HW1

Ternary Raster Operations (ROP3) for Computer Graphics

張嘉祐

2022/09/29

Introduction

- Raster Operation (ROP)
  - Provides bit-level functions for computer graphics
    e.g. on PCs and mobile phones
  - Consist of total 256 different functions
  - Input
    - [7:0] P (Pattern) ← N-bit in this assignment
    - [7:0] S (Source) ← N-bit in this assignment
    - [7:0] D (Destination) ← N-bit in this assignment
    - [7:0] Mode (Function mode)
  - Output
    - [7:0] Result ← N-bit in this assignment

Fifteen ROP3 functions (for Part 1)

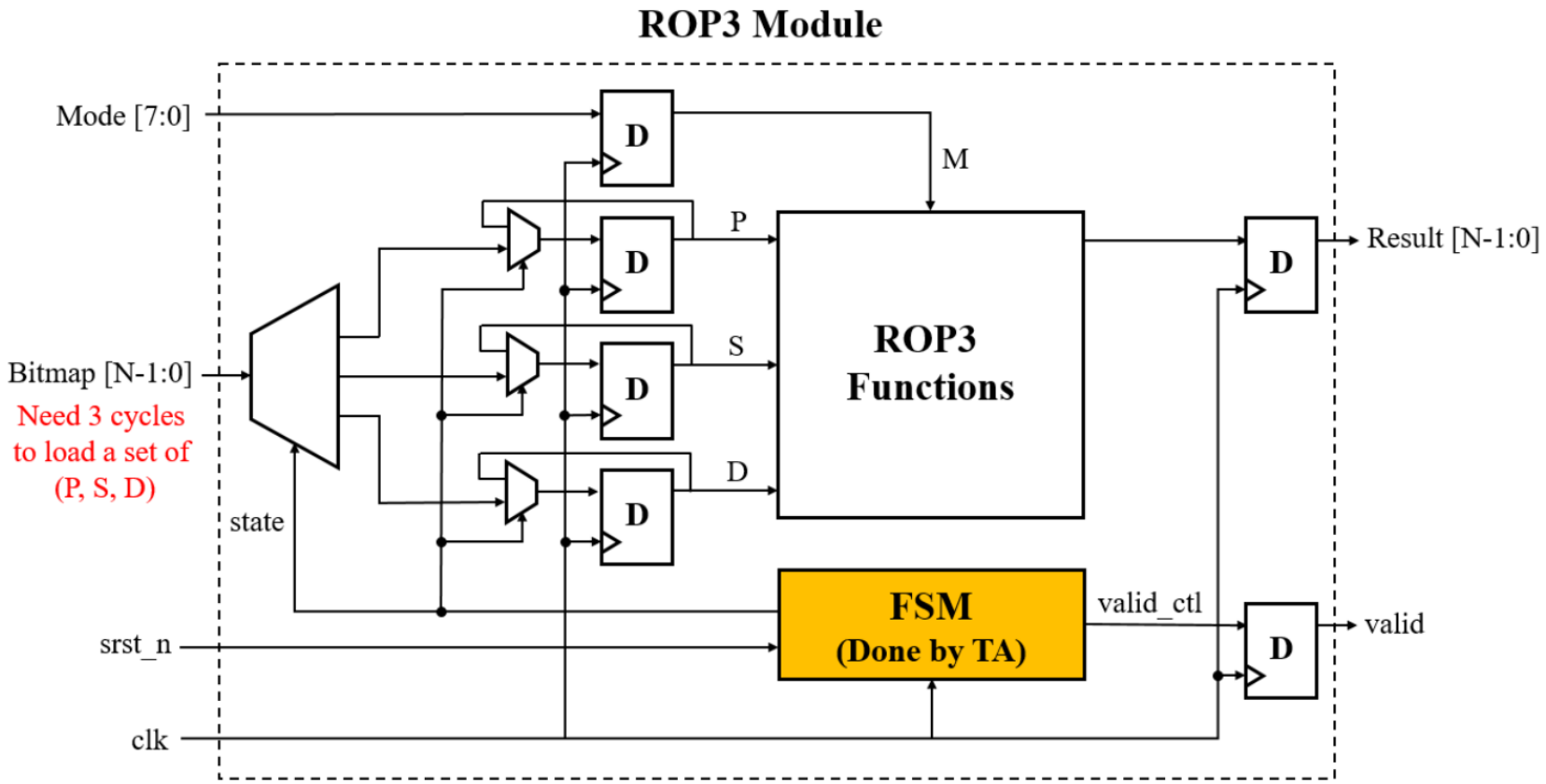
Pattern (P)	1	1	1	1	0	0	0	0	Mode (Hex)	Function Name	Boolean Equation
Source (S)	1	1	0	0	1	1	0	0			
Background (D)	1	0	1	0	1	0	1	0			
Result	0	0	0	0	0	0	0	0	8'h00	BLACKNESS	0
	0	0	0	1	0	0	0	1	8'h11	NOTSRCERASE	~(D|S)
	0	0	1	1	0	0	1	1	8'h33	NOTSRCCOPY	~S
	0	1	0	0	0	1	0	0	8'h44	SRCERASE	S&~D
	0	1	0	1	0	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

Smart formulation of ROP3 (for Part 2)

- Works for each bit position i independently

```
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];
```

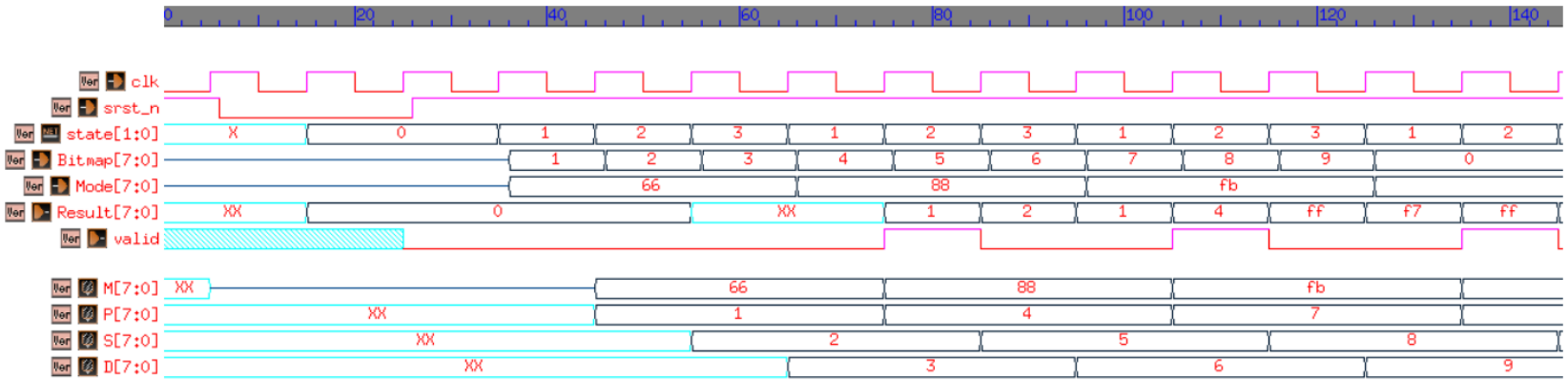
Schematic view



Do not modify the FSM part !

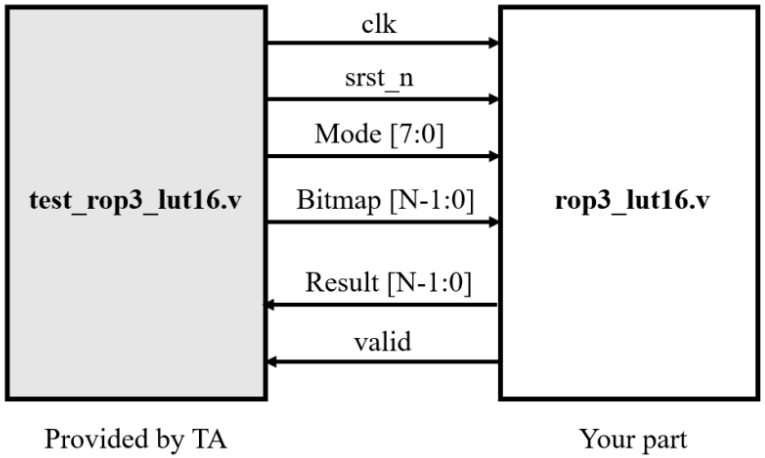
Timing diagram

- Assume we simulate the following 3 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}



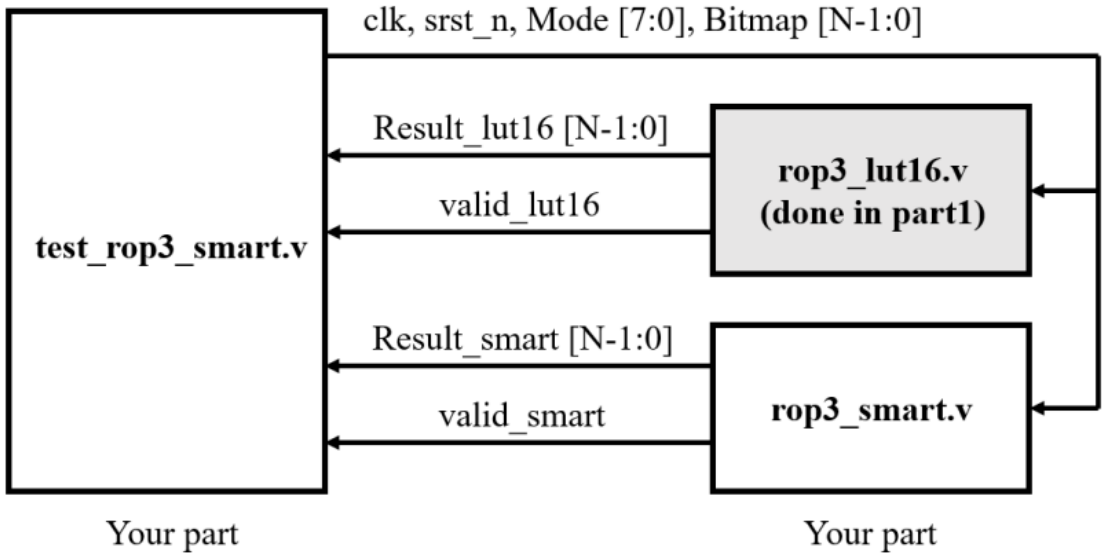
Part 1

- Implement the fifteen ROP3 functions in Table 1
- Complete rop3_lut16.v
- Test your RTL by the provided testbench
- Make sure you pass both tests
  - 4-bit case (N=4) `vcs -f test_rop3_lut16.f -full64 -R -debug_access+all +v2k +define+N=4`
  - 8-bit case (N=8) `vcs -f test_rop3_lut16.f -full64 -R -debug_access+all +v2k +define+N=8`



Part 2

- Implement ROP3 by the smart formulation
- Complete rop3_smart.v
- Write a testbench test_rop3_smart.v to test your RTL



Part 2 Cont'd

- Your testbench should meet some requirements

1) Send identical inputs to rop3_lut16 & rop3_smart
2) Generate all the modes listed in Table 1 and all possible combinations of P, S, D for each mode.

   Use for-loop and traverse with order

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

Take N=1 (1-bit case) as an example

{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}
⋮

Part 2 Cont’d

3) Use *$display* to inform some internal status during simulation. At least display a message whenever a new function mode is tested.

```
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 =============
```

Part 2 Cont'd

4) When the simulation is finished, output a file called "sim_out_part2.csv" which contains all the input patterns and their ROP3 computation results during the simulation.

   Please save the values with hexadecimal format.

Take N=1 (1-bit case) as an 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
```

Part 2 Cont'd

- An example to output a csv file in the testbench

```
1   // file declaration
2   integer fout;
3   // create output file
4   fout = $fopen("toy_example.csv");
5   // write title
6   $fwrite(fout, "INPUT_A,INPUT_B,OUTPUT_C\n");
7   // write values
8   $fwrite(fout, "%h,%h,%h\n", INPUT_A, INPUT_B, OUTPUT_C);
9   // close file
10  $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.

Part 2 Cont'd

5) Use `*define N* in your testbench to make the bit-length of {Bitmap, Result} parameterizable as mentioned in Part 1

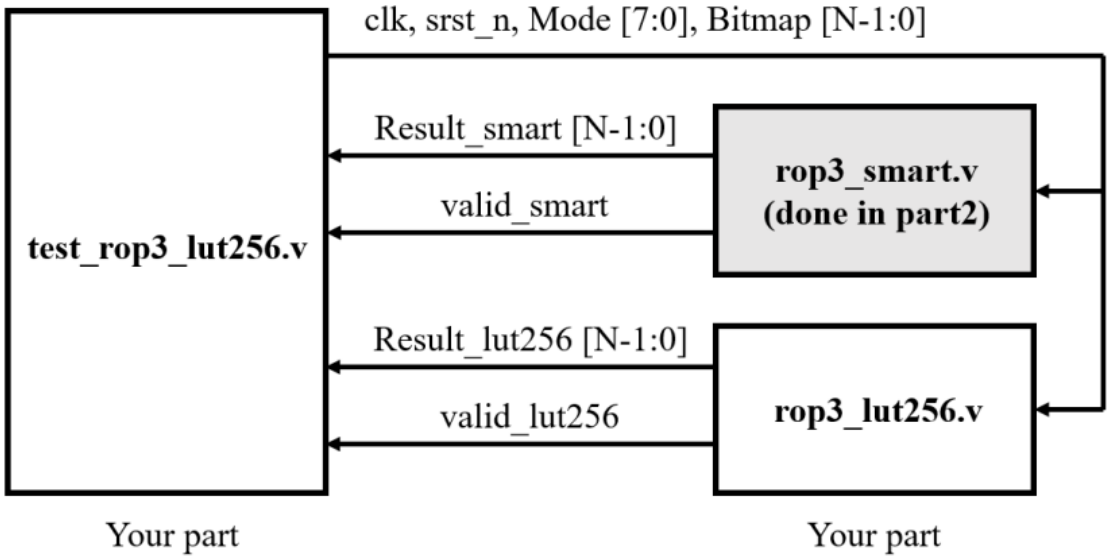
```
vcs -f test_rop3_lut16.f -full64 -R -debug_access+all +v2k +define+N=4
```
for 4-bit case

```
vcs -f test_rop3_lut16.f -full64 -R -debug_access+all +v2k +define+N=8
```
for 8-bit case

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.

Part 3

- Find out all 256 Boolean functions for ROP3, and use a large multiplexer to implement it.
- Complete rop3_lut256.v
- Write a testbench test_rop3_lut256.v to test your RTL



Part 3 Cont'd

- Your testbench should meet some requirements

1) Send identical inputs to rop3_lut256 & rop3_smart
2) Generate all 256 modes and all possible combinations of P, S, D for each mode
3) Use *$display* to inform some internal status during simulation
4) When the simulation is finished, output a file called "sim_out_part3.csv"
5) Use `*define N* in your testbench to make the bit-length of {Bitmap, Result} parameterizable
6) Make sure your RTL design and testbench pass under different bit-length N, where N ∈ {1, 2, 3, 4, 5, 6}

Part 4

- 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
```

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}
⋮

Part 4 Cont'd

- Your testbench should meet some requirements

1) The same as in Part 3.
2) The same as in Part 3.
3) The same as in Part 3.
4) When the simulation is finished, output a file called "sim_out_part4.csv"
5) The same as in Part 3.
6) Make sure your RTL design and testbench pass under different settings of {N, STEP_P, STEP_S, STEP_D}

ReadMe.txt

- 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

Submission

- Deadline
  - 10/13 11:59 p.m.
- Submit to eeclass
  - Make sure the file delivery and organization meet the requirement
  - Wrong file delivery or organization will get 1% punishment
- If you have any question
  - Feel free to ask on eeclass discussion
    (maybe other students also have the same question !)
  - Do not show your code or email your code to TA.
    Coding by yourself !