done by TA. Do not modify this part. You need to implement the remaining part in this figure. Please remember to

(1) make signals {M, P, S, D, Result, valid} registered

(2) make the bit-length of {Bitmap, Result} parameterizable.

Figure 2 shows the timing diagram of the ROP3 module. Assume we simulate the following three patterns sequentially:

(1) {Mode, P, S, D} = {8'h66, 8'h01, 8'h02, 8'h03}

(2) {Mode, P, S, D} = {8'h88, 8'h04, 8'h05, 8'h06}

(3) {Mode, P, S, D} = {8'hfb, 8'h07, 8'h08, 8'h09}

First, we set the reset signal low to reset the finite state machine. After reset, the state signal becomes zero, which means the circuit is in the IDLE state. Then, the state signal turns into one, two, three sequentially with each state kept for one cycle, and repeats this behavior. In state one, P of the pattern is fed through the Bitmap input port. In state two, S of the pattern is fed through the Bitmap input port. In state three, D of the pattern is fed through the Bitmap input port.

Because we need three cycles to feed a complete pattern, the ROP3 computation result would not be valid continuously. Instead, there would be one valid result every three cycles. Therefore, we use a valid signal to indicate whether the output result is valid or not. Take the above three patterns as an example, the ROP3 computation result of the first pattern, the second pattern, and the third pattern is 8'h01, 8'h04, and 8'hff respectively. In Figure 2, we can see that the valid signal is high when the Result signal matches these three values.

It is important to keep in mind that this circuit outputs one valid result every three cycles. When writing RTL design, make sure the output Result of your module is correct when the valid signal is high. When writing testbench, notice that you should only take output Result for comparison when the valid signal is high.

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. Run Spyglass examination to make sure your RTL design is synthesizable
3. Change directory to HW1/sim/part1
4. Edit test_rop3_lut16.sh
   (1) To simulate N=4 (4-bit case), modify the command in test_rop3_lut16.sh as below
   vcs -f test_rop3_lut16.f -full64 -R -debug_access+all +v2k +define+N=4
   (2) To simulate N=8 (8-bit case), modify the command in test_rop3_lut16.sh as below
   vcs -f test_rop3_lut16.f -full64 -R -debug_access+all +v2k +define+N=8
5. Execute test_rop3_lut16.sh by ***$ sh test_rop3_lut16.sh***
6. Make sure you pass both tests (4-bit case & 8-bit case)

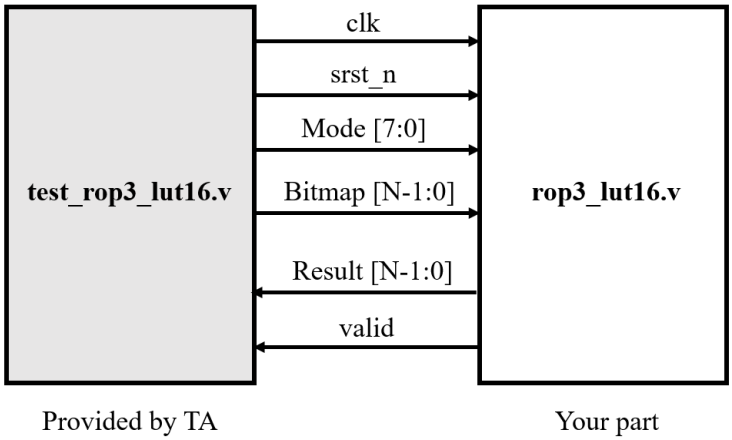

Figure 3: Part 1 Architecture

Part 2 (3%)

In this part, you need to implement the ROP3 function using the formulation mentioned in the below of page 1. 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. Run Spyglass examination to make sure your RTL design is synthesizable
3. Change directory to HW1/sim/part2, write a testbench test_rop3_smart.v to test your RTL. Your testbench should meet some requirements listed below:
   (1) 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 4.

(2) This testbench should generate all the modes listed in Table 1 and all possible combinations of P, S and D for each mode. Try to use for-loop and traverse with order

$$\text{Mode} \rightarrow \text{P} \rightarrow \text{S} \rightarrow \text{D}$$

to generate these input patterns.

Take N=1 (1-bit case) as an example, the order of your input patterns should be like:

{Mode, P, S, D} = {8'h00, 0, 0, 0}
{Mode, P, S, D} = {8'h00, 0, 0, 1}
{Mode, P, S, D} = {8'h00, 0, 1, 0}
{Mode, P, S, D} = {8'h00, 0, 1, 1}
{Mode, P, S, D} = {8'h00, 1, 0, 0}
{Mode, P, S, D} = {8'h00, 1, 0, 1}
{Mode, P, S, D} = {8'h00, 1, 1, 0}
{Mode, P, S, D} = {8'h00, 1, 1, 1}
{Mode, P, S, D} = {8'h11, 0, 0, 0}
{Mode, P, S, D} = {8'h11, 0, 0, 1}
{Mode, P, S, D} = {8'h11, 0, 1, 0}
⋮

(3) Use $display to inform some internal status during simulation. At least display a message whenever a new function mode is tested such as below:

```
Simulate function mode 00 ...
Simulate function mode 11 ...
Simulate function mode 33 ...
Simulate function mode 44 ...
Simulate function mode 55 ...
Simulate function mode 5a ...
Simulate function mode 66 ...
Simulate function mode 88 ...
Simulate function mode bb ...
Simulate function mode c0 ...
Simulate function mode cc ...
Simulate function mode ee ...
Simulate function mode f0 ...
Simulate function mode fb ...
Simulate function mode ff ...

============= Congratulations =============
             All patterns pass !
============= Congratulations =============
```

(4) When the simulation is finished, this testbench should output a file called "sim_out_part2.csv" which contains all the input patterns and their ROP3 computation results during the simulation. A csv (comma-separated values) file is a text file that has a specific format which allows data to be saved in a table structured format. A reference output format is given below (use N=1 1-bit case as example):

```
Mode,P,S,D,Result_lut16,Result_smart
00,0,0,0,0,0
00,0,0,1,0,0
00,0,1,0,0,0
00,0,1,1,0,0
00,1,0,0,0,0
00,1,0,1,0,0
00,1,1,0,0,0
00,1,1,1,0,0
11,0,0,0,1,1
11,0,0,1,0,0
11,0,1,0,0,0
```

Please save the values with hexadecimal format.

Below is a toy example to show how to output a csv file and write values with hexadecimal format in the testbench:

```
// file declaration
integer fout;
// create output file
fout = $fopen("toy_example.csv");
// write title
$fwrite(fout, "INPUT_A,INPUT_B,OUTPUT_C\n");
// write values
$fwrite(fout, "%h,%h,%h\n", INPUT_A, INPUT_B, OUTPUT_C);
// close file
$fclose(fout);
```

Note:

It is suggested to check the output csv file with vim. If you open it with Excel, Notepad, or VS code, a warning may occur when the number of rows (simulated patterns) in the file exceed the upper limit supported by these text editors.

(5) Use *`define N* in the testbench to make the bit-length of {Bitmap, Result} parameterizable. By doing so, TA could run different N (bit-length) by simply modifying the simulation command as mentioned in step 4 of Part 1.

(6) Make sure your RTL design and testbench pass under different bit-length N, where N ∈ {1, 2, 3, 4, 5, 6}. It is not mandatory to run simulation with bit-length larger than 6 because it may take a long time and the size of the output csv file would be large. However, it is still encouraged to test your design with larger bit-length if you have sufficient time and memory spaces.

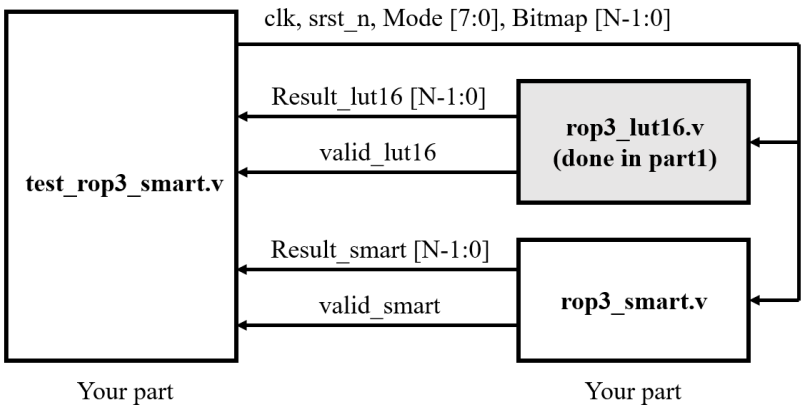

Figure 4: Part 2 Architecture

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, you need to test it by comparing its behavior with the RTL you wrote in Part 2 (rop3_smart.v).

Hint:

Try to observe Table 1 carefully, the bit example of {P, S, D} we show in Table 1 is meaningful. It is feasible to infer the remaining functions by Table 1.

1. Complete HW1/hdl/rop3_lut256.v
2. Run Spyglass examination to make sure your RTL design is synthesizable
3. Change directory to HW1/sim/part3, write a testbench test_rop3_lut256.v to test your RTL.
   Your testbench should meet some requirements listed below:
   (1) This testbench should send identical inputs to both modules (rop3_lut256 & rop3_smart), and compare their computation results. A reference testing architecture is shown in Figure 5.

   (2) This testbench should generate all 256 modes and all possible combinations of P, S and D for each mode. Try to use for-loop and traverse with order

   $$\text{Mode} \rightarrow \text{P} \rightarrow \text{S} \rightarrow \text{D}$$

   to generate these input patterns.
   An example is already provided in Part 2.

   (3) Use $display to inform some internal status during simulation. At least display a message whenever a new function mode is tested. An example is already provided in Part 2.

   (4) When the simulation is finished, this testbench should output a file called “sim_out_part3.csv” which contains all the input patterns and their ROP3 computation results during the simulation. A reference output format is given below (use N=1 1-bit case as example):

```
Mode,P,S,D,Result_lut256,Result_smart
00,0,0,0,0,0
00,0,0,1,0,0
00,0,1,0,0,0
00,0,1,1,0,0
00,1,0,0,0,0
00,1,0,1,0,0
00,1,1,0,0,0
00,1,1,1,0,0
01,0,0,0,1,1
01,0,0,1,0,0
01,0,1,0,0,0
```

Please save the values with hexadecimal format.

(5) Use *`define N* in the testbench to make the bit-length of {Bitmap, Result} parameterizable. By doing so, TA could run different N (bit-length) by simply modifying the simulation command as mentioned in step 4 of Part 1.

(6) Make sure your RTL design and testbench pass under different bit-length N, where N ∈ {1, 2, 3, 4, 5, 6}. It is not mandatory to run simulation with bit-length larger than 6 because it may take a long time and the size of the output csv file would be large. However, it is still encouraged to test your design with larger bit-length if you have sufficient time and memory spaces.

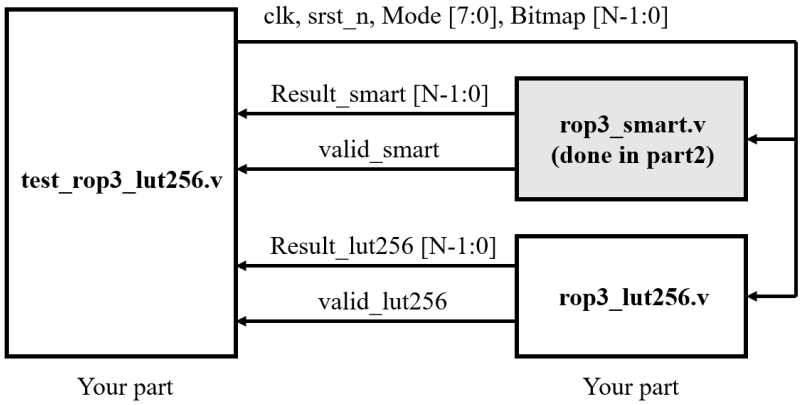

Figure5: Part 3 Architecture

Part 4 (1%)

Since running all possible combinations with larger bit-length is time-consuming, sometimes we will just run certain test data for quick verification at the very first development stage. Based on the testbench written in Part 3, write another testbench test_rop3_lut256_quick_verify.v by using *`define* to add three substitution macros: STEP_P, STEP_S, and STEP_D to control the step of each input in its for-loop when generating input patterns.

For example, if we simulate with command:

```
vcs -f test_rop3_lut16_quick_verify.f -full64 -R -debug_access+all +v2k \
+define+N=4+STEP_P=2+STEP_S=4+STEP_D=8
```

which means simulate with N=4, STEP_P=2, STEP_S=4, STEP_D=8

The order of the generated input patterns for each mode should be like:

{P, S, D} = {4'h0, 4'h0, 4'h0}
{P, S, D} = {4'h0, 4'h0, 4'h8}
{P, S, D} = {4'h0, 4'h4, 4'h0}
{P, S, D} = {4'h0, 4'h4, 4'h8}
{P, S, D} = {4'h0, 4'h8, 4'h0}
{P, S, D} = {4'h0, 4'h8, 4'h8}
{P, S, D} = {4'h0, 4'hc, 4'h0}
{P, S, D} = {4'h0, 4'hc, 4'h8}
{P, S, D} = {4'h2, 4'h0, 4'h0}
{P, S, D} = {4'h2, 4'h0, 4'h8}
{P, S, D} = {4'h2, 4'h4, 4'h0}
{P, S, D} = {4'h2, 4'h4, 4'h8}
{P, S, D} = {4'h2, 4'h8, 4'h0}
{P, S, D} = {4'h2, 4'h8, 4'h8}
{P, S, D} = {4'h2, 4'hc, 4'h0}
{P, S, D} = {4'h2, 4'hc, 4'h8}
{P, S, D} = {4'h4, 4'h0, 4'h0}
{P, S, D} = {4'h4, 4'h0, 4'h8}
{P, S, D} = {4'h4, 4'h4, 4'h0}
{P, S, D} = {4'h4, 4'h4, 4'h8}
{P, S, D} = {4'h4, 4'h8, 4'h0}
{P, S, D} = {4'h4, 4'h8, 4'h8}
{P, S, D} = {4'h4, 4'hc, 4'h0}
{P, S, D} = {4'h4, 4'hc, 4'h8}
⋮

Change directory to HW1/sim/part4, write a testbench test_rop3_lut256_quick_verify.v to test your RTL.

Your testbench should meet some requirements listed below:

(1) The same as in Part 3.
(2) The same as in Part 3.
(3) The same as in Part 3.
(4) The same as in Part 3, except that the output csv filename should be "sim_out_part4.csv".
(5) The same as in Part 3.
(6) Try to run simulation with different settings of {N, STEP_P, STEP_S, STEP_D}.
Make sure your RTL design and testbench pass under different simulation settings.

ReadMe (1%)

Write a ReadMe.txt to describe

(1) what you observe in Table 1 and how you find out all 256 functions for ROP3 in Part 3

(2) how you organize your testbench to test your RTL design in Part 2 & Part 3

Note

1. All RTLs should be synthesizable. i.e. rop3_lut16.v, rop3_smart.v and rop3_lut256.v should pass Spyglass examination.

2. The bit-length N should be parameterizable in both RTL and testbench files. We will run simulation with different settings to test your codes. Use *`define N* in the testbench so as we can simply modify the simulation command as described in step 4 of Part 1 to test different bit-length.

3. It may take a long time when simulating with larger bit-length N, it is suggested to simulate with smaller bit-length at the very first development stage for quick debugging.

Deliverable

1. **Synthesizable verilog**
   - fsm.v
   - rop3_lut16.v
   - rop3_smart.v
   - rop3_lut256.v
2. **Data**
   - rop3_lut16_N4_input.dat
   - rop3_lut16_N4_golden.dat
   - rop3_lut16_N8_input.dat
   - rop3_lut16_N8_golden.dat
3. **Testbench**
   - test_rop3_lut16.v
   - test_rop3_smart.v
   - test_rop3_lut256.v
   - test_rop3_lut256_quick_verify.v
4. **Command-line arguments file**
   - test_rop3_lut16.f
   - test_rop3_smart.f
   - test_rop3_lut256.f
   - test_rop3_lut256_quick_verify.f
5. **Shell script**
   - test_rop3_lut16.sh
   - test_rop3_smart.sh

- test_rop3_lut256.sh
- test_rop3_lut256_quick_verify.sh

6. **Text file**
   - ReadMe.txt

File Organization

Directory	Filename
HW1_{studentID}/	ReadMe.txt
HW1_{studentID}/hdl/	fsm.v
	rop3_lut16.v
	rop3_smart.v
	rop3_lut256.v
HW1_{studentID}/sim/part1	test_rop3_lut16.v
	test_rop3_lut16.f
	test_rop3_lut16.sh
HW1_{studentID}/sim/part1/data/	rop3_lut16_N4_input.dat
	rop3_lut16_N4_golden.dat
	rop3_lut16_N8_input.dat
	rop3_lut16_N8_golden.dat
HW1_{studentID}/sim/part2/	test_rop3_smart.v
	test_rop3_smart.f
	test_rop3_smart.sh
HW1_{studentID}/sim/part3/	test_rop3_lut256.v
	test_rop3_lut256.f
	test_rop3_lut256.sh
HW1_{studentID}/sim/part4/	test_rop3_lut256_quick_verify.v
	test_rop3_lut256_quick_verify.f
	test_rop3_lut256_quick_verify.sh

Note:

If your studentID is 123456789, HW1_{studentID} denotes HW1_123456789.

Compress your whole folder HW1_{studentID} into HW1_{studentID}.zip and submit to eeclass.

Wrong file delivery or wrong file organization will get 1% punishment.