SDL/SystemC co-simulation and code auto-generation

FRUCT 2012

Michel Gillet, Nokia, CTO

NOKIA
Connecting People

Process of writing a specification (1/3)

- When speaking of a specification, most people will think of one document, usually difficult to read and very large.
- But in fact, the process of writing a specification generates many documents:
  - If existing, a formal specification, i.e. written in SDL
  - Text specification
  - Assertion lists, which list of “assertions” of the spec. A simplification is to say that one assertion is one “shall” in the specification
  - Conformance/compliance tests document describing the tests, which an implementation must pass
  - Testing procedure document describing limitations and/or extra requirements to be fulfilled by an implementation to be testable

Process of writing a specification (2/3)

- Based on these various documents, different implementations are made
  - Hardware implementation of the specification
    - to be sold in mass market products
    - by system integration or IP companies, i.e. Cadence, Synopsys, etc.
  - Software implementation of the conformance/compliance tests
    - By system integration companies, i.e. Cadence, etc.
    - Comformance/compliance testing companies, i.e. TestronicLabs, etc.
  - Hardware implementation of the tester
    - companies like Tektronik, Agilent, etc
    - Small companies targeting one or few protocol

Process of writing a specification (3/3)

- In some forum, a specification will be written to define the tester, i.e. MIPI UniPro
- In this case, some additional documents are generated, just for another specification
  - If existing, formal specification for the tester, i.e. in UniPro
  - A text specification
  - Etc.
- Note that having to define a “tester” specification to be able test “a” specification would lead to an infinite number of specifications
  - The loop can be broken by moving some complex part of the tester in the main spec and avoid to have coupled state machines in the tester

Hand made vs. automated (1/2)

- All documents generated by the process of writing a specification are of course all tightly connected to each other
  - One modification in the top of the tree will ripple down to all derivative documents
- In general, the generation of each documents depending of others is a manual process done by the working group writing the specification
  - It induces long delay to propagate a change in all documents
  - The whole process is very error prone
  - Huge administrative overhead is always required to keep track of changes, to which documents they were applied, by whom, when, etc. and across all releases

Hand made vs. automated (2/2)

- As of today, there is little possibility for automation
  - No tools existing
  - As many ways to write a specification than the number of existing specifications
  - Overall is a very large and complex problem
- In earlier Fruct, some attempts to solve portions of the overall problem have been presented
  - SystemC/SDL co-simulation, SystemC/SDL Wrapper

SystemC/SDL co-sim. & wrapper (1/3)

- The starting point is a formal specification written in SDL
- Then a tool from IBM (formerly Telelogic) is used to generate an executable C model of the specification
- Then a SystemC wrapper is made around the C model, by a semi-automated process relying on
  - a database of all the interfaces of the model, i.e. SAPs in this case, defined in a high-level Python script
  - a C++ template library modeling the concept of SAPs in SystemC
    - Implement the input and output of messages queue used to model the SAPs
    - Implement IPC between processes to exchange messages

SystemC/SDL co-sim. & wrapper (2/3)

- What is not automated
  - Handling of input, output messages
  - conversion between internal representation in C by the SDL tool to the external C++ representation for input/output messages and parameters.
- For such work, very skilled individuals are needed
- Unfortunately, such person is very hard to find, which leads to
  - In term of specification work, the cycle of changes integration is too slow
  - This semi-automated solution doesn’t actually help

SystemC/SDL co-sim. & wrapper (3/3)

- The conversion between internal representation in C by the SDL tool to the external C++ representation is the most complex to solve. But to have an automated solution
  - Either the C code generated by the SDL tool need to be parser
  - Either the SDL PR code should be parsed
  - Or both above
- This is particularly true for high-level type or structures used by the upper layers of a specification

One possible alternative

- One possible alternative is to build a compiler taking SDL PR as input and generating directly SystemC code
- However, the SDL PR grammar is very extensive and complex
- Even if the SDL PR grammar is defined in several ITU-T specifications in the form of EBNF, there are quite significant errors
  - Missing production rules
  - Contradicting production rules making the SDL PR grammar ambiguous
- The creation of a compiler from SDL to SystemC is then a major undertaking
  - Probaly equivalent to create a C++ compiler from scratch

Conclusion

- As of today, most specification work is mainly a manual work and thus error prone
  - Every inconsistencies between documents and/or error can cost up to few millions EUR
- Specification work is arguably a niche and not much work is actually done to improve the specification writing process
- However, if there is a formal document for a specification, i.e. SDL, it was shown that it is possible to automate part of the process
  - Creation of SystemC model of both the specification and the HW tester needed for conformance/compliance testing
- But a huge bulk of the process it still entirely manual

Future work

- Automate partly the creating of the text specification from a formal specification, i.e. SDL
- Automate partly or fully, the creation document containing the lists of assertion
  - With full cross referencing with the SDL and text specification
- Automate the generation of the APIs required to interface to tooling companies, i.e. Cadence, Synopsys, etc.
  - Could be a really simple interface if made at the PHY level