EE271 - Assignment 4 Write-up
Catching Difficult Bugs with A-QED

Khushal Sethi
Gongqi Li

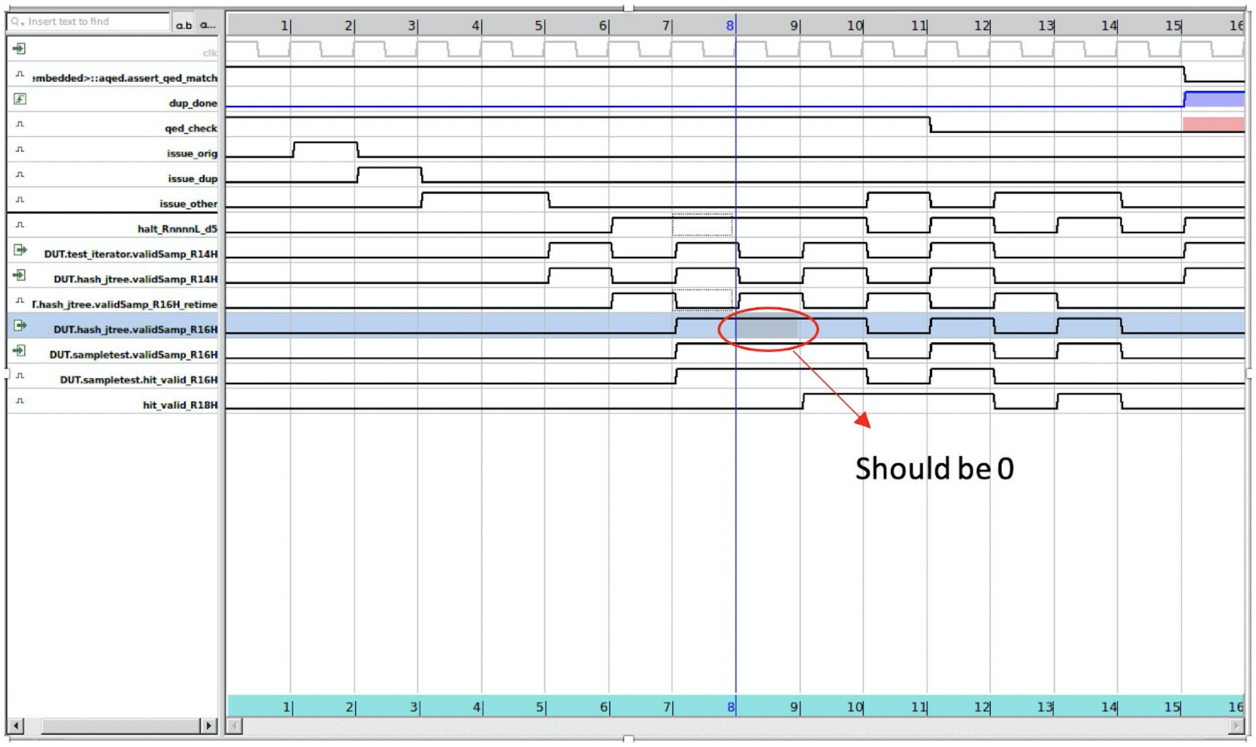
1 Introduction

In this part of the project, we build an A-QED module and connect it to our rasterizer to perform formal verification. The A-QED module basically sends two inputs, one original and one duplicate, to the rasterizer and monitor the output signal. To ensure that the code is bug-free, the necessary condition is that for the same inputs, the rasterizer should always give out the same outputs.

So we run the BMC together with our A-QED module on the CAD tool called Jasper, which can easily visualize the waveform of all the signals and help us trace back to find the origin of any bugs. In this project, **we found two bugs, one associated with the ‘tree_hash.sv’ module and the other with the ‘hash_jtree.sv’ module**.
Details are shown in section 2.

2 Catching Bugs

1. **Bug 1: ‘hit_valid_R18H’ violation of the Functional Consistency**


Insert text to find
clk
embedded>::aqed.assert_qed_match
dup_done
qed_check
issue_orig
issue_dup
issue_other
halt_RnnnnL_d5
DUT.test_iterator.validSamp_R14H
DUT.hash_jtree.validSamp_R14H
T.hash_jtree.validSamp_R16H_retime
DUT.hash_jtree.validSamp_R16H
DUT.sampletest.validSamp_R16H
DUT.sampletest.hit_valid_R16H
hit_valid_R18H
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Should be 0

- From the waveform above, we see that a bug arises at clk period 15.
- The first output subsequence, which corresponds to the original input, is at clk 10, while the second output subsequence, corresponding to the duplicate input, is at clk 12 -- matching the region where ‘halt_RnnnnL_d5’ is low.
  However, we also observe that the ‘**hit_valid_R18H’ is 0 at clk 12, meaning that there is no valid hit for the second subsequence, this is a direct violation of the Functional Consistency.**
  Then we check if the generation of ‘hit_valid_R18H’ contains any bugs.

- After tracing back following the route,
  ‘hit_valid_R18H’ -> ‘sampletest.hit_valid_R16H’ -> ‘sampletest.validSamp_R16H’ -> ‘hash_jree.validSamp_R16H’ -> ‘hash_jtree.validSamp_R16H_retime’ -> ‘hash_jtree.validSamp_R14H’, We found a mismatch between ‘validSamp_R14H’ and the translated ‘validSamp_R16H’ as shown in the waveform above.

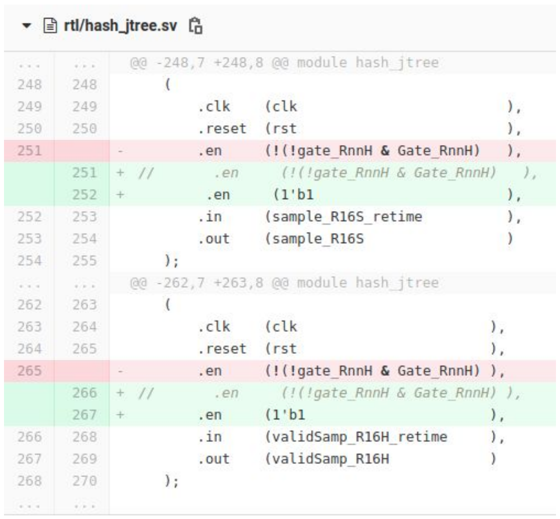
- A detailed investigation of the code showed that the error is within the module **‘hash_jtree.sv’**, where a **very suspicious signal ‘gate_RnnH’** being the input of the d-flip flop from ‘sample_R16S_retime’ to ‘sample_R16S’, and ‘validSamp_R16H_retime’ to ‘validSamp_R16H’.

- Since the enable **‘gate_RnnH’ is not always ‘1b1’, there are cases that ‘validSamp_R16H’ and ‘sample_R16S’ are not updated as required**, just as what is being **red-circled in the wave form above.**

- A-QED is able to find bugs in this while formal verification cannot, because A-QED uses BMC which is significantly more thorough.

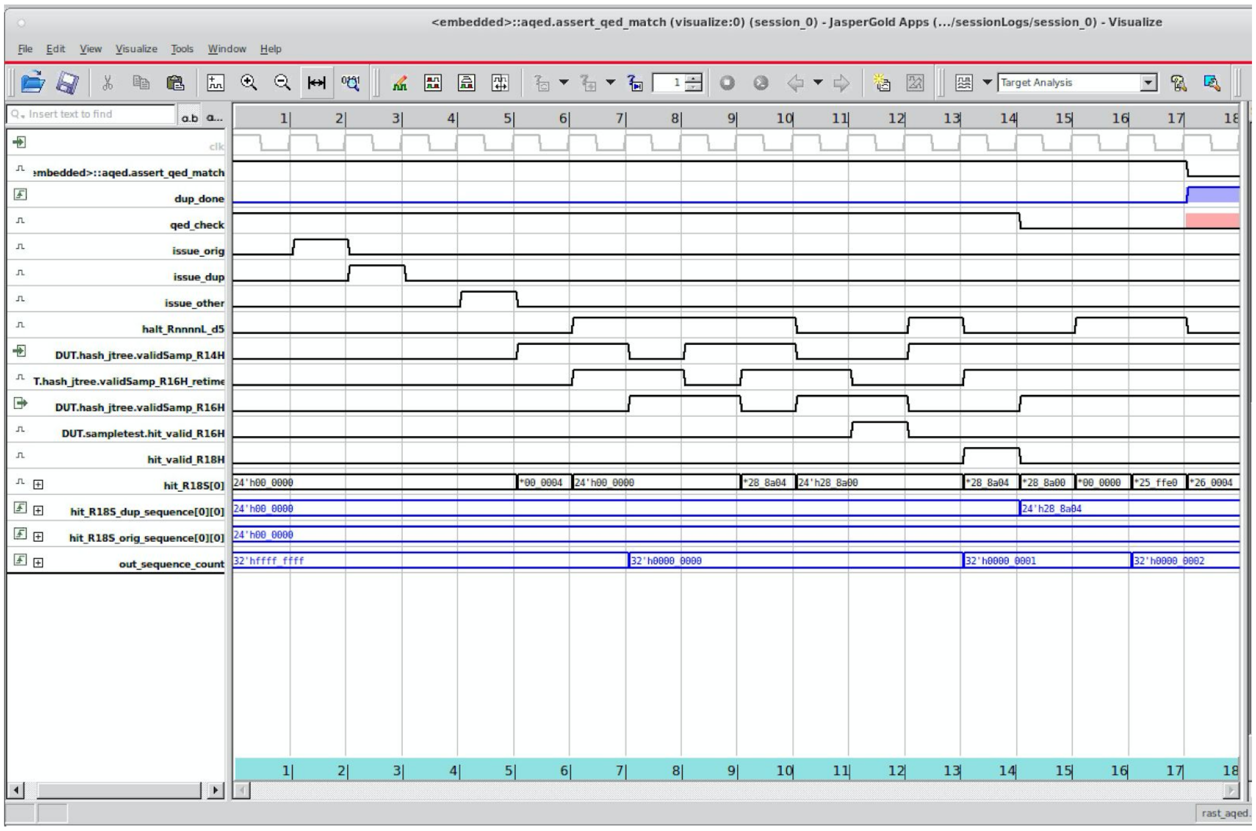
- A-QED found a short counterexample to this error, which was not found by running test vectors in verification earlier or in formal verification, **no formal verification properties were checking ‘validSamp_R16H_retime’ to ‘validSamp_R16H’** and **test vectors do not cover this case when inputs arrive concurrently under stress.**

Code fix :



```
rtl/hash_jtree.sv
...  ...  @@ -248,7 +248,8 @@ module hash_jtree
          (
              .clk    (clk                      ),
              .reset  (rst                      ),
-             .en     (!(!gate_RnnH & Gate_RnnH)   ),
+ //            .en     (!(!gate_RnnH & Gate_RnnH)   ),
+              .en     (1'b1                     ),
              .in     (sample_R16S_retime        ),
              .out    (sample_R16S               )
          );
...  ...  @@ -262,7 +263,8 @@ module hash_jtree
          (
              .clk    (clk                    ),
              .reset  (rst                    ),
-             .en     (!(!gate_RnnH & Gate_RnnH) ),
+ //            .en     (!(!gate_RnnH & Gate_RnnH) ),
+             .en     (1'b1                   ),
              .in     (validSamp_R16H_retime   ),
              .out    (validSamp_R16H          )
          );
...  ...
```

2. Bug 2: Extrinsic signals ‘Mask_RnnH’ and ‘mask_R1H’



<embedded>::aqed.assert_qed_match (visualize:0) (session_0) - JasperGold Apps (.../sessionLogs/session_0) - Visualize
File Edit View Visualize Tools Window Help
Target Analysis
clk
embedded>::aqed.assert_qed_match
dup_done
qed_check
issue_orig
issue_dup
issue_other
halt_RnnnnL_d5
DUT.hash_jtree.validSamp_R14H
T.hash_jtree.validSamp_R16H_retime
DUT.hash_jtree.validSamp_R16H
DUT.sampletest.hit_valid_R16H
hit_valid_R18H
hit_R18S[0]
hit_R18S_dup_sequence[0][0]
hit_R18S_orig_sequence[0][0]
out_sequence_count
24'h00_0000
*00_0004
*28_8a04
24'h28_8a00
*28_8a00
*00_0000
*25_ffe0
*26_0004
24'h28_8a04
32'hffff_ffff
32'h0000_0000
32'h0000_0001
32'h0000_0002

- After fixing the previous bug, we reran the BMC and did see that **the mismatch between ‘validSamp_R14H’ and ‘validSamp_R16H’ disappeared as the waveform shown above.** However, in this case, ‘hit_valid_R18H’ did not return a high for the first subsequence, while only returning high for the second subsequence.

- As a result, the ‘hit_R18S_orig_sequence’ is always 0. We then try to figure out why the first peak of ‘hit_valid_R18H’ disappeared.

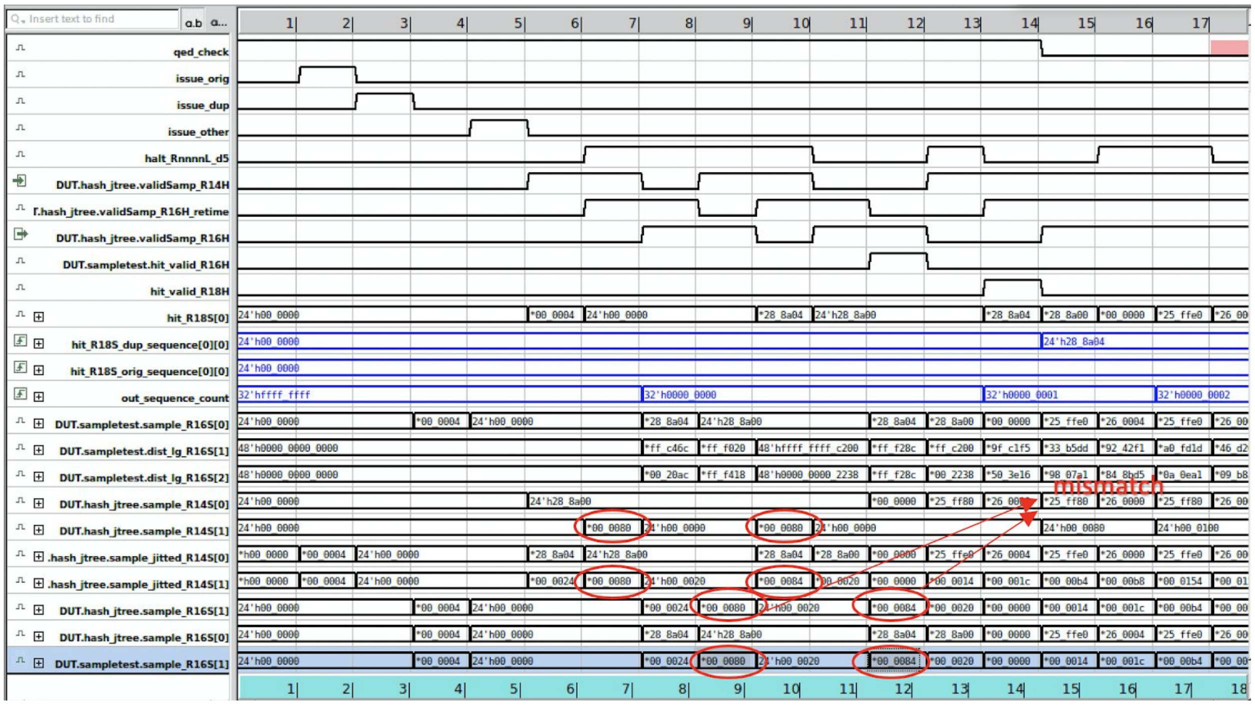
- **We trace back following the route:**
  ‘hit_valid_R18H’ -> ‘sampletest.hit_valid_R16H’ -> ‘sampletest.edge_chk1_R16’ -> ‘sampletest.dist_lg_R16S’ -> ‘sampletest.edge_R16S’ -> ‘sampletest.tri_shift_R16S’ -> ‘sampletest.sample_R16S’ -> ‘hash_jtree.sample_R16S’ ->hash_jtree.sample_R16S_retime’ -> ‘hash_jtree.sample_jitted_R14S’ -> ‘hash_jtree.jitt_val_R14H’ -> ‘tree_hash.out_RnnH’

- As illustrated in the waveform bellowed, a **mismatch arises traveling from ‘sample_R14S’ to ‘sample_jitted_R14S’.** The reason is that the module for the **jittering process does not give out the same output for the same input**.

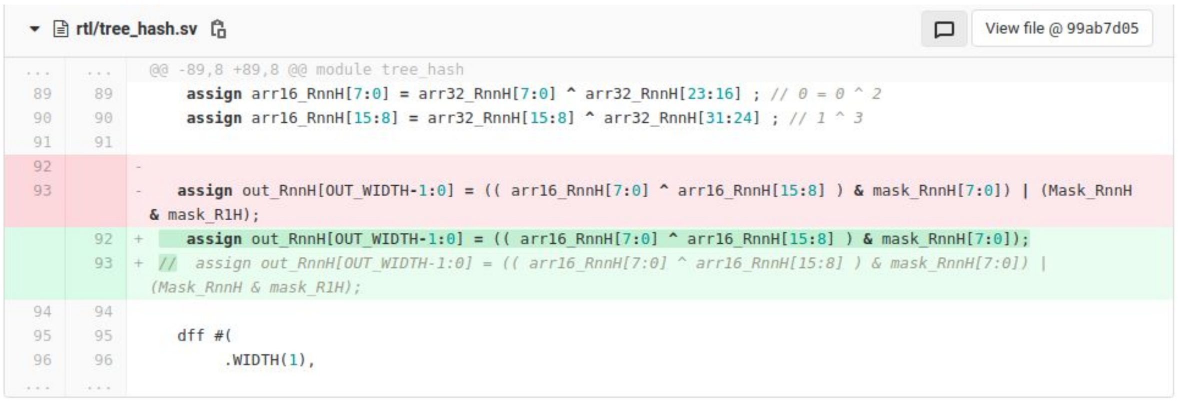
- Looking into the **‘tree_hash.sv’** module, we found that the **output is dependent on extrinsic signals ‘Mask_RnnH’ and ‘mask_R1H’. We thus removed these dependencies.**

- A-QED found a short counter example where **‘Mask_RnnH’ and ‘mask_R1H’ cause the output to change from ‘sample_R14S’ to ‘sample_jitted_R14S’. After some clock cycles ‘mask_R1H’ changes the output bits,** when both are issued concurrently.

- This bug was not found in formal verification or in test_vector verification because the properties of ‘sample_R14S’ to ‘sample_jitted_R14S’ are not formally checked under concurrent execution. The test vectors also don’t cover cases of scenarios where ‘Mask_RnnH’ and ‘mask_R1H’ combine to differ the output.


qed_check
issue_orig
issue_dup
issue_other
halt_RnnnnL_d5
DUT.hash_jtree.validSamp_R14H
DUT.hash_jtree.validSamp_R16H_retime
DUT.hash_jtree.validSamp_R16H
DUT.sampletest.hit_valid_R16H
hit_valid_R18H
hit_R18S[0]
hit_R18S_dup_sequence[0][0]
hit_R18S_orig_sequence[0][0]
out_sequence_count
DUT.sampletest.sample_R16S[0]
DUT.sampletest.dist_lg_R16S[1]
DUT.sampletest.dist_lg_R16S[2]
DUT.hash_jtree.sample_R14S[0]
DUT.hash_jtree.sample_R14S[1]
.hash_jtree.sample_jitted_R14S[0]
.hash_jtree.sample_jitted_R14S[1]
DUT.hash_jtree.sample_R16S[1]
DUT.hash_jtree.sample_R16S[0]
DUT.sampletest.sample_R16S[1]
mismatch

Code fix :



rtl/tree_hash.sv

View file @ 99ab7d05

```
@@ -89,8 +89,8 @@ module tree_hash
     assign arr16_RnnH[7:0] = arr32_RnnH[7:0] ^ arr32_RnnH[23:16] ; // 0 = 0 ^ 2
     assign arr16_RnnH[15:8] = arr32_RnnH[15:8] ^ arr32_RnnH[31:24] ; // 1 ^ 3

-
-    assign out_RnnH[OUT_WIDTH-1:0] = (( arr16_RnnH[7:0] ^ arr16_RnnH[15:8] ) & mask_RnnH[7:0]) | (Mask_RnnH & mask_R1H);
+    assign out_RnnH[OUT_WIDTH-1:0] = (( arr16_RnnH[7:0] ^ arr16_RnnH[15:8] ) & mask_RnnH[7:0]);
+ //  assign out_RnnH[OUT_WIDTH-1:0] = (( arr16_RnnH[7:0] ^ arr16_RnnH[15:8] ) & mask_RnnH[7:0]) | (Mask_RnnH & mask_R1H);

    dff #(
        .WIDTH(1),
```