FAST CONSTRUCTION OF TEST-PROGRAM GENERATORS FOR DIGITAL SIGNAL PROCESSORS

Shai Rubin, Moshe Levinger
IBM Research Division, Haifa Research Lab
rubin@haifa.vnet.ibm.com

Randall R. Pratt, William P. Moore
IBM Microelectronics Division, Essex Junction, Vt.
rrpratt@vnet.ibm.com

Abstract

Test-program generators play a key role in hardware functional verification of large scale processors. However, in the DSP domain, the usage of full-blown test-program generators is much less popular, mainly due to the limited resources (time and money) available when developing such systems. This paper describes a work-model for the fast, low cost construction of a test-program generator for DSPs. The core technology uses Genesys, a known test program generator that, until now, has been used for the verification of large scale processor families, such as PowerPC and x86. We developed the model while using Genesys for verification of the IBM C54XDSP, a recently-announced fixed-point DSP. The case study shows that it is possible to build a full test-program generator in a very short time and thus achieve better verification coverage in spite of the shorter development time.

1. INTRODUCTION

The goal of processor verification is to ensure equivalence of a processor and its architectural specification. In practice, processor verification is carried out by simulating a relatively small subset of selected test programs. These programs are run through the design simulation model and the results are compared with the output predicted by the architecture simulation model. Automatic test-program generators are used in order to produce massive and qualitative test-program subsets [1].

Usually, when developing a new processor ones implements its own architecture-specific test program generator. The Genesys system takes a different approach. It is a generic, architecture independent test-program generation system [2]. A formal model - the Genesys knowledge base - lies at the heart of the system. It allows for the specification of most of the architectural components which may be found in target processor architectures. As a result, the system can be used as a test-program generator for a wide range of processors and architectures.

Genesys was originally devised to cope with very complex, large-scale, processor systems [1,3]. It has a variety of capabilities targeted at verifying complex mechanisms, such as Memory Management Units (MMU - virtual memory), cache protocols and hierarchies, multi-processor configurations, etc. During the last six years, Genesys has been widely used on many IBM products all over the world and was recognized as a key tool for hardware verification by a non-IBM customers like SGS-Thomson [4].

In this paper, we show that the use of the Genesys system as a test-generator can be extended to many other processors and architectures, and in particular, to less complex ones, such as DSPs. We propose a work-model that is particularly useful and suitable for fast construction of test-program generators for this family of simpler processors. The proposed work model calls for the use of already available technology and tools instead of investing time and money building specific tools.

The paper is organized as follows: Section 2 illustrates the main characteristics which render Genesys and our work-model a feasible and effective solution for the DSP domain. Section 3 presents the infra-structure of the Genesys system and its main features. Section 4 proposes the process required for constructing a test-program generator based on the Genesys system. Section 5 describes a case study where a Genesys system was constructed for the IBM C54XDSP[6]. Section 6 concludes the paper.

2. REQUIREMENTS FOR A TEST PROGRAM GENERATOR FOR DSPs

In opposite to large scale processor families (e.g., IBM's PowerPC Intel's x86), DSPs are characterized with short development schedule and the low price per unit [5]. These impose two fundamental requirements on an automatic test-program generator:

- The development period required for constructing a test-program generator must be very short in order to meet the verification demands of a relatively short design schedule.
- The cost of developing an automatic generator for a DSP project should not be too expensive.

Thus, for a test-program generator to become the main verification tool in a DSP project, it must successfully meet the cost and schedule requirements. The Genesys system and our proposed work-model lend themselves well to this task. Firstly, Genesys is a mature platform with a high level of architecture expressiveness. This implies that Genesys can easily encompass new architectures using its current modeling power. Secondly, the work-model proposed in this paper allows the user to play a significant part in the construction of the test-program generator. This reduces the cost of building the generator system and enables users to add their own testing expertise to the tool. Furthermore, Genesys is a generic tool that after the first investment (cost and system foundation), can be used to verify every processor a company might develop in the future. This fact reduce even more the cost of test-generators for the next processor generations.

3. THE GENESYS SYSTEM

The Genesys system consists of the following four basic interacting components (Fig. 1):

a. The engine of the system - a generic, architecture oblivious, *test-program generator* [1,2].
b. A *knowledge base* - an external specification which holds a formal and declarative representation of the targeted architecture and a procedural description of the *testing knowledge* required for its verification.
c. An *architecture (behavioral) simulator* used to predict the

results of the instruction execution.

d. A Graphical *User Interface* (GUI).

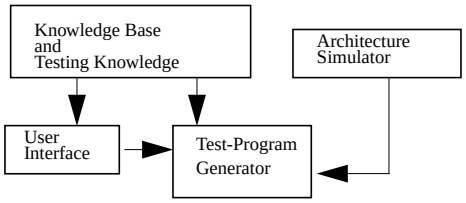

Figure 1. System Components and Interrelation

The system supports two main usage modes: G*lobal Generation* and *Specific Generation*. Global Generation directives apply to the instructions selected for the test and enable the user to direct the generation process towards interesting areas in order to exercise different architecture and implementation mechanisms (e.g., memory accesses, exceptions, etc.). Specific Generation mode allows the user to specify highly delicate test scenarios, where many constraints are placed on the generation process.

The task of the generator is twofold: first, to generate a test-program that meets all the predefined constraints; second, to complete the specific scenario generation using its biased-random capabilities whenever something was not specified.

Several of the major features and capabilities available in the generator are listed below:

1. Instruction Stream - The user can select the set of instructions to be used in the generated test-program.

2. Exception Control - The user may control the frequency of each exception type in the test.

3. Resource Sharing - Using this feature, the user can direct the generator to generate instruction sequences which (re-)use the same resources intensively (i.e., registers, memory locations, etc.) within a small “window” in the instruction stream.

4. Data-Types - The user may invoke testing knowledge procedures (generation functions) which will affect the data selected for the operands of the instructions generated in the test.

5. Macros - The user can define test patterns that will invoke specific parameters during the generation process.

6. Loops - The user may define test scenarios which include various types of loops.

7. Value Enumeration - The user may ask the generator to enumerate all of the relevant values for a given test entity (for example, enumerate all operand data, enumerate possible exceptions, etc.).

3.1 The Genesys Knowledge Base

The Genesys knowledge base is a hierarchical database representing the architectural details and the specific testing knowledge of the design. The first part of the knowledge base contains a description of the architecture. The major classes represent the main processor constituents, such as: Instructions, Operands, Formats, Fields, Exceptions and Registers. For each of these classes, there exists a template of attributes which allows an accurate definition of the desired object. The entire specification of the architecture is done in a declarative fashion. This leads to a simple scheme for knowledge base population and facilitates the construction of new Genesys systems.

The second part of the knowledge base contains the testing knowledge. This part enables the expert user to add testing expertise to the architectural model. This specific knowledge, either architectural or implementation-dependent, is used by the generator during the instruction generation process. An example of such knowledge is the infinity value for floating point operations. Special functions can be easily added into the knowledge base in order to ensure generation of such values for the various Floating Point instructions. A detailed description of the knowledge-base can be found at [1,2,4].

From the above discussion, the reader should note the following main observation:

The power of Genesys is inherently found in two main components: the generic engine and the knowledge base. However, the generic engine can supply extensive verification capabilities even when working with a minimal knowledge base, i.e., one that contains only a few instructions and no testing knowledge.

This fact enables us to rapidly build a verification environment for DSPs. As soon as the designer has a knowledge base with only few instructions, verification can begin. During the verification process, the designer may continue to add more instructions and testing expertise into the tool. This leads to the suggested work-model described in the next section.

4. THE WORK-MODEL

The final goal of the work-model is to build a complete verification environment (Figure 2). Due to the short period allocated to the development effort, the process has to be incremental, fast and productive from the very first stages.

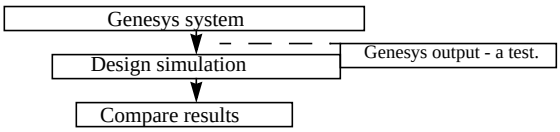

Figure 2. Genesys-based Verification Environment

The main purpose of the work-model is to allow designers to use partial versions of the generator as early as possible. As a result, the verification process may begin even before all stages have been completed. The verification group’s heavy involvement in the Genesys construction process not only contributes to the rapid construction effort, but also develops expertise in the tool as well as in the design’s architecture, thus increasing the quality of the system.

The first step in the process is to build an operational system that generates legal tests for the design. The definition of a legal test, in this case, is a test-program that is completely consistent with the architecture’s specification. Although the tests generated at this time are mostly random, all the internal power of Genesys is already available. The following six stages describe this part of the process (stages 3-5 may be done in parallel).

1. *Architecture study.* The Genesys expert becomes familiar with the essential parts of the architecture. The instruction set is divided

into two sets: Set A and Set B. Set A contains instructions whose modeling is straightforward; they can be generated in a fully random manner (similar to arithmetic instructions). Set B consists of instructions that are more tailored to the specified architecture and in some cases these instructions will be generated only in user-supervision mode.

2. *Building the initial (NOP) system*. In this step, all system components are established. A basic knowledge base is created and any changes necessary are made to the Genesys engine. This stage is done using a NOP-simulator - a simple simulator that fully supports the Genesys requirements, but treats every instruction as a NOP. Using this kind of simulator enables the team to build a complete system, even in cases where the behavioral simulator is not ready for use at this early point in the design. The initial system serves two main purposes:

- Learning - The verification team starts learning how to use Genesys and what it is capable of.
- Beginning integration - The first steps towards integrating the Genesys system into the whole verification environment are performed. For example, establishing the link between Genesys and the design simulation model (usually VHDL/Verilog), writing a preliminary version of the API between Genesys and the architecture's behavioral simulator, etc.

3. *Detailed design of Set A instructions*. The design describes the architecture in a way that is consistent with the knowledge base hierarchy. This stage requires an in-depth understanding of the architecture and is carried out by both the verification and Genesys teams.

4. *Integration with the behavioral simulator.* After constructing the initial (NOP) system, the team integrates the behavioral simulator with Genesys. The work-model assumes that a functional behavioral simulator is already available to the design project for software development purposes. In this case, only the simulator API required by Genesys needs to be implemented.

5. *Establishing the Genesys-based simulation environment*. Some work is usually required in order to convert Genesys outputs (the test cases) to the inputs of the design model. By using the test cases produced by the initial system, the verification team is now ready to integrate Genesys into the verification environment.

After completing steps 1-5 (in the case study, these stages took about a month), the first random test may be generated and fully simulated by the design model. Although the system consists of very few instructions, all the inherent capabilities of Genesys can be used in these tests. For example, full addressing-mode support, interdependency between instructions, long and random tests, etc.

6. *Testing Knowledge.* After the population of Sets A and B is complete, and the simulation environment has been implemented, the extension of the Genesys knowledge-base through the use of Testing Knowledge (TK) becomes an on-going process. Applying TK to the system helps focus the testing onto critical, sensitive areas, and prevents the generator from testing invalid cases, e.g, division by 0, if that is not supported by the architecture. Addition of TK continues through the completion of the verification plan.

The expertise developed in all aspects of Genesys can be carried effectively from project to project and serves to streamline subsequent applications of Genesys.

Figure 3 presents the complete process and all the stages required to establish the full-blown test generator.

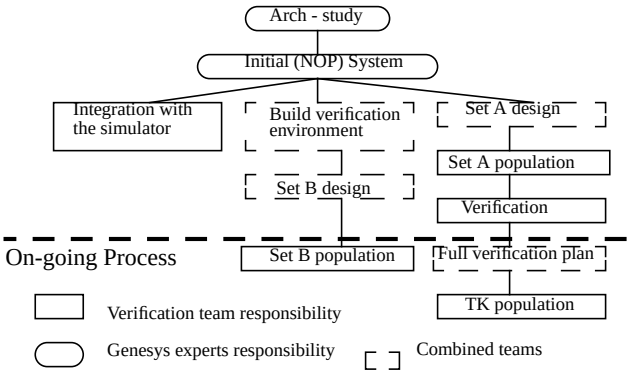

Figure 3. Actions and People Involved in the Construction of the Generator

5. CASE STUDY: BUILDING A GENESYS SYSTEM FOR THE IBM C54XDSP

This section describes the results when our suggested work-model was applied to a general-purpose DSP, the IBM C54XDSP[6]. The end of this section describes how Genesys was used to implement the verification plan.

The C54XDSP is fully compatible with the well-known C54x Texas Instruments DSP family. The C54x architecture includes a highly specialized instruction set of 187 instructions. The processor contains no cache nor MMU, but does include three separate memory spaces for instructions, data, and I/O, as well as multiple buses.

5.1 Constructing the Genesys System - Results

The entire test-program construction process took 3-4 person months (Table 1). The final output of the process is a Genesys system which can generate random tests on over 90% of the instruction set. Tests for the rest of the instruction set can still be generated by the system, but only within a specific generation mode, since control and supervision by the user are required.

Stage	Time	Remarks
Arch - study	1 week	Set A includes 150 instructions Set B includes 37 instructions
NOP-and initial system	1 week	100 lines of C code
Set A design	1 week	150 instructions with 73 different formats
Construction of verification environment	1 week	Done at the verification site through the combined effort of the Genesys people and the verification team
Set A population	1 month	The verification itself can start immediately after populating the first instructions
Set B design	1 week	

Stage	Time	Remarks
Set B population	1 week	Requires close guidance by Genesys experts
Full verification plan	2 months	For certain instructions and functions, much of this can be done in parallel with other stages, as the instructions mature.
Testing Knowledge	On-going	This extends throughout the life of the verification plan.

Table 1. Stages and figures for the building process of the Genesys-*IBM C54XDSP*

5.2 Implementation of the Preliminary Verification Plan

The purpose of the preliminary verification plan is to verify that the basic parts of the design and the verification methodology work properly. Basic parts are considered to be the opcodes (that are currently supported in the design) and the address modes.

The application of Genesys to verification of the C54XDSP began after parts of the design were already implemented and partially verified. Thus, the Group 1 tests below were omitted. The Group 2 tests were the primary debug vehicle (preliminary verification) of the Genesys database, test generation, and verification environment. At this time, architectural verification of the design was productive in finding design errors.

The following groups of tests for the preliminary verification plan were defined as described below:

- Group 1: For each instruction, one instance of the instruction was used per test. This would validate the database and the model. As discussed above, this group of tests was skipped.
- Group 2: For each instruction, ten instances of the instruction were used, separated by 3 NOPs (no-op instructions). NOPs were used to avoid pipeline and latency issues. There were 477[1] different tests generated.
- Group 3: For each instruction that used the indirect addressing mode, one instance of the instruction was used for each of the valid (between 12 and 16) addressing modes. There were 254 tests generated.
- Group 4: For each instruction that used the #lk (long constant) form of indirect addressing, one instance of each #lk addressing format was used, for each of the 8 possible wait state values on the external bus. There were 170 tests generated.

The latter group is an example of the Genesys system's ability to take a problem encountered in a particular situation and quickly generate a group of tests that cover all the relevant instructions and associated modes. In the above group, the problem addressed was the handling of the #lk operand in the pipeline, as affected by the wait states.

5.3 Further Verification Using Genesys

Beyond the rapid involvement of Genesys in the initial verification of the architecture for each instruction and addressing mode, as described above, the power and responsiveness of the Genesys system can also be applied to the following areas:

- Data Dependency tests - Using Genesys' random generation of the operands, coupled with the appropriate testing knowledge, both boundary conditions in the architecture, as well as unsuspected data sensitivities in the design can be tested.
- Random instruction streams - These types of tests have been very effective in detecting subtle bugs, especially in the areas of stalls. While there are inherent restrictions in the architecture on adjacent pairs of instructions, this type of (often intelligently directed, but also random) testing has proven quite beneficial.

5.4 Results

Genesys tests uncovered approximately 100 design errors in the design, in spite of having started after many opcodes and functions had already been verified. A total of 1814 tests were generated for the Groups discussed above. These tests are a key part of the static regression suite. A total of more than 75,000 Data Dependency and Random tests have also been run to date. The rapid integration of Genesys into the Verification Plan was critical to the successful verification of the IBM C54XDSP.

6. CONCLUSION

In this paper we described a work-model and a six stage process for fast construction of test-program generators based on the Genesys platform. This framework can be highly effective for constructing a test-generator for DSPs. The proposed work-model enables the construction of a full-blown, test-program generator within the cost and development time requirements of a DSP design. This work-plan was used in the verification of the C54XDSP. The use of Genesys and the work-model enabled an efficient implementation of the preliminary verification plan in a very short time and help to find many bugs, some of which are considered hard to find. The results obtained from this case-study reaffirm our claim regarding the speed and cost of building test-generator systems for DSPs. Furthermore, the proposed system is a long term investment since it can be used for verification of various processors and their future generations.

7. REFERENCES

1. A. Aharon, Dave G, M. Levinger, Y. Lichtenstein, Y. Malca, C. Metzger, M. Molcho, and G. Shurek, "Test Program Generation for Functional Verification of PowerPc Processors in IBM" in *Proc. ACM/IEEE 32nd Design Automation Conference 1995.*

2. Y. Lichtenstein, Y. Malca, A Aharon, "Model Based Test Generation for Processor Verification" *IBM Technical Report 88.337 1993*.

3. L. Fournier, "Genesys-X86: An Automatic Test-Program Generator for X86 Microprocessors" *IBM Haifa Research Center VLSI Internal Publication.*

4. F. Casaubieilh, A. McIsaac, M. Benjamin, M. Bartley, F. Pogodalla, F. Rocheteau, M. Belhadj, J. Eggleton, G. Mas, G. Barret, C. Berthet, "Functional Verification Methodology of Chameleon Processor" *in Proc. ACM/IEEE 33rd Design Automation Conference 1996*.

5. "Microcontrollers that offer the power of choice" *Electronic Engineering April 1994.*

6. Texas Instruments, "TMS320C54x User's Guide Preliminary" *1995*.

1. The architecture defines 187 instructions. However, for simplicity reasons, in Genesys knowledge-base some of the original instructions were modeled as several instructions.