Vivado HLS: Debug Guide for investigating C/RTL co-simulation issues

This answer record #61063 explains how to analyze an issue reported by the Vivado HLS C/RTL co-simulation feature.

The Vivado HLS C/RTL co-simulation feature uses the user C test bench and generated RTL to confirm the RTL simulation matches the behavior of the C, C++ or SystemC source code.
When the C/RTL co-simulation fails, the following message is issued.

```
@E [SIM-4] *** C/RTL co-simulation finished: FAIL ***
```

If C/RTL co-simulation reports a failure, there are 4 primary reasons why this can occur:

1. Incorrect environment setup.
2. Optimization directives have been incorrectly applied or are unsupported for co-simulation.
3. There are issues with the C test bench or C source code.
4. There is an error in the synthesized RTL netlist or in the co-simulation feature.

Below are the steps to resolve a failure when using C/RTL co-simulation.

Environment Setup

A. Are you are using a 3rd-party simulator or the ISE RTL simulator for the RTL simulation?

   - The path to the simulator executable must be specified in the system search path.
   - There is no requirement to specify the search path when using the Xilinx RTL simulator (Xsim) or SystemC as the RTL simulator.
     Selecting either of these options will help confirm there is no issue with the RTL simulation.

B. If the log file shows the RTL simulator successfully starting and executing, proceed to verify the remaining items in this answer record.

C. If running on Linux, ensure that your resource files (for ecample .cshrc or .bashrc) do not have a change directory command. When c-sim or cosim starts, it spawns a new shell process.
If there is a cd command in your start up script it will cause the shell to run in a different location and eventually fail cosim.

Optimization Directives

A. Are you using the DEPENDENCE directive?

   - If you remove the DEPENDENCE directives from the design and then co-simulation passes, it is a likely indication the directive TRUE or FALSE settings are incorrect.

B. Does the design use volatile pointers on the interface?

   - When volatile pointers are used on the interface, the co-simulation feature needs to know how many accesses (read/writes) are performed on the port in each transaction (each execution of the C function).
   - This is specified using the DEPTH option on the INTERFACE directive.

C. Are you using the DATAFLOW optimization and FIFO channels?

- The default operation of the DATAFLOW optimization uses ping-pong memory buffers to implement the data flow channels. This is very safe but may introduce memories which are unnecessarily large. In streaming designs it is typical to use the CONFIG_DATAFLOW command to change the channels to use FIFOs and explicitly specify the size of the channel. If this size is set too small the RTL simulation may stall.

- Set the size of the FIFOs to the maximum size of the arrays in the design.
  If co-simulation passes, reduce the size as long as co-simulation can still pass successfully.

D. Are you using supported interfaces?

- Not all interface modes are supported for co-simulation. Refer to the Vivado HLS User Guide (UG902) section " Interface Synthesis Requirements" - page 232 in version 2014.2

E. Are you applying multiple optimizations directives to arrays on the interface?

- Some optimizations are not supported when used in conjunction with others.
  Refer to the Vivado HLS User Guide (UG902) section "Unsupported Optimizations"
  page 233 in version 2014.2

C Source and Test Bench Issues.

A. Does the C test bench check the results and return the value 0 (zero) if the results are correct?

- The C simulation must return the value 0 for co-simulation to pass.
  Even if the results are correct, the co-simulation feature will report a failure if the C test bench fails to return the value 0.

- For failures the return value must be in the range 1-255.

B. Is the test bench creating input data based on a random number?

- If the seed for any random number generation is a fixed value, there will be no issue.
  However, if the random number seed is based on a variable, such as a time-based seed, the data used for simulation will be different each time the test bench is executed and the results will be different. Change the test bench to use a fixed seed for any random number generation.

C. Are you using pointers on the top-level interface which are accessed multiple times?

- If a pointer is accessed multiple times within a single transaction (one execution of the C function) the pointer must be a volatile pointer or everything except the first read and last write will be optimized away in observance of the C standard.

D. Does the C code contain any undefined values or perform out-of-bounds array accesses?

- It is possible that a C function will execute and complete even if some variables are undefined before being used and when out-of-bounds array accesses are performed. However, this may result in the co-simulation failing because the random values produced by these operations are now different.

- Confirm that the results of the C simulation are expected and that no output values are the result of "random" data values.

- The top level function array inputs need to be sized appropriately:
  If the C TB uses an array of N elements and the top level always accesses N elements correctly, then the top level input/output must be of size N: it should not be larger than N.

- You might also wish to review using valgrind (details available on the internet) for finding potential out-of-bounds array accesses.

E. Are you using floating point math operations in the design?

- For some of the floating point math operations, the RTL implementation is not 100% identical to the C. As a result you might need to check that the C test bench results are within an acceptable error range rather than perform an exact comparison. Refer to the Vivado HLS User Guide (UG902) section "Verification and Math functions"

- In addition, the RTL model for the floating point cores must be provided for some of the supported RTL simulators. Refer to the Vivado HLS User Guide (UG902) section `"Simulation of Floating-Point Cores"

F. Are you using Xilinx IP blocks and a 3rd party simulator?

- The RTL model for the Xilinx IP must be provided for some of the supported RTL simulators. Ensure that the path to the Xilinx IP HDL models is provided to the 3rd party simulator.

G. Are you using the hls::stream construct in the design?

- A hls::stream is by default implemented as a FIFO with a depth of 1.

- If the design results in an increase in the data rate, for example an interpolation operation, a default FIFO size of 1 may be too small and may cause the co-simulation to stall.

- Analyze the design and use the STREAM directive to increase the size of the FIFOs used to implement the hls::stream.

H. Do you use the __SYNTHESIS__ macro in the C source code?

- This macro causes the C source code to be ignored for synthesis and can result in a difference between the C and RTL simulation.
  Refer to the Vivado HLS User Guide (UG902) section "System Calls"

- In addition, the RTL model for the floating point cores must be provided for some of the supported RTL simulators. Refer to the Vivado HLS User Guide (UG902) section `"Simulation of Floating-Point Cores"

G. Are you using very large data sets in the simulation?

- The co-simulation feature verifies all transaction at once. If the top-level function is called multiple times (for example to simulate multiple frames of video) the data for the entire simulation input and output is stored on disk. Depending on the machine setup and OS this may cause performance or execution issues.

- Use the REDUCE_DISCSPACE option when executing co-simulation. In this mode, Vivado HLS will only execute 1 transaction at a time. The simulation may run marginally slower but this will limit storage and system capacity issues.

Error with Co-simulation or the RTL netlist.

If you are unable to resolve the co-simulation failure, it may indeed be due to an issue with the RTL or it could be an issue with the C/CRL co-simulation feature.
Please check your support options, including forum support or opening a webcase:
http://www.xilinx.com/support/clearexpress/websupport.htm

References

Vivado HLS User Guide, UG902 version 2014.2:
http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_2/ug902-vivado-high-level-synthesis.pdf

Revision History

9th September 2014 – Initial Release, Vivado HLS 2014.2