

ESL techniques for test bench development and acceleration

By Zbigniew (Zibi) Zalewski

Sooner or later in the design process each of us faces the point where the test bench becomes our nightmare and the bottleneck for simulation. It is hard to pinpoint when exactly it happened, but my test bench simulation takes 50% of the total time! What can I do to decrease that? Unfortunately, there is no simple answer to this question. What I could do is to take the simulation accelerator and at least cut the design simulation time, but when such a timing consuming test bench stays in the simulator we can at best double the speed –which isn't really enough to justify the amount of money spent. We need to face the reality of the problem and focus not only on the design, but improve the test bench execution time. The obvious step is to make sure the HDL code is efficient. After that we need to look for alternative ways of test bench development. The most popular way now is system level development **or** ESL (Electronic System Level) which is based on TLM (Transaction Level Modeling).

The basic change is the level of abstraction. We should start looking at the design not from the signal level or – more precisely – the register transfer level (RTL), but from the system level. What does that mean? Put simply, it means you're not working on the signals, you're working on transactions like read or write delivered and received to/from your design. You're looking at your design under test as the black box reacting to high-level operations instead of low level binary data. This way, instead of requiring many simulation events you can transform your test bench into the protocol layer describing the behavior of the project by using the set of transactions.

This is all very well, but what if my design is written in RTL, and I was not assuming any system level interface. Am I stuck now with my test bench improvements or is there a solution for me? Fortunately, there is a way: It is called a transactor. This is a kind of adaptor that translates system level commands to register level signal data. This adaptor allows me to connect the system level test bench with the RTL design.

From theory to reality

Enough of the theory; lets move onto the practical side. In this article I will outline two examples of such a development. One idea is based on SystemC, which is proving itself more and more popular in many design labs and which is being quite well promoted by the big guys of the EDA industry. The other example is based on the well known C/C++ programming which could also be used for system level modeling.

Let's start with the SystemC test bench. Figure 1 shows a configuration with such a test bench, a transactor and the design under test. As already mentioned, the role of the transactor is to translate commands sent from the transaction level test bench to the register level design. Therefore it needs to have transaction and signal or bit level interfaces. We could say the transactor works as a protocol converter. Assuming we have access to an HDL simulator that supports SystemC and, for example, VHDL or Verilog, our main focus will be on developing the transactor and the test bench. While SystemC is used to write the test bench, our transactor can be written in SystemC or in traditional HDL language, depending upon project requirements and our own experience. It is worth mentioning that some IP Core houses already deliver such transactors with a test application for specific communication interfaces (e.g. ATM or Ethernet).

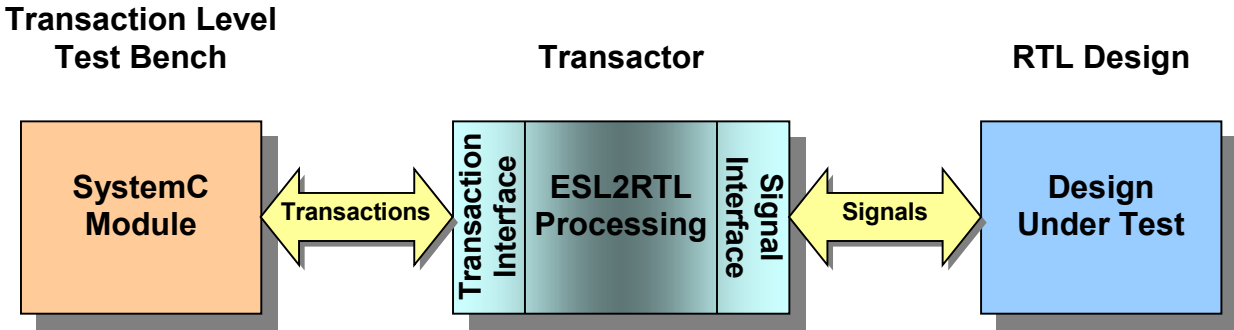


Figure 1. Transaction level design with a SystemC test bench

With such a SystemC test bench and a transactor we can decrease simulation times by a factor of 2 or more compared to simulation with a traditional HDL test bench. Our main gain derives from simplifying the test code into a set of high level commands (e.g. read, write). Another benefit is the possibility of using the same test bench to test a different design with a properly defined transactor.

Of course there is always another side to the coin. Some design engineers argue that SystemC is too complicated, that it is object oriented C++ and that not everybody likes it. Especially the hardware guys are much more familiar with C programming. That's why instead of using SystemC we could try to write our test bench in C or C++. One problem might be that pure C/C++ is not HDL language oriented. It doesn't represent all the HDL language routines and concurrent processing (in the simulation sense), but sometimes it is worth a try as the gains may be even better than with SystemC. Again it all depends upon our specific needs and knowledge. Such a configuration is presented in figure 2. it looks very similar to the

one with SystemC and also needs a simulator that supports mixed level simulation with a C++ code. If that's not available in your simulator you can start with extension interfaces such as PLI or VHPI. Of course, for best results the simulator should support C/C++.

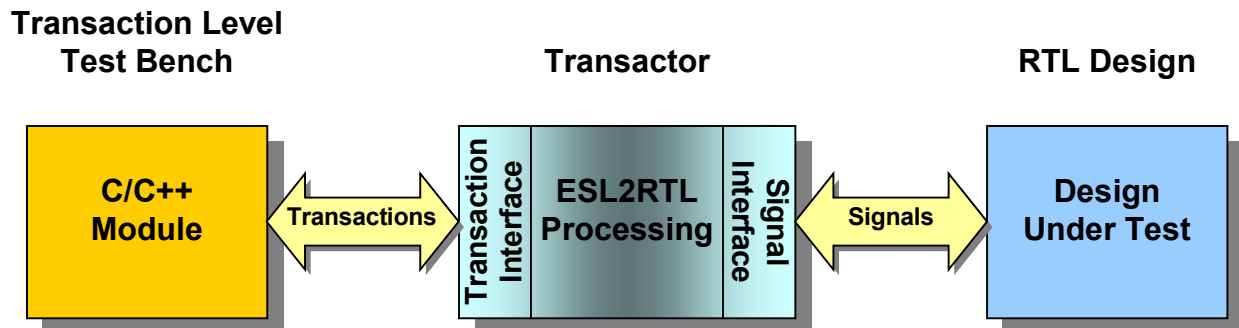


Figure 2. Transaction level design with a C/C++ test bench

This kind of approach could be called custom transaction C. It is easier to program, but offers a limited scope of constructions. The good thing about it is the performance. Since this test bench set up doesn't need a simulation core like in SystemC, we achieve greater execution speeds. Probably the most natural thing to do would be to compile the test bench to the test application.

The possibility to compile the test bench to the executable test application to achieve the highest performance not available in the simulator's environment is an additional benefit of using C-based languages for ESL test bench development. You can do it with SystemC, pure C/C++ or other high level programming languages and extensions.

When interactive debugging in the simulator is not required and we're mostly interested in very fast simulation of our project we could compile our test bench and/or our transactor to the test driver application. But what about the design in such case? Does it still require a simulator? The answer is yes and no. If the simulator allows the addition of an external application communicating with the simulation kernel, we can do it this way.

Going one step further, we can use a hardware accelerator or a prototyping board. Such boards usually have C-based APIs that we could use as connector or a transmission medium for our transaction test bench. Such a software/hardware system level platform could be built as presented in figure 3.

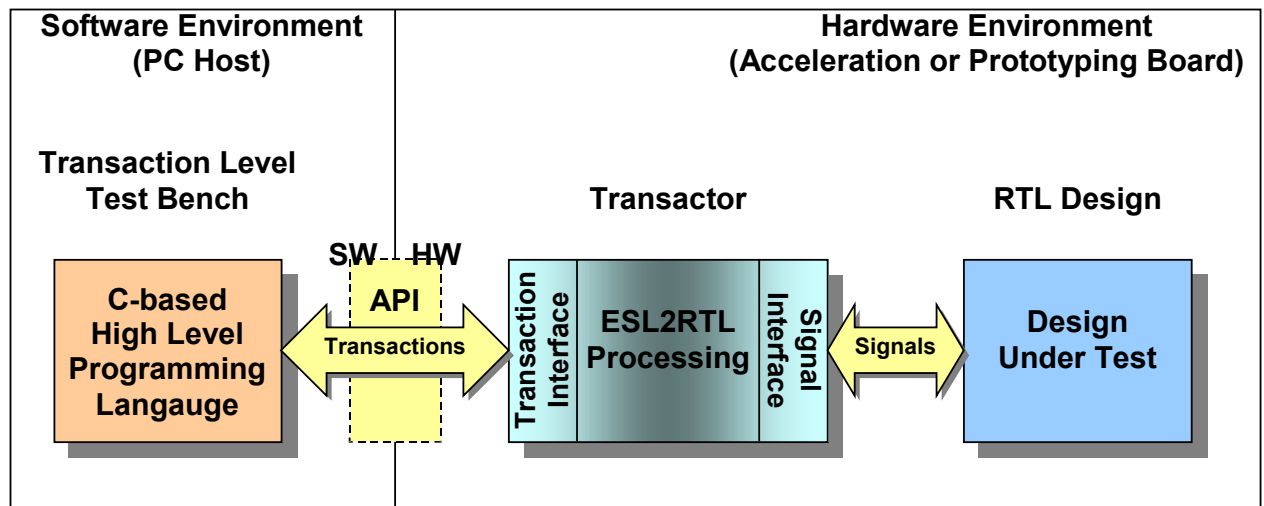


Figure 3. An exemplary software/hardware verification platform for electronic system level simulation

As outlined in this article there are many options of achieving our goal by using electronic system level techniques to speed up test bench simulation. The subject and the variety of options are very wide and the choice of the best solution depends on the specific project requirements. What's important to remember is that design verification is not only about its source code; the testing part is also a key factor. Its efficiency and flexibility affects the overall verification time and the latest ESL innovations are offering new ways of improving testing procedures.