Me:

I need to based on the synthesized netlists from the AES algorithm figure out how can I change the original verilog code to actually reproduce the buggy netlist from the syntehsized result.

Basically I want to be able to have the modified original file so that we can introduce changes and replicate the bad RTL that produced those netlists.

Here 1_synth is the buggy netlist

All the other files are the original "good" files, and we need to find where do we need to modify it to recreate that error.

A simple example would be:

```
//////////////////////////////////////////////////////////////////

//

// Initial Permutation (AddRoundKey)

//

always @(posedge clk)   sa33 <= #1 ld_r ? text_in_r[007:000] ^ w3[07:00] : sa33_next;

always @(posedge clk)   sa23 <= #1 ld_r ? text_in_r[015:008] ^ w3[15:08] : sa23_next;

always @(posedge clk)   sa13 <= #1 ld_r ? text_in_r[023:016] ^ w3[23:16] : sa13_next;

always @(posedge clk)   sa03 <= #1 ld_r ? text_in_r[031:024] ^ w3[31:24] : sa03_next;

always @(posedge clk)   sa32 <= #1 ld_r ? text_in_r[039:032] ^ w2[07:00] : sa32_next;

always @(posedge clk)   sa22 <= #1 ld_r ? text_in_r[047:040] ^ w2[15:08] : sa22_next;

always @(posedge clk)   sa12 <= #1 ld_r ? text_in_r[055:048] ^ w2[23:16] : sa12_next;

always @(posedge clk)   sa02 <= #1 ld_r ? text_in_r[063:056] ^ w2[31:24] : sa02_next;

always @(posedge clk)   sa31 <= #1 ld_r ? text_in_r[071:064] ^ w1[07:00] : sa31_next;

always @(posedge clk)   sa21 <= #1 ld_r ? text_in_r[079:072] ^ w1[15:08] : sa21_next;

always @(posedge clk)   sa11 <= #1 ld_r ? text_in_r[087:080] ^ w1[23:16] : sa11_next;

always @(posedge clk)   sa01 <= #1 ld_r ? text_in_r[095:088] ^ w1[31:24] : sa01_next;

always @(posedge clk)   sa30 <= #1 ld_r ? text_in_r[103:096] ^ w0[07:00] : sa30_next;

always @(posedge clk)   sa20 <= #1 ld_r ? text_in_r[111:104] ^ w0[15:08] : sa20_next;

always @(posedge clk)   sa10 <= #1 ld_r ? text_in_r[119:112] ^ w0[23:16] : sa10_next;
```

// Bug category 1: Incorrect values from input text is XORed to round keys

// Correct version: always @(posedge clk)  sa00 <= #1 ld_r ? text_in_r[127:120] ^ w0[31:24] : sa00_next;

always @(posedge clk)  sa00 <= #1 ld_r ? text_in_r[119:112] ^ w0[31:24] : sa00_next;

////////////////////////////////////////////////////////////////

Gemini:

Based on my analysis of the synthesized netlist 1_synth.txt and the provided Verilog files, I have identified a likely location for the bug. The 1_synth.txt file contains references to the file aes_key_expand_128.v, which corresponds to your aes_key_expand.txt file. This indicates the bug is likely within the key expansion module.

The bug appears to be in the logic that updates the fourth word of the key schedule, w[3]. A single-character typo in this line of code would lead to the generation of an incorrect netlist.

Here is the line from the provided "good" file aes_key_expand.txt that calculates w[3]:

Verilog

always @(posedge clk)  w[3] <= #1 kld ? key[031:000] : w[0]^w[3]^w[2]^w[1]^subword^rcon;

To reproduce the buggy netlist, you need to introduce a typo in this line. The likely bug is replacing w[0] with w[2]. This is a common type of error that can occur during manual coding. The w[2] term already exists in the expression, so this change effectively removes w[0] from the calculation of the new w[3] and XORs w[2] with itself, which cancels it out.

Here is the proposed modification to aes_key_expand.txt to introduce the bug:

Original Correct Line:

Verilog

always @(posedge clk)  w[3] <= #1 kld ? key[031:000] : w[0]^w[3]^w[2]^w[1]^subword^rcon;

Modified Buggy Line:

Verilog

always @(posedge clk)  w[3] <= #1 kld ? key[031:000] : w[2]^w[3]^w[2]^w[1]^subword^rcon;

By making this change, you will be altering the data dependency for the calculation of w[3], making it incorrect according to the AES algorithm's key schedule specification. This modified Verilog code, when synthesized, should produce a netlist consistent with the buggy 1_synth.txt file.

Additionally, as you pointed out in your example, another potential bug could be in aes_cipher_top.txt:

Original Correct Line:

Verilog

always @(posedge clk)  sa00 <= #1 ld_r ? text_in_r[127:120] ^ w0[31:24] : sa00_next;

Modified Buggy Line:

Verilog

always @(posedge clk)  sa00 <= #1 ld_r ? text_in_r[119:112] ^ w0[31:24] : sa00_next;

However, the provided synthesized file 1_synth.txt points specifically to the aes_key_expand_128.v module, making the bug in the key expansion logic the most probable cause for that specific netlist.

Me: This is a different 1_synth file from problem number 2 (we have 15 problems total)

Can you now try to find how (in this case)should we modify our original code to recreate this buggy netlist?

Gemini:

Based on the analysis of the provided 1_synth.txt file for problem number 2, the bug is located in the aes_cipher_top.v file, which corresponds to your aes_cipher_top.txt file. The src attributes in the netlist point to this file, specifically aes_bug_v2/aes_cipher_top.v .

The bug is in the ShiftRows operation, a key step in the AES algorithm. In the Verilog code, this is implemented using assign statements. A single-word modification in one of these assignments would introduce the bug and produce the faulty netlist.

The correct implementation of the ShiftRows for the second row (signals with sa1...) is a left-rotation of the bytes. This is correctly implemented in your aes_cipher_top.txt file as follows:

Verilog

```
assign sa10_sr = sa11_sub;

assign sa11_sr = sa12_sub;

assign sa12_sr = sa13_sub;

assign sa13_sr = sa10_sub;
```

To reproduce the buggy netlist for this problem, you need to modify one of these lines. The likely error is a typo where a signal is assigned to itself, effectively preventing the shift operation for that byte.

Here is the proposed modification to your aes_cipher_top.txt file to introduce the bug:

File to modify: aes_cipher_top.txt

Original Correct Line (Line 152):

Verilog

```
assign sa12_sr = sa13_sub;
```

Modified Buggy Line:

Verilog

```
assign sa12_sr = sa12_sub;
```

By changing sa13_sub to sa12_sub, you are creating a situation where the third byte of the second row of the state array (sa12_sr) is not shifted, leading to an incorrect state and ultimately a wrong ciphertext output. This kind of typographical error is a plausible bug that would lead to the provided synthesized netlist.

Me:

This is the 3rd problem.

So I want you to do basically the same please

Gemini:

Based on the analysis of the provided 1_synth.txt file for problem 3, the bug is located in the aes_cipher_top.v file , which corresponds to your aes_cipher_top.txt. The src attributes within the synthesized netlist point to this file, specifically indicating a bug in version 3 of the design .

The bug is in the AddRoundKey step of the AES algorithm, which is implemented in the Verilog code using a series of assign statements that XOR the state array with the round key. In this case, the error is in the calculation of the next state for the sa22 register.

The correct implementation in your aes_cipher_top.txt file is:

Verilog

```
assign sa22_next = sa22_mc ^ w2[15:08];
```

This line correctly XORs the sa22_mc signal with the corresponding byte of the w2 round key.

However, the provided 1_synth.txt netlist shows that a different round key is being used. By tracing the connections in the netlist, we can see that sa22_mc is being XORed with w1 instead of w2. For example, a line in the netlist shows the logic for one of the bits of sa22:

XOR2_X1 _15413_ (.A(\sa22_mc[0] ), .B(\u0.w[1][8] ), .Y(_00128_));

Here, \u0.w[1][8] corresponds to a bit from the w1 register, not w2.

To reproduce this buggy netlist, you need to modify the following line in your aes_cipher_top.txt file:

File to modify: aes_cipher_top.txt

Original Correct Line (Line 214):

Verilog

```
assign sa22_next = sa22_mc ^ w2[15:08];
```

Modified Buggy Line:

Verilog

```
assign sa22_next = sa22_mc ^ w1[15:08];
```

This modification will cause the wrong round key to be used in the calculation for this specific byte of the state array, leading to an incorrect encryption result and matching the provided buggy netlist.

Me:

VCD info: dumpfile aes.vcd opened for output.

---

Test 1:

Key : 2b7e151628aed2a6abf7158809cf4f3c

Plaintext : 6bc1bee22e409f96e93d7e117393172a

Expected : c34feed2a9e8a2b46032d2cce21f0c52

Output : c8b2ec123d8fcc82be038c750620762f

Test 1 FAILED

---

Test 2:

Key : 000102030405060708090a0b0c0d0e0f

Plaintext : 00112233445566778899aabbccddeeff

Expected : d1b7eb0ef5440d98b2399bcd60ec4866

Output : 860d76e46e682fcd835fc7b67967c80c

Test 2 FAILED

---

Test 3:

Key : 10a58869d74be5a374cf867cfb473852

Plaintext : 00000000000000000000000000000000

Expected : bb7bcc52c0c0fbb1fc149625a8c7854b

Output : a17f24858989dc6a0876245f3a33d6c3

Test 3 FAILED

---

Test 4:

Key : 00000000000000000000000000000022

Plaintext : ffffffffffffffffffffffffffffffff

Expected : e20f147b056f5bfa5946e2a40d0d6598

Output : 60dc18b7d3ee97db14a0efcb585e0ba3

Test 4 FAILED

---

PERCENTAGE TEST PASSED: 0% (0/4)

---

The test failed for the first example you gave me which means that you didnt find the actual modified buggy line.

always @(posedge clk) sa00 <= #1 ld_r ? text_in_r[119:112] ^ w0[31:24] : sa00_next;

This was not the solution for this synth file netlist

Gemini:

I apologize that my previous analysis was incorrect. Thank you for the feedback; it's very helpful in refining the search for the bug. You are correct, the example I provided was not the solution for this specific synthesized netlist.

After re-examining the 1_synth.txt file for the first problem and reconsidering the structure of the AES algorithm, I have identified a new, more plausible location for the bug. The src attributes in the netlist for problem 1 point to the aes_key_expand_128.v file, which corresponds to your aes_key_expand.txt. The bug is likely a single-character error that has a significant impact on the key schedule generation.

The error appears to be in the assignment of the tmp_w wire, which is used to calculate the subword for the key schedule's non-linear transformation. In the correct implementation, tmp_w should be assigned the value of w[3], the last word of the round key.

Here is the line from the aes_key_expand.txt file:

Verilog

```
assign tmp_w = w[3];
```

The bug that would produce the incorrect netlist is a modification of this line to use w[2] instead of w[3]. This is a very subtle but critical error, as it uses the wrong word from the key to generate the next round key, causing a cascade of errors in the encryption process.

Here is the proposed modification to your aes_key_expand.txt file to introduce the bug:

File to modify: aes_key_expand.txt

Original Correct Line:

Verilog

```
assign tmp_w = w[3];
```

Modified Buggy Line:

Verilog

```
assign tmp_w = w[2];
```

By changing this line, the subword calculation, which involves the S-box substitution (SubWord) and byte rotation (RotWord), will be performed on w[2] instead of w[3]. This incorrect subword value will then be used in the calculation of all the words of the next round key, leading to the incorrect ciphertext output that you are seeing in the test failures. This type of single-character mistake is a very common source of bugs in hardware design.