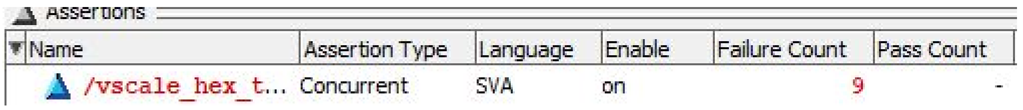
ECE 594BB Lab 2 Report

Author: Zhengqi Yang
Date: 11/17/2019

Note: Assertions see Appendix A

#1 Or assertion

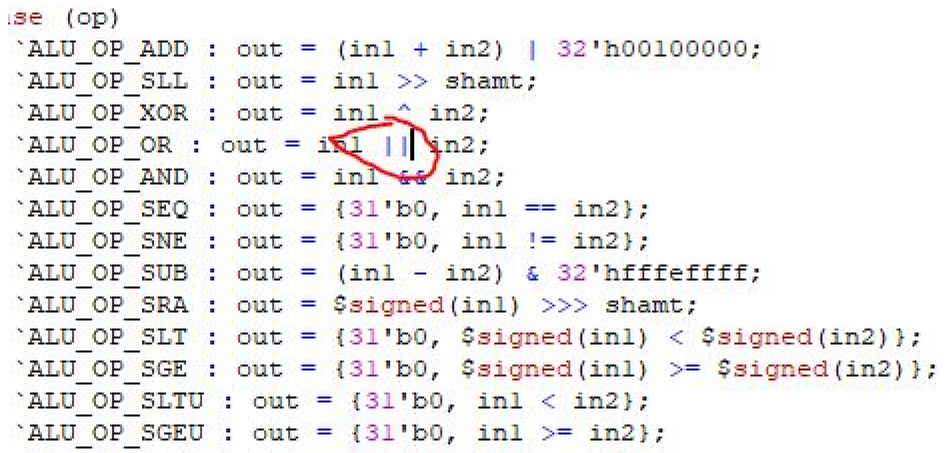
I started debugging with the initial OR assertion by implementing the min_avg_max.asm since it contained a few or statements. By simulating with coverage, the OR assertion did detect errors. Since the input and the output of the OR assertion are from vsacle_alu file, I took a look at the OR statement in this file and found out that the OR statement used there was logical operator instead of bitwise operator. By switching the logical operator into bitwise operator, the errors were cleared shown in Figure 1.



Assertions

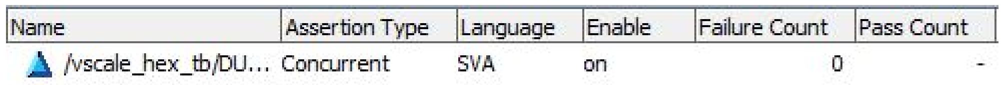
Name	Assertion Type	Language	Enable	Failure Count	Pass Count
/vscale_hex_t...	Concurrent	SVA	on	9	-

Bug location



```
se  (op)
 `ALU_OP_ADD : out = (in1 + in2) | 32'h00100000;
 `ALU_OP_SLL : out = in1 >> shamt;
 `ALU_OP_XOR : out = in1 ^ in2;
 `ALU_OP_OR : out = in1 || in2;
 `ALU_OP_AND : out = in1 && in2;
 `ALU_OP_SEQ : out = {31'b0, in1 == in2};
 `ALU_OP_SNE : out = {31'b0, in1 != in2};
 `ALU_OP_SUB : out = (in1 - in2) & 32'hfffeffff;
 `ALU_OP_SRA : out = $signed(in1) >>> shamt;
 `ALU_OP_SLT : out = {31'b0, $signed(in1) < $signed(in2)};
 `ALU_OP_SGE : out = {31'b0, $signed(in1) >= $signed(in2)};
 `ALU_OP_SLTU : out = {31'b0, in1 < in2};
 `ALU_OP_SGEU : out = {31'b0, in1 >= in2};
```

Bug clear

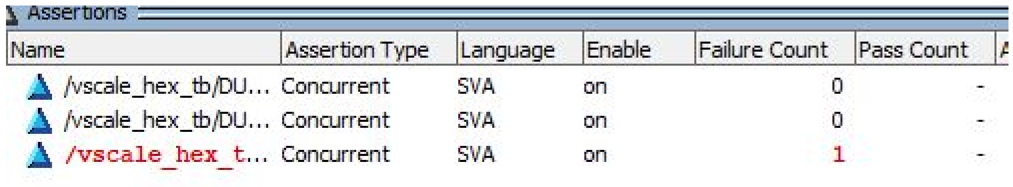


Name	Assertion Type	Language	Enable	Failure Count	Pass Count
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-

Figure 1

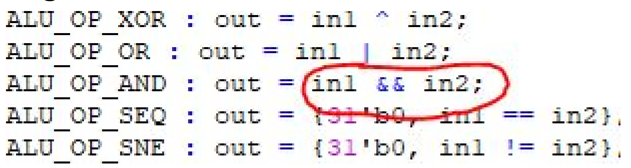
#2 AND assertion

Intuitively, I also checked the AND statement in the vsacle_alu file. As expected, the AND statement there also used the logical operator instead of bitwise operator. In order to prove this, I generated a list of ANDI statements by using constrained_random_generator.py but modifying ORI into ANDI. However, I did not use $zero register in this case since it would always return false with AND operation. With the new AND assertion placed in vscale_pipline, it did detect the error as expected. The reason I put this assertion in vscale_pipline was that AND operation is within ALUand ALU is within vsacle_pipline. After switching logical operator to bitwise operator, the error was cleared.



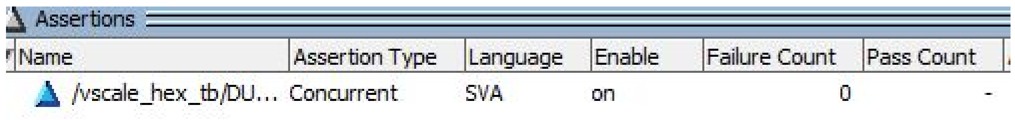
Name	Assertion Type	Language	Enable	Failure Count	Pass Count
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-
/vscale_hex_t...	Concurrent	SVA	on	1	-

Bug location



```
ALU_OP_XOR : out = in1 ^ in2;
ALU_OP_OR : out = in1 | in2;
ALU_OP_AND : out = in1 && in2;
ALU_OP_SEQ : out = {31'b0, in1 == in2};
ALU_OP_SNE : out = {31'b0, in1 != in2};
```

Bug clear



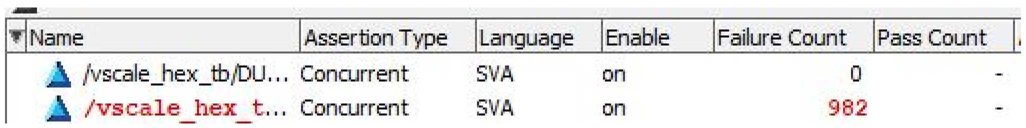
Name	Assertion Type	Language	Enable	Failure Count	Pass Count
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-

Assembly Code

ORI $t1, $zero, 100
ORI $t2, $zero, 200
ANDI $t3, $t1, $t2

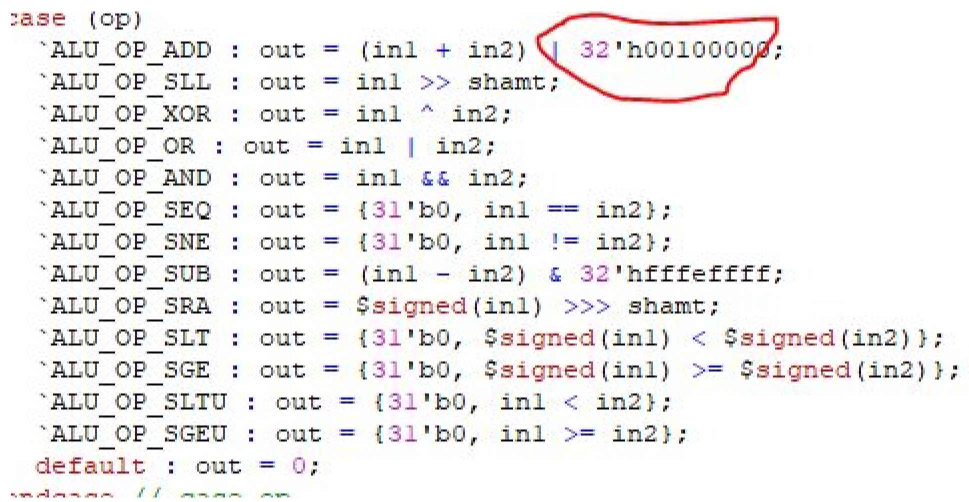
#3 ADD assertion

Then I continued using min_avg_max.asm file to detect other potential bugs. Since this assembly file also contained other kinds of statements such as ADD. In order to check this operator, the add assertion was placed at the same place. By simulating with coverage, it detected errors. By checking the ALU_ADD statement, I found that there was a redundant part. By deleting this part, the errors were cleared.



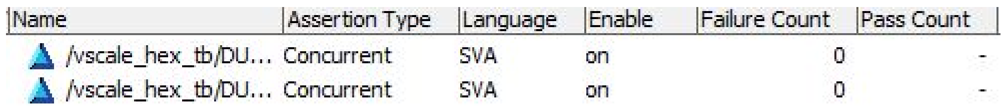
Name	Assertion Type	Language	Enable	Failure Count	Pass Count
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-
/vscale_hex_t...	Concurrent	SVA	on	982	-

Bug location



```
case (op)
  `ALU_OP_ADD : out = (in1 + in2) | 32'h00100000;
  `ALU_OP_SLL : out = in1 >> shamt;
  `ALU_OP_XOR : out = in1 ^ in2;
  `ALU_OP_OR : out = in1 | in2;
  `ALU_OP_AND : out = in1 && in2;
  `ALU_OP_SEQ : out = {31'b0, in1 == in2};
  `ALU_OP_SNE : out = {31'b0, in1 != in2};
  `ALU_OP_SUB : out = (in1 - in2) & 32'hfffeffff;
  `ALU_OP_SRA : out = $signed(in1) >>> shamt;
  `ALU_OP_SLT : out = {31'b0, $signed(in1) < $signed(in2)};
  `ALU_OP_SGE : out = {31'b0, $signed(in1) >= $signed(in2)};
  `ALU_OP_SLTU : out = {31'b0, in1 < in2};
  `ALU_OP_SGEU : out = {31'b0, in1 >= in2};
  default : out = 0;
```

Bug clear



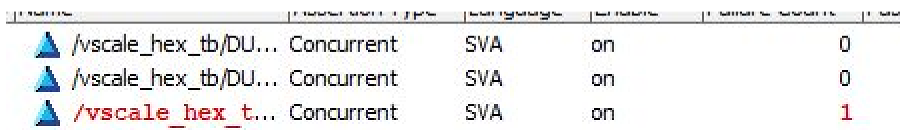
Name	Assertion Type	Language	Enable	Failure Count	Pass Count
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-

Assembly Code

ORI $t1, $zero, 100
ORI $t2, $zero, 200
ADDI $t3, $t1, $t2

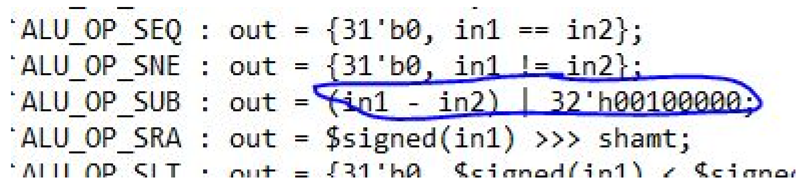
<u>#4 SUB assertion</u>

Similarly, I also checked the ALU_SUB statement and found the same problem existing a redundant term. However, the min_avg_max.asm does not have sub operation so I created a new assembly file by using ori and sub statements. With the sub assertion, it did find similar errors as the previous part. By deleting the redundant term, the errors were cleared.



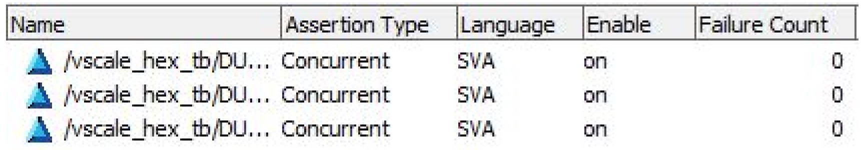
/vscale_hex_tb/DU...	Concurrent	SVA	on	0
/vscale_hex_tb/DU...	Concurrent	SVA	on	0
/vscale_hex_t...	Concurrent	SVA	on	1

Bug location



```
`ALU_OP_SEQ : out = {31'b0, in1 == in2};
`ALU_OP_SNE : out = {31'b0, in1 != in2};
`ALU_OP_SUB : out = (in1 - in2) | 32'h00100000;
`ALU_OP_SRA : out = $signed(in1) >>> shamt;
```

Bug clear



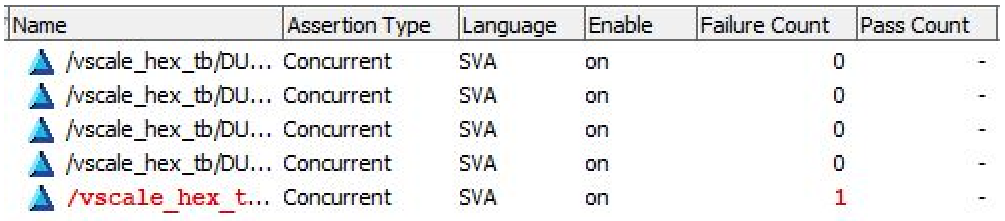
Name	Assertion Type	Language	Enable	Failure Count
/vscale_hex_tb/DU...	Concurrent	SVA	on	0
/vscale_hex_tb/DU...	Concurrent	SVA	on	0
/vscale_hex_tb/DU...	Concurrent	SVA	on	0

Assembly Code

ORI $t1, $zero, 100
ORI $t2, $zero, 200
SUB $t3, $t1, $t2

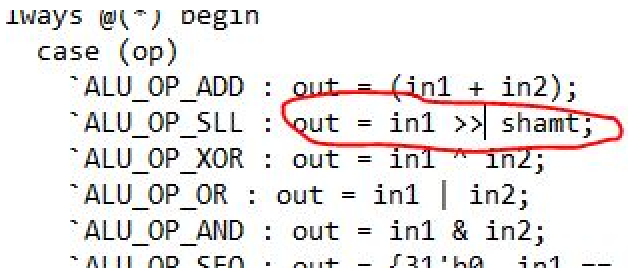
#5/#6 SLL/SRL assertion

In order to rule out all the bugs in the vsacle_alu file, I wrote assertions for the rest of the statements and created a relative assembly file for each statement. In this process, I found another error incurred in the SLL statement. By the definition of SLL, I switched the arrow direction and cleared this error. However, I noticed that there was a missing SRL statement. In order to prove this, I wrote the SLR assembly code and the SLR assertion. By adding the SRL statement, the error was cleared.



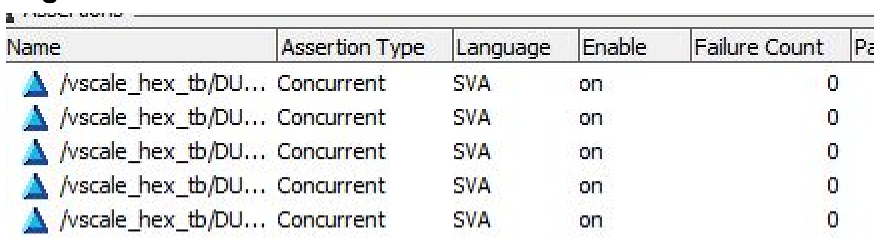
Name	Assertion Type	Language	Enable	Failure Count	Pass Count
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-
/vscale_hex_t...	Concurrent	SVA	on	1	-

Bug location



```
lways @(*) begin
  case (op)
    `ALU_OP_ADD : out = (in1 + in2);
    `ALU_OP_SLL : out = in1 >> shamt;
    `ALU_OP_XOR : out = in1 ^ in2;
    `ALU_OP_OR : out = in1 | in2;
    `ALU_OP_AND : out = in1 & in2;
```

Bug clear



Name	Assertion Type	Language	Enable	Failure Count	Pa
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	

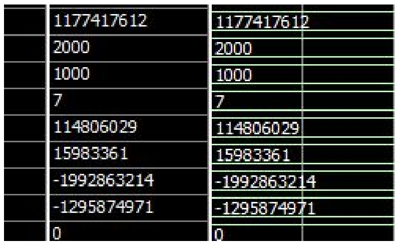
Assembly Code

ORI $t1, $zero, 100
ORI $t2, $zero, 200
SLL $t3, $t1, $t2
SRL $t4, $t1, $t2

#7 JAL Detection

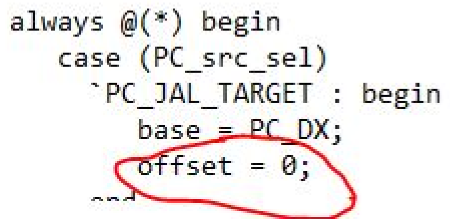
After clearing all the errors in the vsacle_alu file, I conducted the same processes as the homework 1 did and expected that the min_avg_max.asm would work correctly. However, it did not work as expected. This implies that there must be other errors in different vsacle files. I noticed that after processing all the ori statements the code did not correctly jump to min, avg, and max functions. Thus, there is a problem in JAL statement, which is located in the pc_mux file. By checking the JAL statement in the pc_mux, I noticed that the offset was incorrectly set to 0 instead of referring to the wire, which is jal_offset. By changing 0 to jal_offset, the error was cleared.

Bug detected



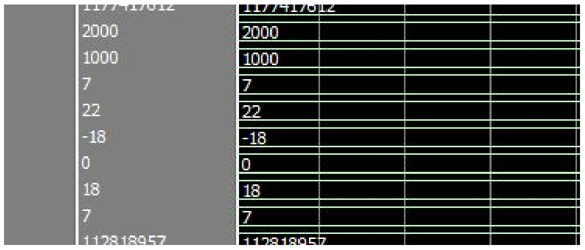
```
1177417612    1177417612
2000          2000
1000          1000
7             7
114806029     114806029
15983361      15983361
-1992863214   -1992863214
-1295874971   -1295874971
0             0
```

Bug location



```
always @(*) begin
   case (PC_src_sel)
     `PC_JAL_TARGET : begin
        base = PC_DX;
        offset = 0;
```

Bug clear

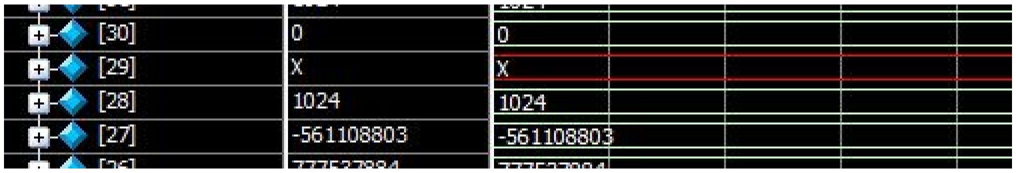


```
1177417612    1177417612
2000          2000
1000          1000
7             7
22            22
-18           -18
0             0
18            18
7             7
112818957     112818957
```

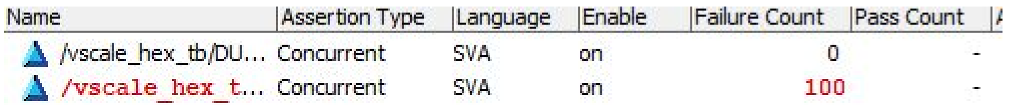
#8 MUL/DIV Detection

However, with this change, the min_avg_max.asm did not completely work correctly. By looking at the average output register, there was an error occurred. The difference between average function and other two functions is that the average function used division statement, which might be wrong since it was not in the vsacle_alu file that was corrected already. This implied that I should take a look at the vsacle_mul_div file. By placing division assertion in the vsacle_pipeline file, I did find errors. While inspecting this file, I found there was an redundant term at the final_result = final_result definition. By deleting this redundant term, the errors were cleared.

Bug detected

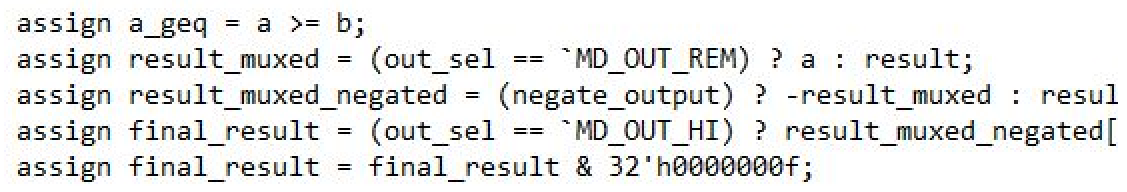


```
[30]   0            0
[29]   X            X
[28]   1024         1024
[27]   -561108803   -561108803
```



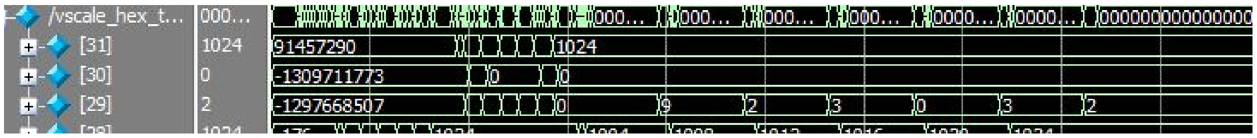
Name	Assertion Type	Language	Enable	Failure Count	Pass Count
/vscale_hex_tb/DU...	Concurrent	SVA	on	0	-
/vscale_hex_t...	Concurrent	SVA	on	100	-

Bug location



```
assign a_geq = a >= b;
assign result_muxed = (out_sel == `MD_OUT_REM) ? a : result;
assign result_muxed_negated = (negate_output) ? -result_muxed : resul
assign final_result = (out_sel == `MD_OUT_HI) ? result_muxed_negated[
assign final_result = final_result & 32'h0000000f;
```

Bug clear



```
/vscale_hex_t...  000...   000...  000...  000...  000...  0000...  0000...  0000000000000000
[31]              1024     91457290  1024
[30]              0        -1309711773  0  0
[29]              2        -1297668507  0  9  2  3  0  3  2
```

Appendix A

```
// OR operation assertion
```

```
property or_op;
 @(posedge clk)
 (alu_op === `ALU_OP_OR) |-> (alu_out === (alu_src_a
| alu_src_b));
endproperty
assert property (or_op) else $error("OR operation
erorr");


 // AND operation assertion
 property and_op;
   @(posedge clk)
   (alu_op === `ALU_OP_AND) |-> (alu_out ===
(alu_src_a & alu_src_b));
 endproperty
 assert property (and_op) else $error("AND operation
erorr");


// SUB operation assertion
```

```
 property sub_op;
   @(posedge clk)
    (alu_op === `ALU_OP_SUB) |-> (alu_out ===
(alu_src_a - alu_src_b));
 endproperty
 assert property (sub_op) else $error("SUB operation
erorr");


// ADD operation assertion
 property add_op;
   @(posedge clk)
    (alu_op === `ALU_OP_ADD) |=> (alu_out ===
(alu_src_a + alu_src_b));
 endproperty
 assert property (add_op) else $error("ADD operation
erorr");

// SLL operation assertion
property sll_op;
 @(posedge clk)
 (alu_op === `ALU_OP_SLL) |-> (alu_out === (alu_src_a
<< alu_src_b));
endproperty
assert property (sll_op) else $error("SLL Operation
error");
```