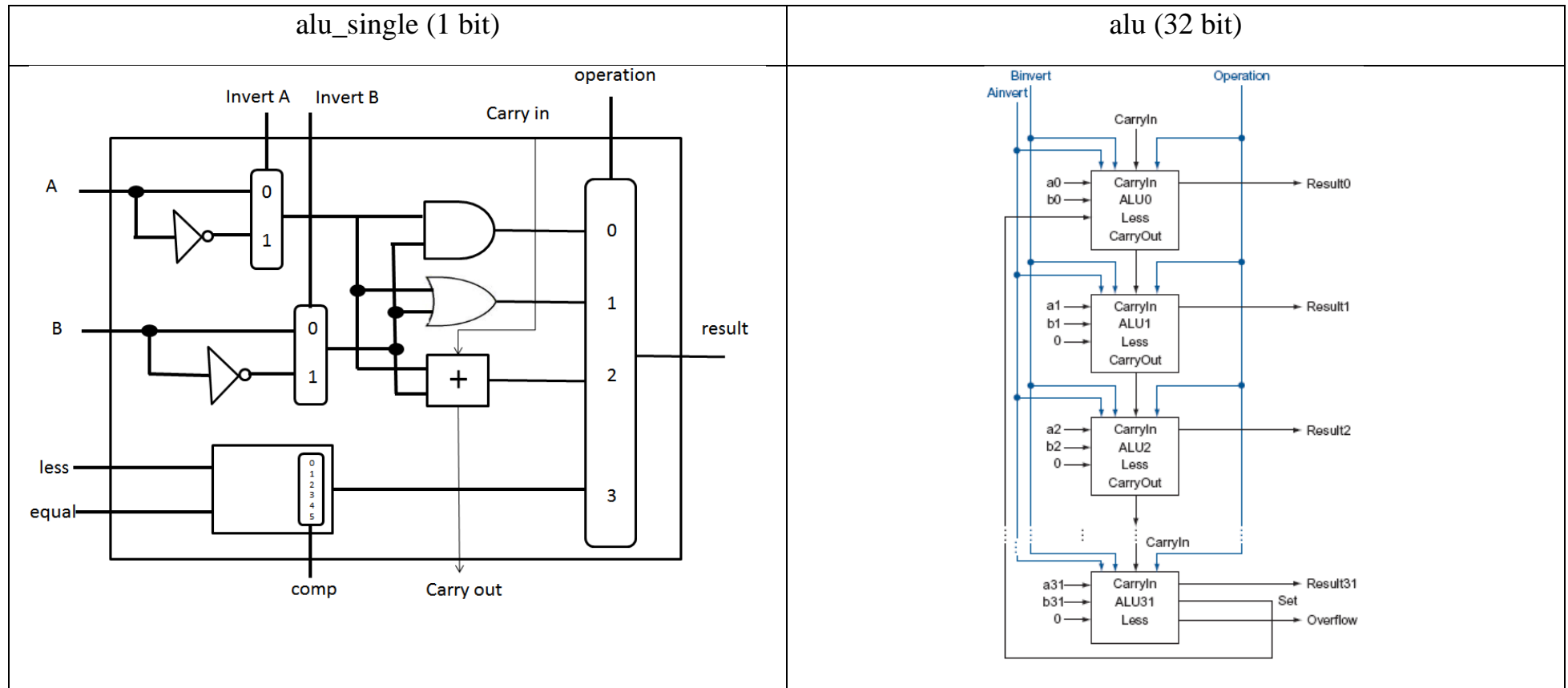


Computer Organization Lab 1

0113110 Po-han Chen, 0316213 Yu-wen Pwu

Architecture Diagram

We've implemented the ALU accordingly. (ie. We haven't changed the architectures specified in the HW document.)



For the final result, please refer to the appendix at the end of the report.

Detailed Description of the Implementation

We have designed a 32-bit ALU that supports common arithmetic operations including: And, Or, Add, Subtraction, Nor, Nand and Comparisons (Set less than, Set greater than, Set less equal, Set greater equal, Set equal, Set not equal).

The design is composed of 32 1-bit ALUs interconnected sequentially. Let's denote the ALU for the LSB (least significant bit) by ALU_0 , the ALU for the second LSB by ALU_1 and all the way to the ALU for the MSB (most significant bit) by ALU_{31} .

- For ALU_i , where $1 \leq i \leq 31$:
 - (Input) *src1* and *src2*, the 2 single-bit input numbers, are connected to *src1[i]* and *src2[i]* respectively.
 - (Input) *less*, the input for the comparison signals, is connected to 0 (constant).
 - (Input) *src1_invert* and *src2_invert*, indicating whether the input bit should be flipped, are connected to *src1_invert* and *src2_invert* respectively.
 - (Input) *cin*, the carry-in for this ALU_i , is connected to *cin[i]*.
 - (Input) *operation*, indicating the type of arithmetic operation to be carried out, is connected to *operation*.
 - (Output) *result* is the calculated result of ALU_i .
 - (Output) *cout*, the carry-out for bit *i*, connects with the carry-in for the next bit (ie. *cin[i + 1]*).
- For ALU_0 , everything is the same except that:
 - (Input) *less*, the input for the comparison signals, is connected to *less*.
- Regarding the wire connections outside the 1-bit ALUs, we considered them reversely during implementation:

- *cin[0]*, the carry-in for the LSB, is:
 - 1, if *ALU_control* is 4'b0110 (Subtract) or 4'b0111 (Compare)
 - 0, otherwise
- *src1_invert*, indicating whether *src1* should be inverted, is:
 - 1, if *ALU_control* is 4'b1100 (Nor) or 4'b1101 (Nand)
 - 0, otherwise
- *src2_invert*, indicating whether *src2* should be inverted, is:
 - 1, if *ALU_control* is 4'b1100 (Nor), 4'b1101 (Nand), 4'b0110 (Subtract) or 4'b0111 (Compare)
 - 0, otherwise
- *operation*, indicating the type of calculation for the 1-bit ALUs is:
 - 2'b00, if *ALU_control* is 4'b0000 (And) or 4'b1100 (Nor)
 - 2'b01, if *ALU_control* is 4'b0001 (Or) or 4'b1101 (Nand)
 - 2'b10, if *ALU_control* is 4'b0010 (Add) or 4'b0110 (Subtract)
 - 2'b11, otherwise
- On the higher-level considerations, or mathematical foundations, we have:
 - $src1 \bar{\vee} src2 = \neg src1 \wedge \neg src2$
 - $src1 \bar{\wedge} src2 = \neg src1 \vee \neg src2$
 - *real_less* (whether $src1 < src2$)
 - $= (src1 - src2 < 0)$
 - = bit 31 of *result* is 1 or negative integer overflow
 - $= (src1_{31} \oplus src2_{31} \oplus cin[31]) \vee (src1_{31} \wedge src2_{31} \wedge \neg cin[31])$
 - $equal = (src1 == src2)$

- *less*
case (bonus_control)
 - 3'b000 (Set less than): $less = (src1 < src2) = real_less$
 - 3'b001 (Set greater than): $less = \neg (src1 < src2 \text{ or } src1 == src2) = \neg (real_less \vee equal)$
 - 3'b010 (Set less equal): $less = (src1 < src2 \text{ or } src1 == src2) = (real_less \vee equal)$
 - 3'b011 (Set greater equal): $less = \neg (src1 < src2) = \neg real_less$
 - 3'b110 (Set equal): $less = (src1 == src2) = equal$
 - 3'b100 (Set not equal): $less = (src1 \neq src2) = \neg equal$
 - default: $less = real_less$
- *zero* = (*result* == 32'b0)
- *cout*
 = *cin*[32], if *operation* is 4'b0010 (Add) or 4'b0110 (Subtract)
 = 0, otherwise.
- *overflow*
 = (*cout*[31] \oplus *cout*[30]) = (*cin*[32] \oplus *cin*[31])
 = 0, otherwise

Problems Encountered and Solutions

We haven't encountered any problems during the implementation.


We have detailed our solutions or approaches above and within the comments of our code.

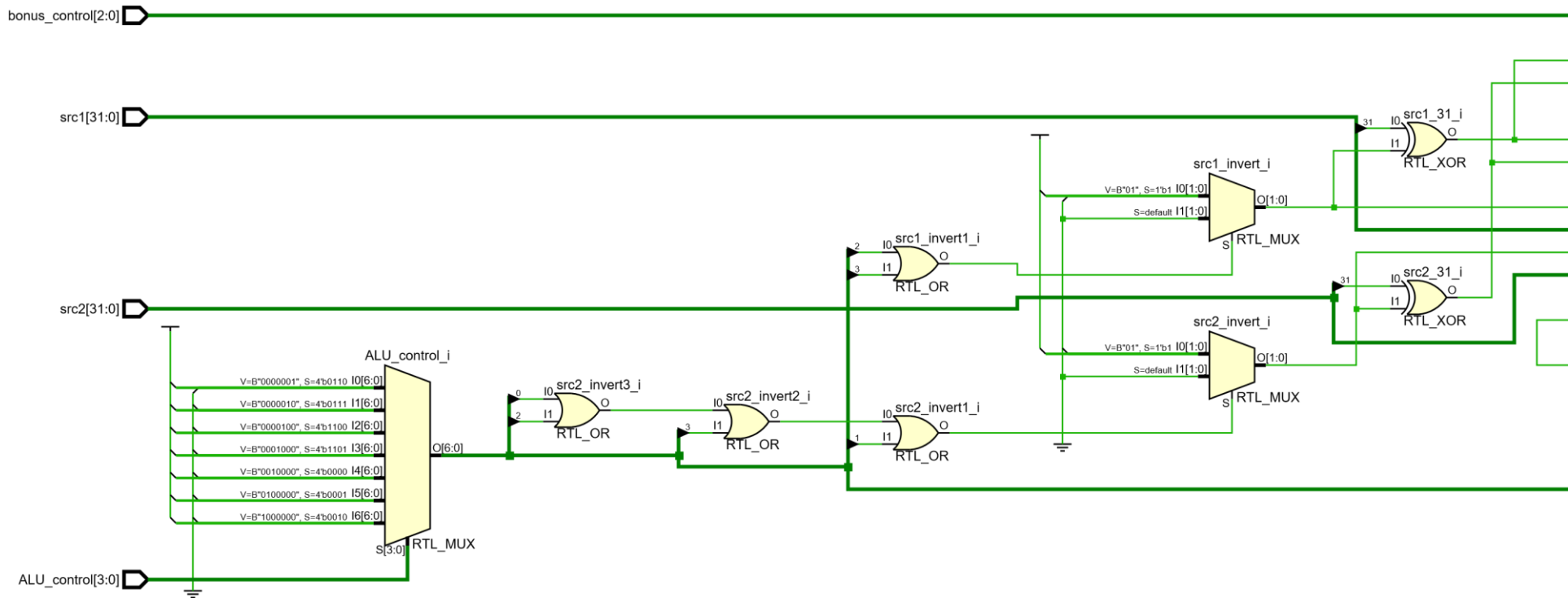
Lesson Learned (If Any)

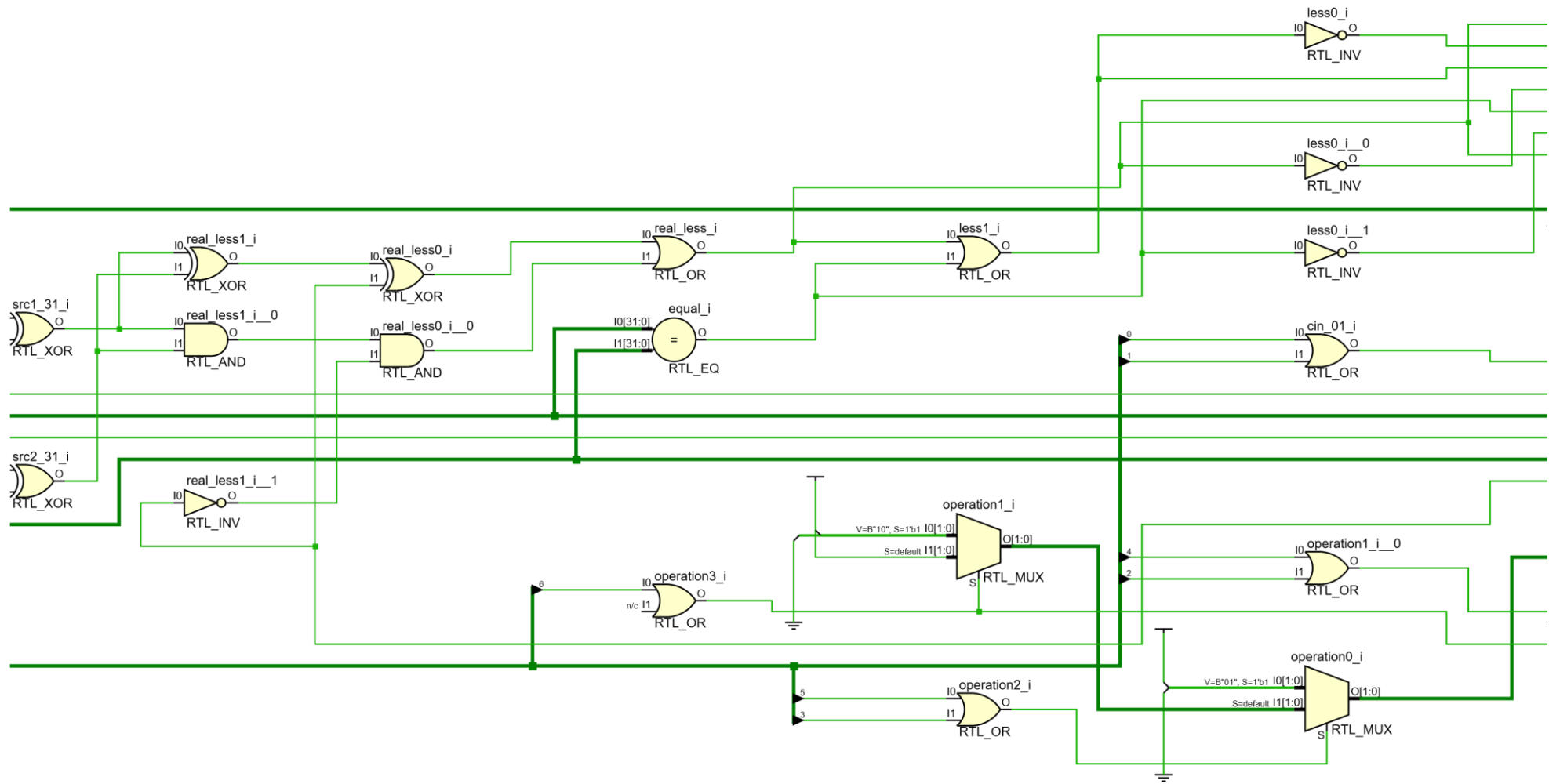
- I. 肝毋好，人生係黑白的；
 肝若好，作業係空白的。

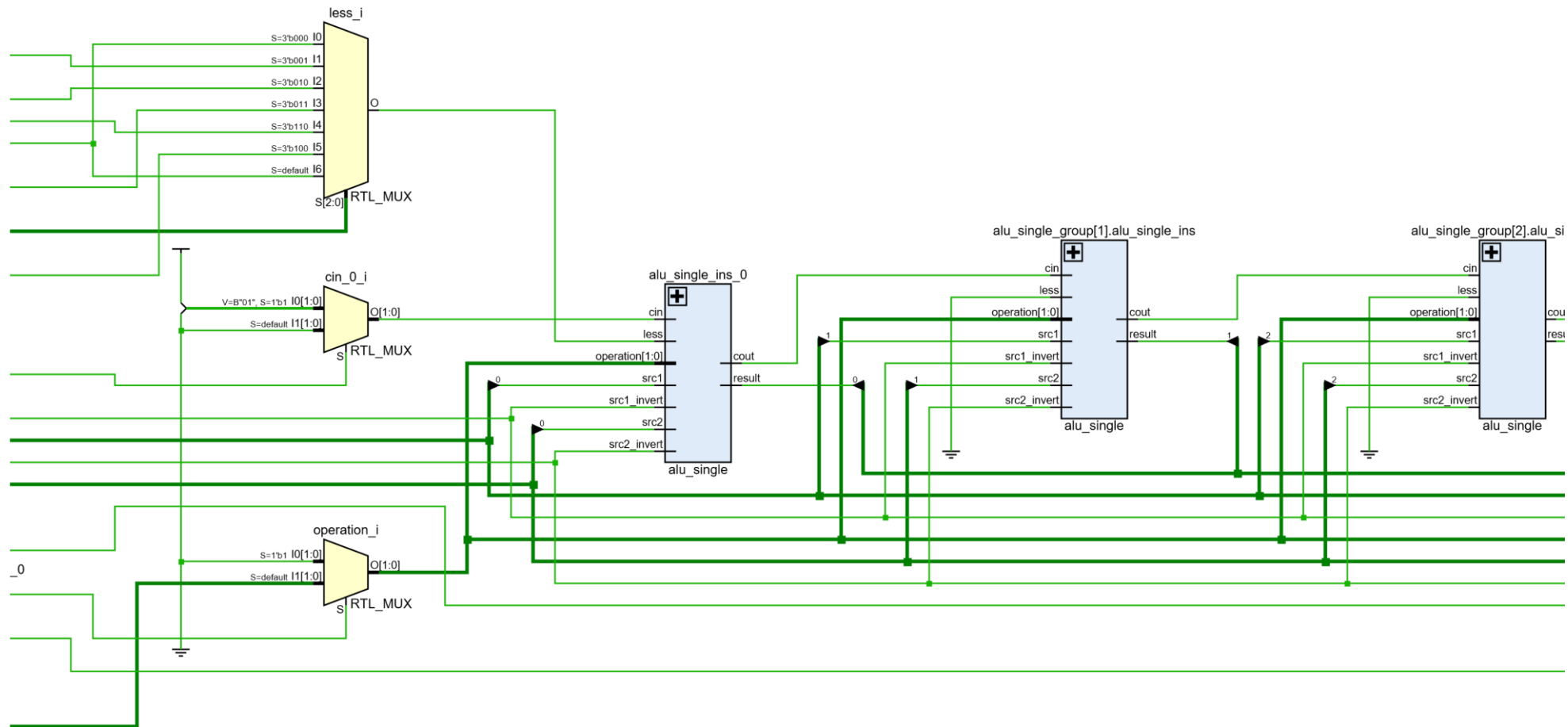
II. Verilog 2001 added the `generate` block, which is very handy to instantiate an abundance of modules.

III. Xilinx Vivado Design Suite is much more powerful than its legacy EDA tool – Xilinx ISE Design Suite.

rst_n 







X

