Development of an approach to automatic test generation based on the graph model of a cache hierarchy

Garashchenko, A.V., Gagarina, L.G., Kyaw Zaw Ye, Dorogova, E. and Kochneva, M.

Development of an Approach to Automatic Test Generation Based on the Graph Model of a Cache Hierarchy

Anton V. Garashchenko[1], Larisa G. Gagarina[2],
Kyaw Zaw Ye, Ekaterina Dorogova
National Research University of Electronic Technology
Moscow, Russia
[1]ant.gar1@mail.ru; [2]gagar@bk.ru

Maria Kochneva
Faculty of Art, Science and Technology
Wrexham Glyndwr University
Plas Coch, Mold Road, Wrexham, LL11 2AW, UK

Abstract—**Verification of the cache hierarchy in modern SoCs due to the large state space requires a huge amount of complex tests. To cover the entire state space of the cache memory hierarchy graph model is proposed. The generation of tests based on this model, whose vertices (V) are the set of states (tags, values, etc.) of each cache and the edges (E) are the many transitions between states (instructions for reading, writing). Thus a graph model is constructed that describes all the states of the cache memory hierarchy. Each edge in the graph is a separate verification sequence. Vector-block operation with memory is provided. The approach described in the paper showed a good result when checking the hierarchy of the multi-port cache memory of the developed kernel with the new vector VLIW DSP architecture, revealing several architectural and functional errors. Further, this approach will be applied to test other processor cores and their blocks.**

Keywords—automatic test generation; testing of the processor's cache; verification of heterogeneous system-on-chip; cache verification

I. Introduction

One of the most important trends in the development of modern microelectronics due to a decrease in the technological process of semiconductor production and an increase in the degree of integration of microcircuits is an increase in the performance of computer systems by increasing heterogeneity which has led to the emergence of new classes of processors, such as networks and systems (SoC) on a chip. Multicore heterogeneous computing systems on a chip consist of general-purpose cores and specialized computing cores with different architectures. This is due primarily to the well-known advantages of using different types of processors to perform individual classes of tasks [1]-[3].

One of the decisive factors affecting the performance of computing systems is the required time to complete one memory access operation [4] since the processor clock speed is an order of magnitude higher than the memory frequency. It should also be borne in mind that the time of interaction with memory is subject to the influence of other access operations, is limited by the capacity of the signal bandwidth, and is determined by the volume and architecture of the memory subsystem.

The cache replacement policy or caching algorithm is the main design parameter of any memory hierarchy. The effectiveness of the replacement policy affects both the frequency of access and the delay in access to the cache system. The higher the associativity of the cache, the more important the cache algorithm becomes. There are a large number of replacement policies, and each of them represents a compromise between the frequency of calls, cost (in terms of the required hardware resources) and performance: least used (LRU), most recently used (MRU), PseudoLRU, least used (LFU) and many others. The following are the details of the LRU policy since the case study presented in Chapter 2 uses a cache that implements this replacement policy. The LRU cache algorithm is aimed at reducing the cache loss rate in associative-multiple caches by replacing a block that has not been used for the longest when cache loss occurs. The replacement logic should also update the correct ordering among the elements of the block on each cache access based on their time in the cache, regardless of access to the cache causes a hit or miss.

Currently, in highly computing systems, which include SoCs, a hierarchical memory consisting of several levels is used to organize the most effective, in terms of time, access to memory; these are most often registers, instruction and data cache memory, temporary buffers for address translation, RAM, virtual and hard drives [5]. The hierarchy is formed according to the principle - the higher the level, the faster the speed of access to it, and the memory size is less, since high-speed memory is quite expensive, in terms of hardware costs is space on the chip, so its volume cannot be large. This approach becomes effective if the data is stored in a memory hierarchy in order of frequency of their use.

At the current stage of development of microelectronics, product design takes place at the level of RTL-description of SF blocks. Design is a multi-stage process. Due to the architectural complexity of modern multi-core processors used in developing systems on a chip, more than sixty percent of design resources are spent on their verification. This is due to the high combinatorial complexity of verifying the correct operation of both individual cores and the system as a whole, so functional verification becomes one of the most important stages in the design of SoCs.

In modern SoCs, including heterogeneous ones, the cache hierarchy is one of the most architecturally complex blocks in the memory subsystem due to the huge number of its functional state states. It is necessary to develop a huge number of verification sequences, sort through a large number of states to perform a full functional check of such complexity units. Modern formal verification algorithms have an exponential dependence on the complexity of their development on the size and complexity of the subsystems and blocks to which they are applied. Formal algorithms are usually applied to small blocks or subsystems (ALU, MMU, or TLB) or to individual computational properties of the kernel, which are easy to localize [6]. In addition, in order to develop a complete set of all kinds of tests, it is necessary to spend large time resources, which is impossible within the framework of the modern design route. Therefore, it is necessary to automate the development of formal tests, which is a non-trivial task. Thus, dynamic verification is still one of the main methods for checking any processor unit, including the cache hierarchy of computing cores. However, the task of automating the creation of verification sequences for a full check of the processor core and the creation of new tools for verifying the correct operation of multinuclear structures remains relevant.

Test generators have long served as the main tool for covering computational cores with tests [7]-[9]. The development of modern generators of verification sequences with the generation of edge and critical situations for the cache requires a lot of time since the size of the space of functional states of the hierarchy of the multiport cache is about 1016. Different architecture and coherence protocols make it difficult to transfer developed generators and tools to new projects. Another limiting factor is the time required to run the test sequences, as the modeling of modern SoCs is growing. Therefore, the verification test generator to achieve the required coverage should consist of a sequence of a minimum number of simulation instructions. When generating, it is also necessary to take into account the global space of functional states of the entire system, i.e., it is necessary to create situations that are rudimentary for working with all the blocks and subsystems of the processor. An example of such a functional state can be the execution of a given combination of program control commands in parallel with the execution of cache access instructions that conflict with each other when accessing the cache line. A random stream of simulation instructions constructs such a state over the years of simulation. Directional manual tests may not reach the boundary state at all, since the verifier may not put such an algorithm into the patterns of their behavior.

The verification of the cache hierarchy and coherence protocols, including the development of effective algorithms and methods for generating tests, is considered in many scientific papers [3]-[6]. These works are devoted to a deep study of individual methods for generating stressful situations for cache memory, which indicates the need for research in this direction.

The proposed methods for constructing a graph model of a multi-port cache hierarchy for generating test sequences, within the framework of the current study, will improve technical and economic indicators about systems already developed on the market [6], [7], [9].

The technical and economic characteristics of the verification process depend on the quality and efficiency of the test generation algorithms. Therefore, the urgent problem is the need to increase the completeness of the test coverage and speed up the verification process of the cache hierarchy of heterogeneous multicore structures.

II. Related Work

The cache memory of heterogeneous SoC is an intermediate fast buffer with the most frequently used data from less high-speed types of memory. In this part of paper verification sequences with the generation of edge and critical situations generating based on the graph model of the hierarchy of the multiport cache memory is described.

To cover all states from functional space of the hierarchy of the multiport cache, a graph model building according to the following rules is suggested:

- G is the directed graph model (Fig. 1), G = (V, E) (classic definition);
- V = {v1, v2, … vn} – is the set of vertices – hierarchy of the multiport cache functional state (level, tags, values of the ways, etc.);
- |V| depends on the configuration of the memory subsystem;
- E = {e1, e2, … en} – is the set of edges – instructions of cache memory hierarchy accessing for writing and reading;
- |E| depends on the configuration of the memory subsystem (in described model |E| = 5);
- V ∩ E = □. f : vi □ f(vi), vi+1 = f(vi), where f ∈ E.

Graph model (G) built according to the rules above for one level of cache hierarchy with only one channel, 4 ways, LRU policy and |E| = 2 consists of about 108 vertices.

Now we need to define a Labeled Transition System (LTS) is a state/transition graph. States in this graph model don't provide information except for the changes in states. The information is represented in the labels (actions or transitions). Theoretically, we use an LTS for generating verification sequences of metalanguage instructions.

Theoretical definition an LTS is the directed graph model G = (V, E, T, v0), where:

- V = {v1, v2, ... vn} – is the set of vertices – hierarchy of the multiport cache functional state (level, tags, values of the ways, etc.);
- |V| depends on the configuration of the memory subsystem;

- E = {e1, e2, … en} – is the set of edges – instructions of cache memory hierarchy accessing for writing and reading;
- |E| depends on the configuration of the memory subsystem (in described model |E| = 5);
- T ⊆ V × E × V – is the conversion relations set;
- v0 ∈ V is the initial state, v0 ∈ V;
- V ∩ E = ∅ . f : vi ↦ f(vi), vi+1 = f(vi), where f ∈ E.
- T – transitions set;
- ti - ▪ vi, ej, vi + 1 ▪ , ti ∈ T;
- fi(vi) → vi + 1) means that the graph model has transitioned from state vi to state vi+1 by applying the edge ei;
- L = {L1, L2, L3}, where Li – level in cache memory hierarchy;
- C = {c1, c2 ... cn} – value from cache line (used for comparing);
- H = {G0 ∪ G1 ∪ ... ∪ Gn}, where n = count (L), the subgraphs (hi) of which are the graphs G of each cache level;
- S = {s1, s2 ... sn}, where si = {ei, ek, ej}, ei, ek, ej ∈ E is the set of transition sequences;
- P = {p1, p2 ... pn} – set of the cache extrusion strategies;
- updated(H, si): vi↦ updated(vi), vi + 1 = updated(vi), where si ∈ S is the transition function of the entire cache hierarchy to the next state.

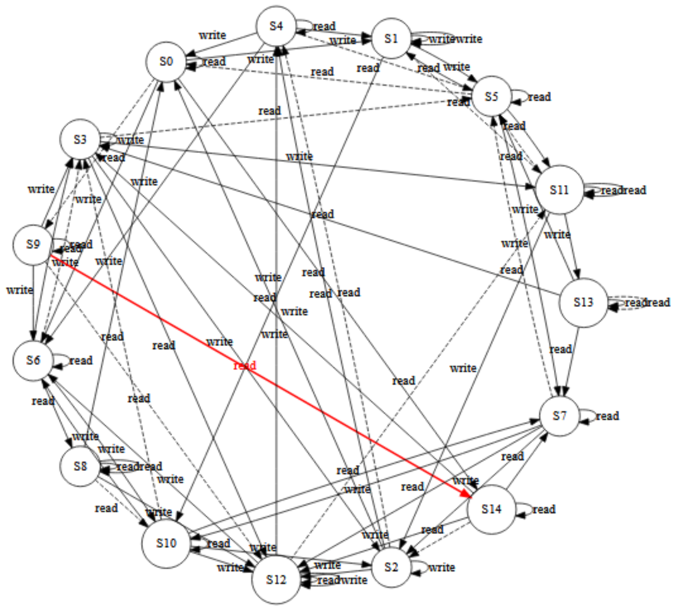


Fig. 1. Graph example.

Fig. 1 shows an example of an LTS graph model for a cache hierarchy consisting of one level containing 1 row with LRU policy and 4 ways, where | E | = 2 (memory access instructions for reading and writing).

The graph model for two levels of the cache hierarchy with the LRU (4 ways and one channel) substitution policy contains about 108 vertices.

The cache hierarchy of modern heterogeneous structures consists of several levels (usually of the order of 2-4). The Li cache buffers access the Li + 1 cache. The last level cache is the largest, and the data comes from RAM (Random Access Memory). Depending on the inclusiveness of the cache, data can be located immediately at all levels of the hierarchy, or only in one - these types of cache are called inclusive and exclusive. When accessing RAM, the processor checks whether the required data is in the buffer (without taking into account coherence in multi-core structures); if the data is in the buffer (getting into the cache), it is taken from the cache. Otherwise (lack of cache), one of the data blocks (depending on the preemptive policy) contained in the buffer is replaced with other data from RAM.

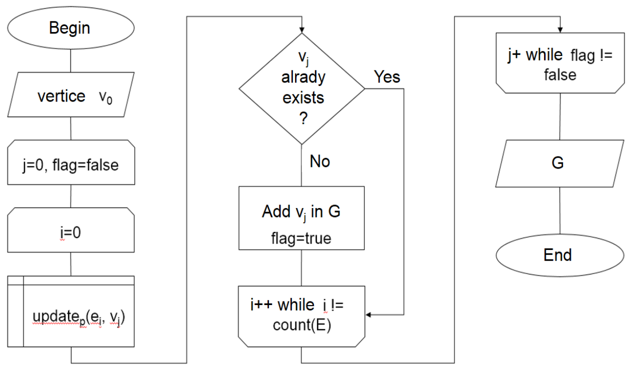


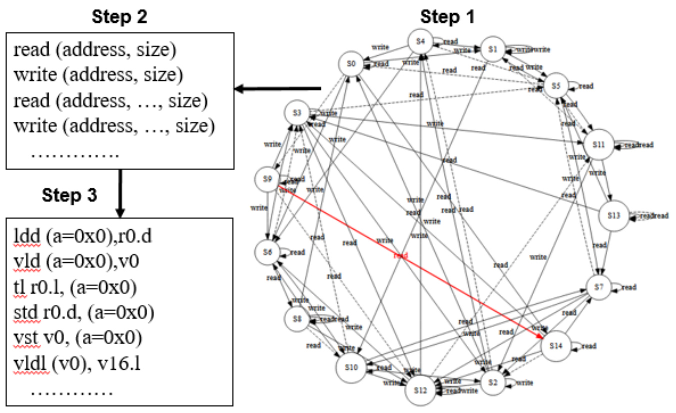
Fig. 2. Algorithm for constructing a graph model of the cache hierarchy.

In order to cover the entire space of functional states of the cache hierarchy, in the proposed paper, we construct the graph model H (H = G0 ∪ G1 ∪ ... ∪ Gn), where n = count(Li), the subgraphs (hi) of which is the set of graphs (hj = Gj , where Gj = {g1, g2, ... gn}) of each cache level and directed edges E is a set of transitions (memory access operations for writing and reading) between the states of the cache hierarchy.

When a channel is selected at the cache level from the hierarchy, even without taking into account the inclusiveness, a non-deterministic situation arises. It will not be possible to solve such situations at the level of the graph model, since the choice of the channel depends on many situations, including memory access from other devices that are not considered in the framework of the graph model. Therefore, it is proposed to generate a transition in a graph model, i.e., create a separate gi ∈ G subgraph for each channel. When comparing the results of the RTL model and the alphabet model, check which channel was selected and select the appropriate transition, if the state of the cache hierarchy after the non-deterministic transition ti ∈ T does not correspond to any vertex in subgraphs G, it is concluded that the cache memory does not work correctly, that is, ∃ gi , ti, where ti ∈ T, gi ⊆ G. The error criterion, in this case, is the state that does not correspond to the state in {g0, g1, ..., gn}, where n is the number of channels at the corresponding level of the cache hierarchy.

A test generation technique based on a graph model of the cache hierarchy of a multi-core structure with minimal time spent on generating test sequences, which allows simulating all possible functional states of the memory subsystem is shown in Fig. 3. Fig. 2 shows the graph model construction algorithm. Each test sequence is a transition (edge) in the graph model (for example red edge on Fig. 1) of the cache hierarchy of the multi-core structure H. Accordingly, to generate verification sequences, the graph model of the cache hierarchy G is bypassed - enumeration of all edges of the graph H.

The methodology for generating verification tests consists of generating corresponding meta-language instructions in a graph model. First, a standard sequence is generated, consisting of a metalanguage instruction that transfers the memory subsystem to the necessary state (without making all the transitions in the graph model to save time during modeling). Next, a sequence is created consisting of instructions corresponding to the checked transition in the graph model. The instructions that correspond to the edges in the set E are written in conditional metalanguage to go to a higher level of abstraction, which allows you to untie the generation of verification tests from a specific processor architecture or structure. It is only necessary to implement a translator of metalanguage to a specific assembler or environment commands. Such an approach provides the possibility of transferring the generator developed as part of this work to other projects, including those with a completely different architecture of computing cores.



```
read (address, size)
write (address, size)
read (address, …, size)
write (address, …, size)
  …………
```

```
ldd (a=0x0),r0.d
vld (a=0x0),v0
tl r0.l, (a=0x0)
std r0.d, (a=0x0)
vst v0, (a=0x0)
vldl (v0), v16.l
  …………
```

Fig. 3. Test generation technique.

The metalanguage contains two instructions for working with memory - read (read) and write (write). If necessary, within the framework of the architecture being tested, you can implement a translator in assembler or test environment commands with support for vector memory accesses. As was done as part of the current work to check the hierarchy of the cache memory of a multiprocessor SoC, consisting of several ARM, MIPS cores, and its implementation of VLIW DSP with a new architecture, it is a translator with support for no more than 16 vector memory accesses.

III. Conclusion

The approach, using the developed methods and algorithms for verifying the hierarchy of multi-port memory caches, allows increasing the completeness of the test coverage of the code of RTL models with minimal time spent on generating test sequences, as well as to model all possible functional states of the memory subsystem. Fig. 4 shows test coverage comparing with other approaches.

The reliability of the methods and algorithms developed as part of the study is confirmed by evaluation criteria, theoretical calculations and their good convergence with computer modeling using verified models of the cache hierarchy, as well as experimental results obtained using the test generation system for verifying the vector VLIW DSP processor with the new architecture.

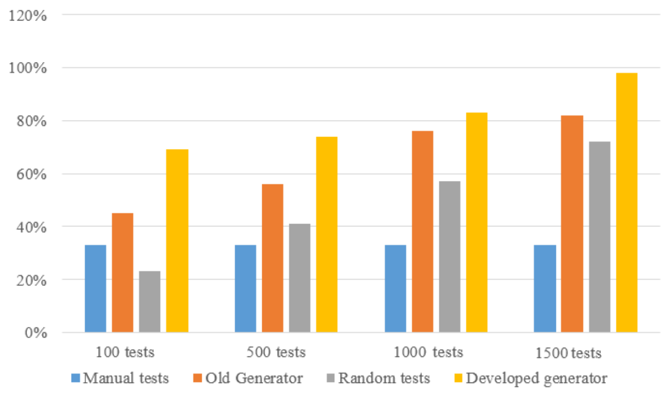


Fig. 4. Comparing with other approaches.

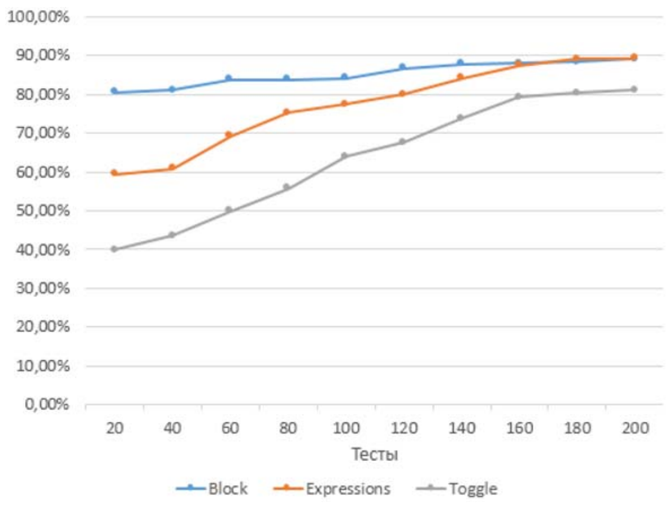


Fig. 5. Test coverage.

As a quality metric, a metric developed by Cadence was chosen, which is based on measuring the test coverage of code, blocks, expressions, and signals of the RTL model (Fig. 5).

The created software tool allowed:

- identify 12 architectural and functional errors in verifying the cache link of the developed vector VLIW DSP core with the new architecture, as well as find errors in the core itself and stand-alone cache memory environments;

- 30% reduction in the time needed to verify autonomous cache environments;
- significantly reduce the time of formation of test sequences, as well as quantities;
- reach 90% of the test coverage of the RTL model code.

In the future, it is planned to apply this approach to verify other processor cores.

References

[1] H. Zhao, X. Jia, and T. Watanabe, "Router-integrated cache hierarchy design for highly parallel computing in efficient CMP systems," *Electronics*, vol. 8, no. 11, arcticle 1363, Nov. 2019.

[2] A. Basak, X. Hu, S. Li, S.M. Oh, and Y. Xie, "Exploring core and cache hierarchy bottlenecks in graph processing workloads," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 197-200, 2018.

[3] R. Garibotti, L. Ost, A. Butko, R. Reis, A. Gamatie, and G. Sassatelli, "Exploiting memory allocations in clusterised many-core architectures," *IET Computers & Digital Techniques*, vol. 13, no. 4, pp. 302-311, July 2019.

[4] A. Garashchenko, A. Nikolaev, F. Putrya, S. Sardaryan, "System of combined specialized test generators for a new generation of VLIW DSP processors with Elcore50 architecture," *Problems of Developing Promising Micro- and Nanoelectronic Systems*, no. 2, pp. 9-15, 2018.

[5] A. Garashchenko, L. Gagarina, E. Fedotova, A. Vysochkin, and V. Zaitsev, "Development of a verification test generator for multi-nuclear structures," *Informatization and Communication*, no. 4, pp. 20-25, 2017.

[6] F. Putrya, "The use of random program generators and random background effects in the functional verification of multicore systems on a chip," in *Proc. 7th Int. Conf. Computer-aided Design of Discrete Systems*, 16-17 Nov. 2010, Minsk, Belarus, pp. 234-241.

[7] K. Gurin, A. Meshkov, A. Sergin, M. Yakusheva, "Memory architecture development in the «Elbrus» series computer models," *Radioelectronics Questions*, ser. EVT, no. 3, pp. 62-70, 2010.

[8] L.G. Gagarina, A.V. Garashchenko, A.P. Shiryaev, A.R. Fedorov, and E.G. Dorogova, "An approach to automatic test generation for verification of microprocessor cores," in *Proc. 2018 IEEE Conf. of Russian Young Researchers in Electrical and Electronic Engineering*, 29 Jan. – 1 Feb. 2018, Moscow, Russia, pp. 1490-1491.

[9] B. Greene, and M. McDaniel, "The Cortex-A15 verification story," *DVClub*, 7 Dec. 2011, Austin, USA, pp. 1-7.