RTL Generation of Channel Architecture Templates for a Template-based SoC Design Flow

Jinhyun Cho[1, 2], Soonwoo Choi[1], and SooIk-Chae[1]
[1]*Seoul National University,* [2]*Samsung Electronics Corporation*
{holybit,ssoonoo,chae}@sdgroup.snu.ac.kr

Abstract

In this paper, we propose the design methodology for communication channel templates from formal specification to RTL description. In this flow, design and verification start from one source, LTL property. We constructed LTL-to-TRS, which is translator from LTL property sets to Bluespec term-rewriting system (TRS) description. And, we use a Bluespec compiler as a synthesizer from TRS to RTL. Also, to match the implementation with the formal specification, we use a VIS solver as a model checker. And then, channel instances generated by proposed design method are transformed into channel template-generators for communication channel library. These channel templates can be used in DSE process in SoC design flow.

1. SoCBase-DE: A Template-based SoC Design Flow

To explore the SoC design space more intensively, we developed a SystemC-based design environment, which will be referred to as SoCBase-DE [1]. We provide a communication channel library that defines four categories of abstract channels such as FIFO, array, broadcast, and variable, which can be reused to capture behaviors related to communications and memories of multimedia systems. Furthermore, for each channel, the library provides channel architecture templates (CATs), which are reusable parameterized implementations of the channel. A CAT, which was captured by SystemC, can be refined into various communication architectures while keeping the interfaces of computation blocks. Figure 1 shows an example of communication refinement on the SoCBase-DE design flow. Array channel A can be refined into a SDRAM-based array channel (A'), and array channel B can be refined into an on-chip SRAM based array channel (B').

In SoCBase-DE, designers can easily explore the architecture space of a complex system by selecting appropriate CATs of channels, configuring their parameter values, and changing abstraction levels of CATs.

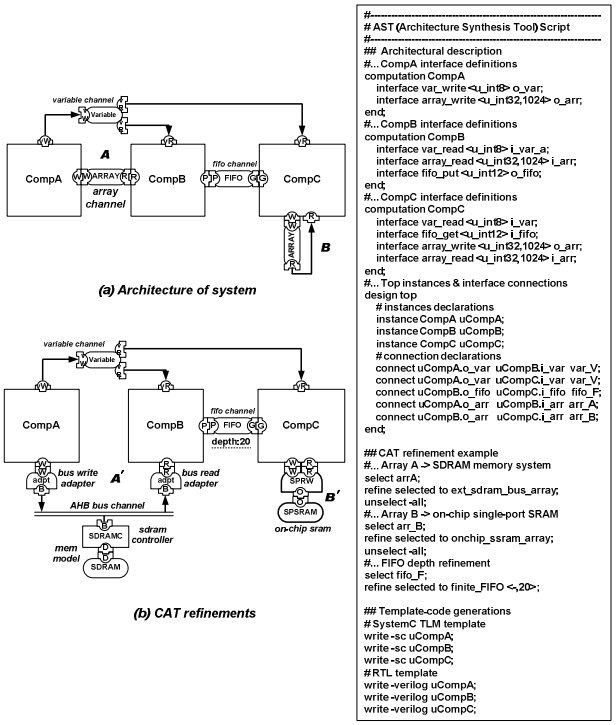

Figure 1. SoCBase-DE CAT refinement

For each CAT, we provide a source code generator that outputs an instance of the CAT configured with specific parameter values at a selected abstraction level. If all the instances of each CAT are completely verified, the correct-by-construction of communication architectures can be guaranteed. Therefore, the confidence level of CATs is very important in the SoCBase-DE design flow.

2. RTL Generation for CATs using Formal Specification

The confidence level of CAT should be much higher just like that of the ASIC standard cell library. In this paper, we propose a CAT design and verification flow.

978-1-4244-2265-4/08/$25.00 © 2008 IEEE

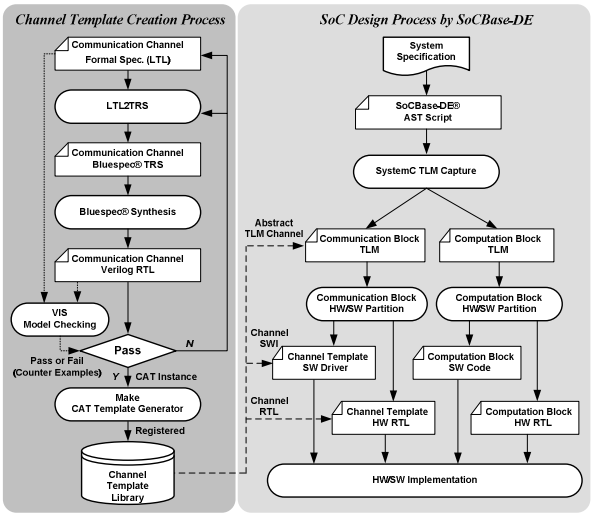

Figure 2. CAT generation for SoCBase-DE

Communication channels have simple functions and precise protocols. Therefore, we try to create the communication blocks automatically by constructing a translator from LTL formulas to Bluespec tem-rewriting-system (TRS) formulas [2]. With a Bluespec compiler, a Verilog RTL can be synthesized from TRS formulas. This RTL implementation is then verified by model checking with LTL formal specifications using VIS solver [3]. Furthermore, these CAT instances are transformed to CAT template generators of the communication channel library to be used in communication DSE on the SoCBase-DE design flow as show in Figure 2.

First of all, input LTL properties are split into atomic properties, which are then modified and transformed into Bluespec TRS. Each execution sequence has inverted indexes [4] to its LTL properties to create Bluespec rules effectively. And the Bluespec rules' conditions are combined from condition sequences indicated by the inverted indexes. It is necessary to make the atomic properties from the input properties for obtaining Bluespec TRS by recombining the atomic properties.

1. if seq1 $\Rightarrow$ seq2 $\wedge$ seq3,
then divide it into seq1 $\Rightarrow$ seq2 and seq1 $\Rightarrow$ seq3
2. if seq1 $\vee$ seq2 $\Rightarrow$ seq3,
then divide it into seq1 $\Rightarrow$ seq3 and seq2 $\Rightarrow$ seq3
3. if seq1 $\Rightarrow$ seq2 $\vee$ seq3,
then divide it into seq1 $\wedge \neg$ seq2 $\Rightarrow$ seq3
and seq1 $\wedge \neg$ seq3 $\Rightarrow$ seq2

In order to determine the task to perform in the current cycle or in the next cycle, it is essential to reduce the number of the operator X less than one. In that form, the signals of the properties can be mapped into registers or wires. For example, in the below property P3, if r1 is high, g1 has to be high in the two cycles later. In other words, if r1 was high in the previous cycle, g1 should be high in the next cycle.

$$P3 : G[r1 \Rightarrow XX\ g1] \Leftrightarrow G[X^{-1}r1 \Rightarrow X\ g1]$$

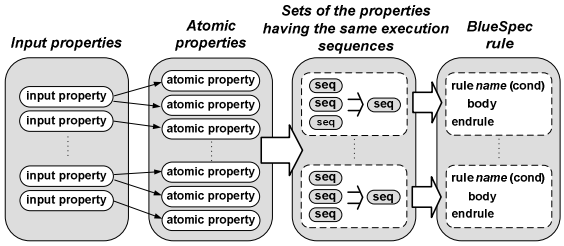

Figure 3. Flow of the LTL2TRS

In this manner, the properties are translated into Bluespec rules when the execution sequence includes the operator X and the signal of the execution sequence is mapped into a register, or when the execution sequence doesn't include the operator X and the signal of the execution sequence is mapped into a wire. The conditions of Bluespec rules are generated from the condition sequences of a set of LTL properties which including the execution sequence. A series of logical Ors of the condition sequences is translated into the conditions. Then the execution sequence is translated into the body of the Bluespec rules. And then, Bluespec TRS description generated from LTL properties is compiled to Verilog HDL by the Bluespec compiler.

As shown in Figure 2, model checking is adapted for verifying CAT instances in the CAT design flow. No matter what Bluespec checks the equivalence from Bluespec TRS to RTL, the model checking flow is needed for detecting errors from LTL to Bluespec TRS. Properties, which are inputs of model checking, are already prepared, because design starts from LTL properties on the proposed CAT generation flow. This enables the designer to unify design and verification for CAT generation from one source.

3. Acknowledgments

This work was supported by ISRC of SNU, BK21, SystemIC 2010, IP/SoC of Seoul, and IDEC, Korea.

4. References

[1] Sanggyu Park et al., "Reusable component IP design using refinement-based design environment," Proc. ASP-DAC, pp.588-593, Jan 2006.
[2] Bluespec, "Bluespec," Website: http://bluespec.com/.
[3] VIS, http://www-cad.eecs.berkeley.edu/~vis/.
[4] http://en.wikipedia.org/wiki/Inverted_index