Report 01

Filippo Nevi - VR458510

October 2020

1 Div module

1.1 div_RTL.hh

The signal *clk* is not declared, it has to be added in the signals declarations by writing `sc_in<bool> clk;`

Also, the sensitivity list of the thread isn't correct: there is a compilation error because the line isn't complete; to fix this, we have to specify that the thread has to be sensitive to the rising edge of the clock signal by writing `sensitive << clk.pos();`

1.2 div_RTL.cc

The argument of *div_proc* doesn't match the signature in the header file, so we have to remove the argument `int number`.

There is a missing semicolon after `b = tmp_value.range(23,16)`: this generates a compilation error. Another similar error is generated because, in an operation, `R`, `G` and `B` are written in upper-case but the variables are declared with lower-case letters, so we simply have to rewrite these three names using lower case.

The variable `tmp_value` is not declared, and its initialization line is wrong since these is no value after the equals and there's not a semicolon at the end of the line. We can fix these problems by adding the line `sc_uint<32> tmp_result;` and assigning the value 0 in the first *if* statement: `tmp_value = 0;`

When reading the value of the signal *rst* in the *while* condition, we have to compare it to the value `sc_logic('1')` in order to achieve the correct behaviour of the system, although the compiler will not generate an error if we leave the value 1. A similar error is done in `while(start.read() != sc_logic(1))`, so we need to add single quote marks (') around 1.

In order to synchronize this thread with the testbench we need to write the logic value 1 on the *done* signal, otherwise the testbench thread will forever wait for this thread to finish its job: we need to add `done.write(sc_logic('1'));` after the *result.write* line.

1.3 div_RTL_testbench.cc

There is no sensitivity list in the constructor, so we need to write one: `sensitive << clk.pos();`. This synchronizes the two threads so they are able to communicate through their signals.

In order for the *div* thread to exit from its waiting loop, the *testbench* thread has to write the logic value 1 in the *start* signal, so we have to write `start.write(sc_logic('1'));`

2 Root module

2.1 main_root_RTL.cc

The signal *reset_signal* is not bound to the testbench, so we need to do this operation by writing `i_src_RTL.reset_to_RTL(reset_signal);` in the main file.

2.2 root_RTL_testbench.cc

In the function *run*, inside the *for*, the process needs to send the value to be calculated. We can do this by writing the line `p_Out_data.write(temp_data_in);`

2.3 root_RTL.cc

In the *elaborate_SQRT_FSM* method, the state **ST_1** has to store the value read from the signal *number_port*, but the original code stores the constant 0. This can be fixed by changing the code into `Number.write(number_port.read());`

The final issue is in the state **ST_4**, where we have to write the root result on the *result_port*. Since the value stored in the signal *Root* has to be right-shifted, we perform both operations in one line: `result_port.write(Root.read() >> 1);`