Verification of the CAD System for an Application-Specific Processor by Property-Based Testing

Daniil Prohorov
Faculty of Software Engineering and Computer Systems
ITMO University
Saint-Petersburg, Russia
daniilprohorov@gmail.com

Aleksandr Penskoi
Faculty of Software Engineering and Computer Systems
ITMO University
Saint-Petersburg, Russia
aleksandr.penskoi@gmail.com

***Abstract*—An application-specific processor CAD system is a complex software system with high dependability requirement. Verification of such system is a complicated process which involves many closely connected components, different computational process representation and tools, and requires much time and work. In the article, the authors describe their experience of usage property-based testing in the NITTA project (application-specific reconfigurable processor CAD for real-time application). Property-based testing allowed to significantly improve source code coverage and identify many hard to find bugs. The article will be interesting to CAD developers and as a real-world property-based testing application example.**

Keywords—CAD; embedded system; ASIP; high-level synthesis; property-based testing; NISC; TTA; verification; CoDesign

I. Introduction

Embedded systems have to solve highly specific problems with time, power consumption limitations, and other restrictions that do not allow using general-purpose products/programs/processors to solve specific types of problems with constraints. Specialized systems require specialized tools for developing. The embedded industry has many platform architectures for developing, starting from multiple types of microcontrollers ending with Field-Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC). Each platform has a unique tool for developing, SDK, or CAD systems, and they may have errors because the embedded industry is not big enough for all these tools testing. Errors in these systems can bring significant and expensive effects. For example, software error in the industrial controller leads to the situation when all parts of the factory may be stopped that causes them to lose money and clients.

In the comparison of developing systems for embedded and PC industries, it can be seen that PC has more money, more people, and more standard solutions. That lets developers not thinking about issues in development tools and operational environment. But embedded systems significantly depend on problem-specific requirements, hardware architecture, customer preferences. It raises a count of used project tools (compilers for specific processors, simulators, scripts for code generation, etc.), which usually have low comparative acceptance and, as a consequence, are less tested.

This article will be considered the problem of CAD verification for developing application-specific processors by Property-Based Testing (PBT). Our verification system works with the following parts:

- internal parts of the CAD system (synthesis algorithm and processor unit models);
- hardware implementation of processor units;
- a target computer system, which contains generated hardware and interchangeable software and implements an application algorithm.

The main component of our CAD system is the algorithm of target system scheduling and synthesis. It has to be tested for working properly with different input cases. One of the most expensive and hard to identify issues in development is internal errors of the synthesizer/interpreter CAD system and other tools for developing, and we want to predict them. Another kind of hard to find issues are inconsistencies between hardware parts and synthesis parts (model and hardware implementation of processor unit).

In the article, we show an example of using PBT for complicated real-world cases. We observe the NITTA project, describe the verification problem, overview of verification approach for such systems, and explain how PBT can help developers to improve the quality of their software system.

II. NITTA

Not Instruction Triggered Transport Architecture (NITTA) is the CAD system for developing real-time reconfigurable application-specific processors. It is a complicated program that makes developing Application-Specific Instruction-set Processor (ASIP) systems and software for it faster and easier. Used approaches similar to high-level synthesis [1], but with keeping programming abilities.

The research was supported financially by Russian Foundation for Basic Research (Project No. 18-57-45004)

NITTA processor organization is the combination of two different architectures [2]:

- No Instruction Set Computer (NISC, Figure 1);
- Transport Triggered Architecture (TTA, Figure 2).

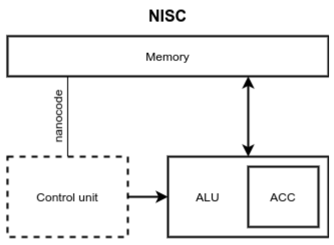

Figure 1. Not Instruction Set Controller

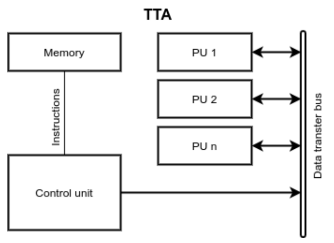

Figure 2. Transport Triggered Architecture

NITTA architecture (Figure 3) takes advantage of NISC and TTA [3]:

- From NISC, we take an idea of using nano code. It makes the target processor faster (by avoiding instruction set abstraction level and direct control over hardware) and lighter (control unit degenerates to memory). The cost of it is a complicated synthesis and scheduling process, but it can be done in a design-time.
- From TTA, we take a data transfer network (bus) with universal Processor Units (PU), that let us have one structure for all kind of operations with data, including storage and transfer.

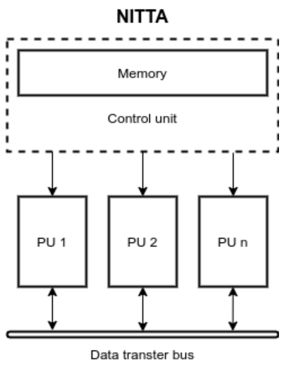

Figure 3. NITTA microarchitecture

Today NITTA has a few process units:

- Fram – data storage;
- SPI/I^{2}C/PCIe – for input / output;
- Loop – data transfer between process cycles;
- Accum, Div, Mul, Shift – for mathematical operators.

To make the target ASIP, a user should set the target application algorithm and microarchitecture of the processor (processor unit set and interconnect configuration). The algorithm has to be written on Lua language subset and implements the primary function cycle that repeats multiple times. The algorithm in Lua transforms to a function unit set. Processor units can execute each of them.

For all process units, CAD contains its models with the same interface. That models implement a few actions: for binding and getting options for data sending or receiving by processor unit from the internal processor network.

The synthesis process starts after the Lua code has transformed into functional units. At first, it tries to bind all FUs to PUs. If it has been completed, then the synthesis algorithm gets options about data transferring to PU and receives a result with a list of endpoints with timings. CAD finds all possible transfers between process units. Otherwise, it takes a step back and tries another option. When the synthesis process ends with success, then the target project can be generated.

For correct work, all parts of the CAD system and library should be consistent. For example, if the hardware needs to make computation done in two clock cycles, CAD should know it to generate a correct schedule. If schedules don't have enough time – another processor unit gets corrupted data. If the program has more time, the processor unit will be inefficient.

III. VERIFICATION

In the software development process, one part of the work is testing. At first, you can do it manually, but when the project grows, you have to be sure that your solution is working after changes properly. For these purposes, you have to apply some testing or error finding methods that can be static or dynamic.

Verification methods can be separated into the following groups, static verification check code without executing and try to find bugs or inconsistency. Dynamic verification vice versa execute code in a test environment and try to find unhandled exceptions or incorrect behavior in runtime. Static verification usually works with system properties and proves their correctness for all use-cases (which consume a lot of time). Dynamic verification allows to quick check behavior in a very limited number of specific cases, but test development with high quality also consume a log time. From that point of view, property-based testing is a hybrid approach, which connects system properties in general representation and test case checking.

A. Static Verification

To verify a program, developers can use specialized languages like Coq to formally prove program properties. But this group of languages is tough to use by developers, and they have made like proof assistant tools, not like general purposes language.

Another option – using strong static typing, helps to write fewer tests, and find errors at compile time. In the NITTA project, we use Haskell because it has a Hindley-Milner type system for types deducing [4]. Type system lets developers write code without type definitions on each expression and let compiler check types in compile time.

Also, it is possible to use linter tools that statically check your code from mistakes related to brackets or words correctness, deadlocks, typical security issues.

B. *Dynamic Verification*

Dynamic methods can be divided into two parts: unit and integration tests.

Unit tests are writing for each module and work in the runtime process. But the main problem of unit tests is that you have to write many cases to have enough coverage. All this test depends on code structure, and if you do a few changes in code, you have to rewrite part of unit tests. But the main advantage of unit tests is that you can find and quickly fix it because you know where your system has fallen.

Integration tests verify how your modules interact with each other. They are expensive for developing, and if you find a bug, you can not understand in which function or method it is. Because modules are involved, and each of them can contain thousands of lines of code. In the NITTA project, most of the tests should be integration tests because they need to check the coordination of different parts of the CAD system and its library.

C. *Property-Based Testing*

The main idea of PBT is to use system properties to test it on random input data. For test case creation, we take the property and generate data for this property [5]. For example (see Listing 1), $a + b = b + a$, here we can test the commutative property of addition operation. To make this test work, we need to specify what a and b values would be. With static typing, we can specify it more comfortably, but we also can do it by creating a generator by yourself. If we say that a, b would have type *Int*. After these inputs exits, we can start the property-based testing process. In this process, the test environment randomly generates *Int* values for a and b, puts it to property, and if the result is false, it means that we have an error in this case. Also, our system can print the test case and input values with this error.

In the project, we use the standard library for PBT - QuickCheck, where we need to implement an arbitrary function for each type that we want to test in the property [6].

In the last example, we have seen that the sqrt function can work only with values above zero.

In the comparison of unit tests and PBT, they do the same testing (testing units), but using different methods for it. It means that the Unit test can be replaced by PBT in a lot of cases [7].

```
> quickCheck ((\(a, b) -> a + b == b + a) :: (Int, Int) -> Bool)
+++ OK, passed 100 tests.
```
```
> quickCheck ((\(xs) -> reverse (reverse xs) == xs) :: ([a]) -> Bool)
+++ OK, passed 100 tests.
```
```
> quickCheck ((\x -> sqrt (x ^ 2) == x) :: Double -> Bool)
*** Failed! Falsified (after 2 tests and 4 shrinks):
-0.1
```
```
> verboseCheck ((\x -> sqrt (x ^ 2) == x) :: Double -> Bool)
Passed:
0.0
Passed:
0.35527320547774477
Failed:
-0.9512445230341
```

Listing 1. Examples with QuickCheck

IV. Property-Based Tests in context of NITTA

In the NITTA project, we have two abstractions: processor units and functions. They all represent the computational process element but from a different point of view. A processor unit represents a target processor component, which allows executing various functions. A process unit implementation includes source code on hardware description language (Verilog) and the CAD model, which describe how to use its unit. A function specifies target algorithm parts. In the classical processor, it is called instruction.

The project analysis allows to determine the following properties:

- Synthesis process completeness – all functions and transferred variables of a target algorithm are represented in target system software. It enables founding rough bugs in the synthesis process or processor unit models.
- CoSimulation – functional simulation of a target algorithm must be identical to the logical simulation result of the target system. It allows founding scheduling bugs in the processor unit implementation (model and hardware), including its inconsistencies.

Data flow of property-based testing for the NITTA project represented in figure 4, where a rectangle is a CAD component process, a rounded rectangle is a PBT testing component.

For algorithm generation for each PU implemented function *arbitrary* that generates functions which this PU can solve. For example, Accum PU takes Sub and Add functions. Fram PU can evaluate Loop, Reg, and Const functions.

Functions also can be used as part of a CAD functional testing process because each of them can be simulated for getting test data.

For example, if an algorithm contains a division function, the CAD can simulate it. That takes two input values, divides it, and produces test data. It is an essential part because these features allow CAD to compare the results of the target system with results in function simulation. In this comparison, we have built a property for PBT. But to compare Verilog results and function simulation, we must have logic simulation. That takes function simulation and translates it to logic.

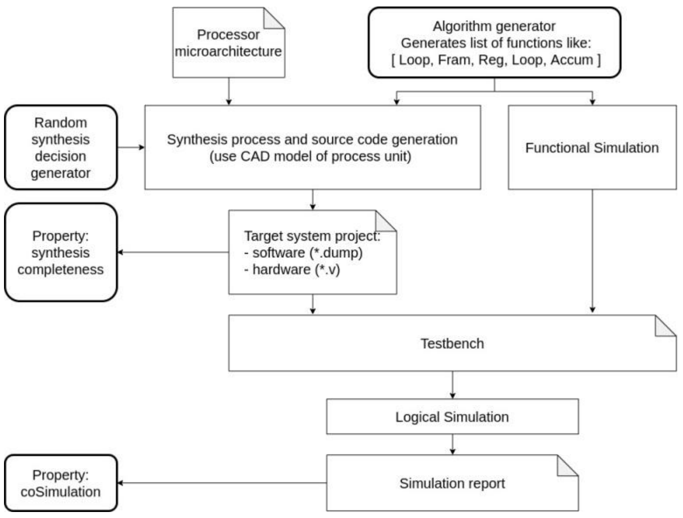
Processor microarchitecture
Algorithm generator
Generates list of functions like:
[Loop, Fram, Reg, Loop, Accum]
Random synthesis decision generator
Synthesis process and source code generation
(use CAD model of process unit)
Functional Simulation
Property: synthesis completeness
Target system project:
- software (*.dump)
- hardware (*.v)
Testbench
Logical Simulation
Property: coSimulation
Simulation report

Figure 4. Property-based testing data-flow for the NITTA project

After the creation of random function from the list of available, CAD randomly tries to synthesize the evaluation process. For these purposes, we have a unique function that has a list of tasks. Each task contains a Bind or Transport data type. Bind has made for trying to connect function from the algorithm to PU. If we can't do it, we just skip it. Transport has been made for transfer data between PUs if one PU can unload value, and another can upload at the same time we transfer this data.

```
VCD info: dump file web_ui_net_tb.vcd opened for output.
set data for sending [] by spi_io_test_input
set data for sending [] by spi_io_test_input
cycle:0     tick:0    actual:0
cycle:0     tick:1    actual:0    expect:0    x#0
cycle:0     tick:2    actual:1    expect:1    1@const#0
cycle:0     tick:3    actual:0
cycle:0     tick:4    actual:0
cycle:0     tick:5    actual:1    expect:1    y#1
cycle:0     tick:6    actual:0
cycle:      tick:0    actual:0
cycle:1     tick:1    actual:1    expect:1    x#0
cycle:1     tick:2    actual:1    expect:1    1@const#0
cycle:1     tick:3    actual:0
cycle:1     tick:4    actual:0
cycle:1     tick:5    actual:2    expect:2    y#1
cycle:1     tick:6    actual:0
```

Listing 2. The test bench output

CoSimulation property should be explained in more detail. That property generating a target project (including hardware and software parts) and puts it in the operational environment of a testbench. That testbench mocks all input and output data channels for full emulation of an operational environment and asserts internal NITTA processor states for several computational cycles. If these assets do not fail, the test system can conclude the consistency behavior of the NITTA CAD system.

The automatic random target algorithm and synthesis process decision generation allow making a thousand checks when the count of manually written test stops on tens.

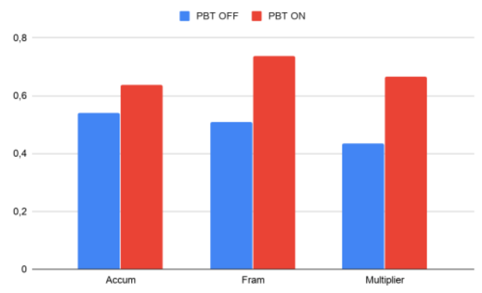

Figure 5. Test coverage statistics

In Fram PU, PBT has found many internal model issues, which caused by resource restrictions and concurrent function evaluation.

V. Conclusion

The article considered a verification problem in the CAD system for application-specific processors. Overview and analysis of static and dynamic verification methods are presented. Method of integration testing based on property-based testing was proposed. It allows checking the CAD internal consistency between the synthesis method, process unit models for a synthesis process, and hardware implementation of processor units (main building blocks of a target application-specific processor). That method significantly improve existed source code coverage for the NITTA project and reduced the number of unit tests for new functions and processor units. Also, the proposed verification method helps to find some difficult bugs related to modeling internal process unit concurrency and resource restrictions.

References

[1] Martin G., Smith G. High-Level Synthesis: Past, Present, and Future // IEEE Design & Test of Computers. 2009. Vol. 11. P. 18-25.

[2] P. Mishra and N. Dutt, *Processor Description Languages*. 2008.

[3] Penskoi A., Yanalov R., Romanov G., Platunov A. Hybrid NISC/TTA High Level Synthesis Tool // 18th International Multidisciplinary Scientific GeoConference Surveying Geology and Mining Ecology Management, SGEM-2018 - 2018, Vol. 18, No. 2.1, pp. 453-460.

[4] B. C. Pierce, "Types and programming languages: the next generation," 2003.

[5] Fink G., Bishop M. Property-based testing: a new approach to testing for assurance // ACM SIGSOFT Software Engineering Notes. 1997. Vol 22. P. 74-80.

[6] Koen C., John H. QuickCheck: a lightweight tool for random testing of Haskell programs. September 2000. available at: https://doi.org/10.1145/351240.351266

[7] Tom Sydney Kerckhove, QuickCheck, Hedgehog, Validity. February 2019. available at: https://www.fpcomplete.com/blog/quickcheck-hedgehog-validity (Mar. 2020)