Scheduling-based Test-case Generation for Verification of Multimedia SoCs

Amir Nahir, Avi Ziv
IBM Research Lab, Haifa, Israel
{nahir, aziv}@il.ibm.com

Roy Emek
r_emek@yahoo.com

Tal Keidar, Nir Ronen
Zoran Microelectronics Ltd., Haifa, Israel
{tal.keidar, nir.ronen}@zoran.com

ABSTRACT

Multimedia SoCs are characterized by a main controller that directs the activity of several cores, each of which is in charge of a stage in the processing of a media stream. The verification of these SoCs is a significant challenge due to time-to-market constraints and system complexity. We present a novel approach to system-level, random test case generation for multimedia SoCs, and a tool, called SoCVer, that implements this approach. We use the SoC's main controller point of view for controlling the flow of data in the SoC. Test case generation is done by allocating processing tasks to the various cores and determining which core processes which data item at what time. Solving these scheduling problems allows SoCVer to generate software for the SoC's main controller; this software coordinates and synchronizes the operations of all the cores on the chip without the need for the real operational software. We demonstrate the use of SoCVer using a DVD player SoC.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design aids – *Verification*

General Terms: Verification.

Keywords: Functional Verification, System on a Chip, Test Generation

1. Introduction

During the last few years, complex hardware designs have shifted from custom ASICs toward SoC (system on a chip)-based designs, which include ready made components or 'cores'. Multimedia applications is one of the fields in which SoC-based design is dominant. Many consumer products of this type, including DVD recorders and players, and digital cameras, share a common base structure. This structure includes several cores, such as DSPs, encoders, and decoders, that communicate through shared memory (or memories) and a main microprocessor that controls and coordinates the entire system.

Verifying SoC-based designs for multimedia applications incorporates several challenges [2,7]. First and foremost is the need to verify the integration of several previously designed cores in a relatively short time. In addition, the system's embedded software is typically not fully written until fairly late in the development cycle. These challenges are compounded because while several cores can work concurrently in such systems, the system's functionality enforces temporal constraints on the order in which each of the cores carries out its tasks. Thus, stimuli generation that exercises these temporal constraints is one of the main challenges in the verification of such SoCs.

The main existing verification solutions do not provide an adequate solution to this challenge. Manually writing tests is a time consuming error-prone process. In addition, it is nearly impossible for a verification engineer to completely comprehend all the dependencies within the design under verification (DUV), and therefore, many aspects of the system are ignored or set to static values during the verification process. Traditional test bench authoring tools, such as Specman and Vera [4,6], focus on generating and driving stimuli to a complete system, although many challenges lie in the coordination and synchronization of internal components. Using the operational software and SW/HW verification is not always possible because the software is not always available in the early stages. In many cases, it is also hard to reach hardware corner cases with the operational software.

We developed SoCVer, a random test case generator aimed at coping with the above challenges and generating software for the system's main controller. A high-level depiction of SoCVer is shown in Figure 1. As input, SoCVer accepts an abstract model of the DUV, a test template that describes the content of the test case to be generated, and a high-level description of the system's input (i.e., the media stream to be processed).

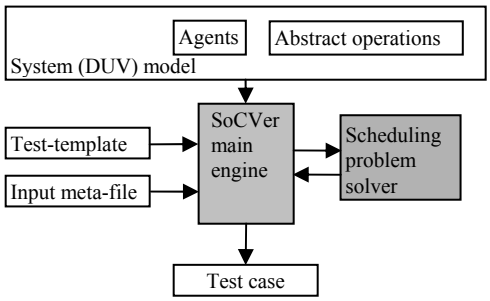

Figure 1: SoCVer high level description

The abstract model contains a description of the different cores (agents) in the system, the operations supported by the system, and the tasks the agents perform as part of the system's operations. The media stream description contains information about the types and order of the items comprising the stream. We refer to this as the input's meta-data, as it contains information about the media stream itself. Finally, the test templates determine the contents of the generated test case in terms of the operations performed by the DUV (e.g., "play – stop – play" for a DVD player SoC).

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

SoCVer transforms the system's operations comprising the test case, one-at-a-time, into a scheduling problem. This problem represents the times in which the system's agents process the different input items. Finally, the test case generated for the microprocessor implements the agents' actions as commands given at different times to the system's cores. These commands are, in fact, the driving element of the test, as opposed to traditional verification environments where the test is mainly driven by the system's input.

We experimented with SoCVer, focusing on the verification of a SoC-based design for a DVD Player. At the same time, we proved the feasibility and compatibility of this methodology with several other SoCs.

The rest of the paper is structured as follows: Section 2 describes a DVD player SoC that is later used as a running example. Section 3 presents our proposed solution along with its application to the DVD player SoC example. Conclusions and directions for future work are presented in Section 4.

2. DVD Player SoC Example

We chose to use a DVD player SoC to demonstrate SoCVer's practice and capabilities. Our example focuses on the 'Play' operation over an MPEG-2 video stream. MPEG-2 is a standard for encoding moving pictures and associated audio [8]. The standard defines three possible ways to encode a single picture or frame: Intraframes (denoted 'I'), Predicted frames (denoted 'P'), and Bidirectional frames (denoted 'B'). An MPEG-2 video stream is a sequence of 'I', 'P' and 'B' frames.

The DVD player's main tasks are to read the data from storage, decode the MPEG-2 encoded movie, and turn it into a standard composite video signal. In addition to decoding the movie, the DVD player decodes the movie's soundtrack and subtitles (if requested). The DVD player SoC is depicted in Figure 2.

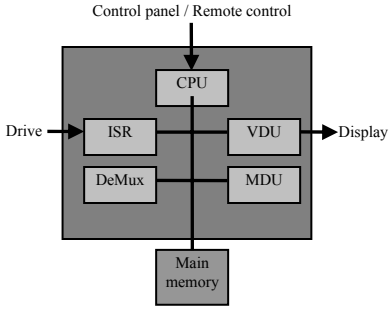

Figure 2: DVD Player SoC structural block diagram

The DVD player's actions are coordinated by a central controller that implements user actions in software as a sequence of commands issued to the various modules.

The DVD player is composed of several internal modules ('cores'). The Input Stream Reader (ISR) reads the data stream from the DVD drive or hard drive. The Demultiplexer (DeMux) breaks the stream into video images, audio track, and subtitles. The MPEG Decoder Unit (MDU) decodes the video images and the Video Display Unit (VDU) prepares the images for display. The VDU converts each video frame into two fields, containing the image in the format in which it is displayed. In addition, whenever the VDU has no new fields to display, the controller instructs the VDU to display the last two fields again, causing the image in the viewer's screen to freeze.

Each of the processed items (frames, fields) is stored in a main memory module. To get an item processed by one of the modules, the controller first sets the module to the required processing mode. The module then reads the item from the memory, processes it, and stores it back in the memory. Figure 3 depicts the data flow within the DVD player when playing an MPEG-2 video stream.

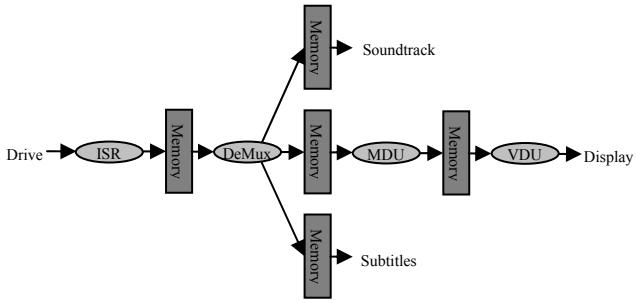

Figure 3: 'Play MPEG-2' data flow within the DVD Player

3. Proposed Solution

SoCVer is a random test case generator for multimedia SoCs. The main goal of SoCVer is to automatically generate "software" for the DUV. By incorporating expert knowledge, it helps improve coverage and increase the probability of hitting corner cases.

SoCVer is designed to address the drawbacks of the verification techniques presented in Section 1. The automatic generation of test cases reduces the time verification engineers need to create a given scenario. In addition, using the main controller's view point, SoCVer can coordinate and synchronize the stimuli to the peripheral cores of the DUV in a way that enables the creation of the requested scenarios. Finally, since SoCVer generates the software for the DUV's main controller, it does not rely on the existence of software for the verification of the SoC.

A high-level depiction of SoCVer is shown in Figure 1. SoCVer is comprised of two main components: an abstract model of the DUV and a generic generation engine. The DUV's model contains information about the main cores of the DUV and their operation. The model also contains expert knowledge used by the test generator to improve the quality of the generated test cases. The generation engine uses the DUV's model to convert the test template provided by the user into a set of scheduling problems. A scheduling problem solver is used to solve these problems and generate a test case in the form of software for the main controller. In this section, we describe SoCVer, along with an example based on our verification of a DVD player SoC.

3.1 System Model

To generate quality test cases, the test-generator must have a profound understanding of the DUV. To achieve this, we designed SoCVer as a model-based [1] test case generator. SoCVer provides a framework for modeling, which includes the following basic concepts:

Agents – Agents are the components ('cores') from which the system is comprised. Each agent has a set of resources (e.g.,

registers) indicating its internal state. In our example, the agents are the ISR, DeMux, MDU and VDU. The VDU has an internal state property indicating whether subtitles should be displayed.

Time – All the system's agents are synchronized with a single, system-level time-line. The unit—or "tick"—of this time-line represents a single quantum of processing time (e.g., a field processing time by a single agent in a DVD player). Media processing times are integer quantities of time-ticks. Note that this time-line is not directly tied to any of the DUV's hardware clocks.

Shared Work Spaces – These are segments of memory where data is stored. In the DVD player SoC, a shared work space is used, for example, to store frames after their decode stage, but before they are fed into the VDU.

Abstract Operations (AOPs) – AOPs represent the system's functionality (e.g., 'Play', 'Pause', etc.). The data flow within the system is modeled as a directed acyclic graph (DAG). Each node of the data flow's DAG designates one of the DUV's resources (agents or shared work spaces). Each DAG node has attributes that model how the resource designated by the node processes input items. Examples for such attributes are the duration of time it takes an agent to process an item and the rules it applies to input items. DAG arcs are used to designate data transition, that is, agents reading from and writing to shared work spaces, as shown in Figure 3.

Rules – Rules can be used to express unique behavior of the DUV. Such behavior can include alteration of the input items' format, permuting the order of input items, or unique agent operation. In our example, the MPEG-2 frames are stored in 'IPBB-PBB-PBB…' order, but are displayed in 'IBBP-BBP-BBP…' order. This change of order can be expressed by modeling an appropriate rule and indicating a shared work space over which this rule applies.

Input Modeling – Since the system's operation is affected by the properties of its input, we need to model the input items and input streams (a collection of input items). The values given to the input properties in the input meta-file may affect the generated test. For example, the input item size affects the number of items that can reside simultaneously within a given shared work space.

3.2 Generation Scheme

SoCVer's generation scheme follows a 'One-AOP-at-a-time' concept, where each AOP in the test template is handled individually. A single AOP's generation scheme can be partitioned into two stages: (1) task calculation, and (2) constructing and solving a scheduling problem. The solution of scheduling problems for all the AOPs in the test template yields an abstract test. Once the abstract test is complete, a third stage of refinement is performed. Applying the refinement stage to the abstract test produces a concrete test case that can be used in simulation. In this section, we provide a more detailed description of the three different stages of the generation scheme.

3.2.1 Task Calculation Stage

A task is either the processing of an input item by one of the DUV's agents or the storing of an input item in a shared work space. The goal of the task calculation phase is to create, for each of the DUV's agents and shared workspaces, a set of tasks that could potentially be performed by it during the execution of the AOP. For this purpose, we distinguish between two types of tasks: mandatory tasks and optional tasks. Mandatory tasks are tasks that we are sure will be performed during the currently processed AOP. Optional tasks are tasks that could potentially be performed during the execution of the AOP, but we cannot determine whether they'll be performed at all at this stage. For example, when executing a 'Play MPEG-2' AOP over an 'IPBB' stream, we know the DeMux will process the P-Frame, hence there will be a mandatory task denoting the DeMux processes the P-Frame; however, we do not know how many times the VDU will process the first P-Field, so there will be several optional tasks denoting the VDU processing of this P-Field.

We calculate the tasks by propagating input items through the AOP's modeled data flow according to the information given in the input's meta-file. We apply the related modeled rules at each of the data flow's nodes. If the rules imply that processing the current AOP requires some reference to tasks performed during the execution of previous AOPs, we retrieve old tasks and use them to create new ones.

3.2.2 Constructing and Solving a Scheduling Problem

The goal of this stage is to schedule the execution of all tasks. For mandatory tasks, this means setting the time the task starts and ends; in optional tasks this also means determining whether execution of the task is required. In addition, the location of storage tasks within a shared work space is determined.

Based on the results of the task calculation stage, a scheduling problem is constructed, consisting of variables and the constraints over them. We use variables primarily to represent the time-ticks relevant to the task. Additional variables mark the segment where an item might be stored within a shared work space. Constraints express the relations derived from the AOP's data flow:

- AOP start-time constraints – The entry-time of the first mandatory task to any of the root nodes in the data flow's DAG is constrained to be greater than the end-time of the previously executed AOP in the designated agent. Note that the execution of an AOP does not end at the same time-tick for all agents.
- Processing-time constraints – These express the duration required for an agent to process a given data item by constraining the data item's exit-time from the agent to be equal to its entry-time plus the required processing time. No constraint is placed on the duration of time a data item remains in a shared work space, other than requiring that its exit-time be greater than its entry-time.
- Node internal order constraints – These express the order in which mandatory tasks are executed in a node by constraining the exit-time and entry-time of sequential data items.
- Agent-writes constraints – A data flow arc from an agent node to a shared work space node implies the agent writes to the shared work space. We reduce this link to a constraint indicating the storage task in the shared work space begins exactly when the corresponding processing task in the agent completes.
- Agent-reads constraints – A data flow arc from a shared work space to an agent node implies the agent reads from the shared work space. We reduce this link to two constraints, one indicating the agent can read the input item from the shared work space only after the item has been written to the

shared work space, and the other indicating the input item remains in the shared work space as long as it is needed by the reading agent.

- Mutual exclusion constraints – These are used to ensure that no two input items are stored in the same shared work space segment at the same time, and that all agents are limited to performing a single task at a given time.
- Optional tasks specific constraints – The same rules used to direct the creation of optional tasks are also used to clarify the way those tasks are related to other tasks (both optional and mandatory)
- Other constraints – Users can specify additional constraints as part of the DUV's abstract model or the test template. For example, users can add a constraint requiring at least three time-ticks between the entry-time of an input item to the MDU and its entry-time to the VDU.

After formulation, an in-house solver, running a variant of the maintaining arc consistency (MAC) algorithm [3,5] is used to solve the scheduling problem.

To increase the quality of the generated test cases, expert knowledge of the DUV can be incorporated into the system model in the form of non-mandatory ('soft') constraints. For example, giving priority to an odd number of time-ticks between the entry-time of an item to the MDU and its entry-time to the VDU.

Figure 4 depicts a solution to the 'Play MPEG-2' over an 'IPBB' stream scheduling problem. On top of the basic AOP, the test template contains a directive that emulates a slow down in the rate frames arrive to the ISR. This could represent a problem in the DVD reader. The input stream flows from the ISR, through the DeMux to the MDU, where each unit processes the frames whenever they become available. The handling by the VDU is more complex. In addition to the fact that the VDU processes fields, there are other elements that affect the processing by the VDU. First, the VDU starts handling the data in the eighth time tick because of a start-at-an-even-tick testing knowledge directive. Second, the order in which the VDU processes the fields is different from their arrival order because of the MPEG-2 reordering rule. Finally, the VDU is required to repeat some of the fields, because of the slow rate of data arrival. When formulated as a CSP, such a scheduling problem consists of about 500 variables and over 3000 constraints, and takes the solver about one minute to solve.

3.2.3 Refinement Stage

To create an executable test case, all tasks performed at a given time-tick are implemented as a set of commands issued by the controller to the system's modules at that time. In our example, at time-tick 5, the controller issues the following commands: to the MDU—read and process the 'I' frame from segment 12; to the DeMux— read and demultiplex the 'P' frame from segment 9; and to the ISR—read the first 'B' frame from segment 3.

Some of the controller's commands are more complicated and require additional parameters. Issuing some commands might require changing one of the core's state properties. A new CSP is formulated to determine values for all the parameters and state properties required for the execution of a given command. This CSP consists of variables for parameters, state properties and input item properties (as required for the specific command), and constraints over them. Constraints can originate in the DUV's model or the test template. For example, when the VDU processes an I-Field, the controller might choose to display only part of the image on the viewer's screen. The CSP created for this command is made up of variables for the VDU's zoom property and screen resolution property, the 'I'-Field's size property and the read command's fragment size parameter. The fragment's size is constrained to ensure that multiplying the size by the zoom factor will result in the screen resolution. Based on the results of the CSP, the controller reconfigures the VDU's state properties and instructs the VDU to read the required fragment of the 'I'-Field using the read command.

Once generated, the test cases produced by SoCVer are used as the controller software in simulation. Checking for DUV proper behavior is done by comparing the simulation results with those achieved by running the test cases on a reference model.

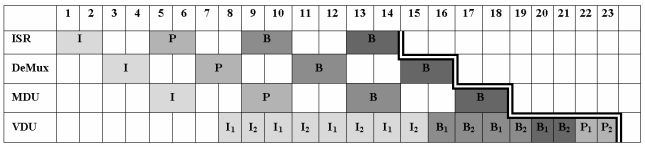

Figure 4: 'Play MPEG-2' over an 'IPBB' stream scheduling problem solution

4. Conclusions

We presented a novel approach to system-level test-case generation for multimedia SoCs, based on modeling and solving scheduling problems. The main benefit of this scheduling-based approach is the automation of test case creation for these systems. SoCVer, a tool implementing this approach, is designed in a model based fashion, which requires less effort to bring up. SoCVer is currently in stages of deployment as the lead solution for test case generation in the verification of a DVD player design. Based on an abstract model of the DUV, SoCVer generates test cases implementing a test template, which are used as software for the player's microcontroller. We are currently working to improve SoCVer's integration with the verification environment. Future work will include adding checking capabilities.

5. References

[1] Aharon, A., Goodman, D., Levinger, M., Lichtenstein, Y., Malka Y., Metzger, C., Molcho M., and Shurek, G. Test program generation for functional verification of PowerPC processors in IBM. In 32nd Design Automation Conference, pages 279-285, 1995.

[2] Bergeron, J. Writing Testbenches: Functional Verification of HDL Models. Kluwer Academic Publishers, January 2000.

[3] Bin, E., Emek, R. Shurek, G., and Ziv, A. Using constraint satisfaction formulation and solution techniques for random test program generation. IBM Systems Journal, 41(3):386-402, 2002.

[4] Haque, F., Michelson, J. and Khan, K. The Art of Verification with Vera, Verification Central, 2001.

[5] Mackworth, A. Consistency in Networks of Relations. Artificial Intelligence, 8(1):99 – 118, 1977.

[6] Palnitkar, S., Design Verification with e, Prentice Hall, 2003.

[7] Wile, B., Goss, J. C. and Roesner, W. Comprehensive Functional Verification – The Complete Industry Cycle, Elsevier, 2005.

[8] ISO/IEC 13818-1: Generic coding of moving pictures and associated audio information.