FUNCTIONAL VALIDATION OF MICROPROCESSOR UNITS BASED ON CONTRACT SPECIFICATIONS

Alexander Kamkin (kamkin@ispras.ru)
Institute for System Programming of Russian Academy of Sciences (ISPRAS)[1]

Abstract

Here we present an approach to functional validation of microprocessor units that is based on contract specifications. Such specifications describe behavior of a unit in the form of preconditions and postconditions of microoperations. Our experience shows that contract specifications are very suitable for testbench automation, since, first, they allow to represent functional requirements on a unit in comprehensible declarative form, and second, they make it possible to automatically construct test oracles, which check unit correctness.

The approach is supported by the CTESK test development tool from the UniTESK toolkit [2, 3]. We have successfully applied our approach to several units of the industrial MIPS64-compatible microprocessor. The approbation has demonstrated the effectiveness of testbench development with the aid of contract specifications. We have found a number of bugs in the implementation of the microprocessor that had not been discovered earlier at the chip level; there are some critical bugs among them.

Introduction

Microprocessors play an important role in present-day life. They underlie all digital computer systems including safety-critical ones, such as airplanes control systems, medical systems of life support, etc. The most commonly used way to ensure functional correctness of a microprocessor is a *simulation-based validation* of its register-transfer-level (RTL) model.

The main task of functional validation is to check correspondence between design under test behavior and *functional requirements*. To have the ability to do it automatically requirements should be represented in machine-readable form. Such form of requirements representation is usually called *formal specifications* or *specifications* for short.

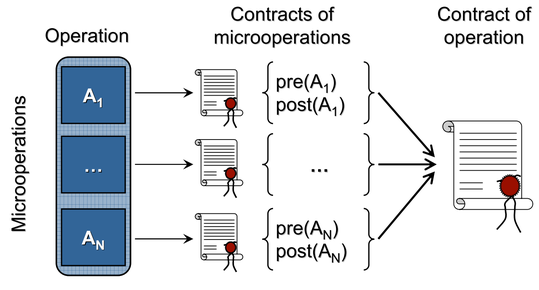
In this work we consider specifications of a specific form, so-called *contract specifications*. The central metaphor of the approach is borrowed from business. Components of a system interact with one another based on mutual obligations and benefits. If a component provides the environment with some functionality, it may impose a *precondition* on its use, which determines an obligation for the environment and a benefit for it. The component also guarantees execution of a certain action with the help of a *postcondition*, which determines an obligation for it and a benefit for the environment.

Contract of operation A

pre(A) – obligation
post(A) – benefit

If a client meets the precondition, then the component must fulfill the postcondition

Specification of Requirements



The process of specification of an operation can be outlined as follows. First, a precondition restricting situations in which the operation can be supplied for execution is determined. Based on the analysis of the requirements, functional decomposition of the operation into a sequence of microoperations is carried out. For each microoperation a postcondition describing requirements to it is defined. Then, the postcondition of each microoperation is associated with the cycle at the end of which it should be fulfilled. Thus, contract of the operation consisting of n microoperations is formalized by the structure:

$$\text{Contract} = \langle \text{pre}, \{(\text{post}_i, \tau_i)\}_{i=1,n} \rangle,$$

where pre is the precondition of the operation, post_i is the postcondition of the i^{th} microoperation, and τ_i is the number of the cycle when the i^{th} microoperation is executed.

For the purpose of clearness hereinafter we consider such operations, that their microoperations are executed sequentially, i.e., $\tau_1=1$, …, $\tau_n=n$. This assumption does not restrict the generality. In this case contract of the operation consisting of n microoperations is given by the formula:

$$\text{Contract} = \langle \text{pre}, \{\text{post}_i\}_{i=1,n} \rangle.$$

In general case there are data dependencies between operations; execution of an operation is suspended until all necessary data is prepared and all required resources are deallocated by previous operations. Requirements on such operations are specified with the help of the following contracts:

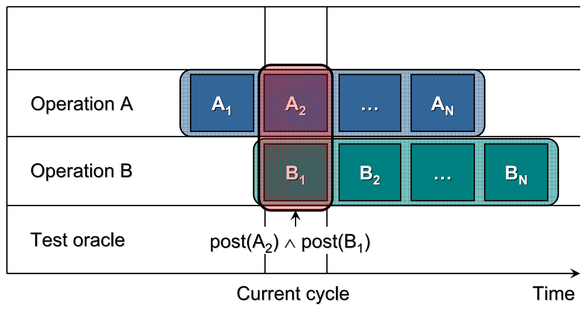
$$\text{Contract} = \langle \text{pre}, \{(\text{pre}_i, \text{post}_i)\}_{i=1,n} \rangle,$$

where pre is the precondition of operation, pre_i is the interlock condition of the i^{th} microoperation, and post_i is the postcondition of the i^{th} microoperation.

Validation of Requirements

Suppose that at some moment of time the unit under test performs m operations X_1, …, X_m (without interlocks for clarity) that have been supplied for execution $\tau_1=m$, …, $\tau_m=1$ cycles earlier, respectively. Let, at the moments when the operations were supplied, their preconditions have been fulfilled. Then, to validate correctness of the unit behavior at the given moment of time it is required to check satisfiability of the predicate called *test oracle*:

$$\text{Oracle}(X_1, \tau_1; \ldots; X_m, \tau_m) = \text{post}(X_{1,m}) \wedge \ldots \wedge \text{post}(X_{m,1})$$



Test oracle organization for pipelined operations is illustrated in the figure. In the first cycle operation A is started. Then, one cycle later operation B is supplied for execution. At the end of the second cycle both postcondition of microoperation A_2 and postcondition of microoperation B_1 should be fulfilled.

To validate such requirements testbench should keep track which microoperations are finished and check corresponding postconditions. Special mechanism that in each cycle of simulation calculates a set of postconditions to be checked is called *temporal composition of contracts* [4, 5].

Experience

The first unit that we have specified and validated is the translation lookaside buffer (TLB). We have found nine bugs including very critical ones. Most of the bugs (67%) are high-quality bugs in pipeline organization. We have also validated the floating-point unit (FPU) of the microprocessor. In this project we have discovered one critical bug in the SQRT.D operation (square root for normalized double-precision numbers). The other unit that we have applied our approach to is the arithmetic logic unit (ALU) of the microprocessor. In that case we have not found bugs in the unit implementation.

Now we are developing specifications and tests for the L2 cache of the microprocessor. It is a direct-mapped 256 KB cache that consists of sixteen 16 KB blocks; access to each block can be done independently. The microprocessor supports direct memory access to the L2 memory via load/store instructions.

Unit	Size of RTL model / specifications, KLOC	Number of operations / pipeline operations	Number of requirements	Number of errors / pipeline errors
TLB	~ 8.0 / 2.5 (31%)	5 / 5	~100	9 / 6 (67%)
FPU	~ 6.0 / 1.0 (17%)	4 / 0	~ 40	1 / 0 (—)
ALU	~ 8.0 / 0.5 (6%)	43 / 0	~ 50	0 / 0 (—)
L2	~ 2.5 / 2.5 (100%)	4 / 4	~ 250	In progress

References

1. http://www.ispras.ru
2. http://www.unitesk.com
3. A. Kamkin. *The UniTESK Approach to Specification-Based Validation of Hardware Designs*. ISoLA'06: The 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, November 2006.
4. A. Kamkin. *Contract Specification of Pipelined Designs: Application to Testbench Automation*. SYRCoSE'07: The 1st Spring Young Researchers' Colloquium on Software Engineering, 2007.
5. A. Kamkin. *Testbench Automation for Pipelined Designs Based on Contract Specifications*. EWDTS'07: The 5th East-West Design & Test Symposium, 2007.