Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 111 (2005) 73–91
www.elsevier.com/locate/entcs

Lightweight Specification-based Testing of Memory Cards: A Case Study

Seung Mo Cho and Jae Wook Lee[1,2]

Mobile Solution Lab., Software Center, Corporate Technology Operation
Samsung Electronics Co.
Shinsa-Dong, Kangnam-Ku, Seoul, Korea

Abstract

As the markets of mobile devices are expanding, needs for developing reliable memory cards are increasing, too. Samsung, one of the major players in memory card business, is also trying to improve the validation process for their memory card products. To this aim, we conducted a pilot project where a formal method and a specification-based testing technique are adopted to validate our MMC (MultiMediaCard) system. System under testing (SUT) is an MMC card which is implemented in two languages, Verilog for RTL and C for firmware. To test MMC cards, we formalize the fully general behavior model of MMC host with Esterel. It is also used as a test oracle in order to automate testing of SUT. Then, the two models of host and card are co-simulated on the verification environment Seamless. We conducted scenario-based testing and random testing.

Keywords: MultiMedia Cards, Specification-based Testing, Esterel, Seamless

1 Introduction

Our modern life heavily depends on many systems that are carrying computers in them. Cellphone is one of the most obvious, and also the most complex, examples. The source code for a cellphone has the size of more than 1 million lines. There are many other examples, such as electric ovens, traffic light controllers, power plant management systems, and car-navigation systems, with various size and complexity. These systems are called *embedded systems*.

[1] Email: seungm.cho@samsung.com
[2] Email: rtos21@samsung.com

doi:10.1016/j.entcs.2004.12.008

External memory cards are also embedded systems. They have their own micro-processors, internal buses, and of course, flash memories. The demand for external memory cards is continuously increasing as the markets for various mobile devices are expanding.

There are a number of industrial standards for memory cards, like SD card, memory stick, and CF card. MultiMedia Card(MMC)[9] is also one of the specification for memory cards. It has been adopted by many vendors, including Samsung, in the fields of PDA, digital camera, camcorder, and MP3 player.

When Samsung started to produce MMC cards, people considered performance the most important issue. However, it has soon become obvious that reliability is also crucial to the success in the market. A man from the marketing division expressed this like the following: "We can sell low-performance products - by cutting the price, but we can not sell them if they crash when plugged."

In this paper, we present a case study we conducted to improve the validation process for MMC cards. We proposed a new method for specification-based testing of memory cards, and applied it to our MMC cards.

Memory card systems form a subcategory of embedded systems. They are also communication systems in that they interface with the host such as PDA, cellphone, and digital camera. Therefore, the specification of memory card systems usually consists of two parts: the low-level hardware requirements specification and the communication protocol specification between host and card. They have a number of unique features that distinguish them from either typical communication systems or embedded systems.

- Generally, they operate on a special-purpose hardware. Since hardware and software are co-designed, design decisions of hardware affect heavily on the design of software and vice versa. Moreover, both hardware and software have complex application logics. It is not general for embedded systems, where off-the-shelf real-time OS is used to reduce the direct functional dependencies between software and hardware.
- Since, embedded systems often tend to a large harm to safety or finances, they are usually developed with great care and many resources. However, memory cards are not considered so critical to safety or finances. Moreover, there are severe market needs to ship relatively reliable products, quickly.
- Related to the above reasons, architectures of memory card systems tend to be poorly layered. For example, OSI 7-layer reference model is accepted as the standard architecture for distributed network systems. Thus, diversity of vendors providing each layer causes little problem because their roles are

well-defined. On the other hand, the specification of MMC defines only the physical-level, bit-wise, and communication protocol; different vendors of hosts and cards use different separation of functionalities among hardware, firmware, and device drivers.

According to these features, we think it is better to design our own specification based testing method rather than to adopt existing one. We build a *cycle-accurate* model of MMC host which shows fully general behaviors. It is built on the basis of the official MMC specification only. It also enables us to encompass a *test oracle* in the host model. MMC specification restricts the permitted behavior of MMC cards. The host model is also used as a test oracle because it keeps up the state of MMC card.

This paper present a method, and the result of our pilot project conducted. The contribution of this paper can be summaried as follows.

First, we set-up a new lightweight[6] specification-based testing method, suitable to memory cards. Our approach builds a formal behavior model of the *environment* and derives a test harness from it. Most specification-based testing methods are tailored for communication systems (eg. [7]) or safety-critical embedded systems (eg. [10]). We found that distinguishing features of memory cards make our approach more attractive than the existing specification-based testing methods.

Second, we formalized the environment part of MMC specification. The specification, which seems trivial at first glance, was found to have several complex requirements on the host and the card. We formalized the cycle-accurate behavioral model of MMC host using Esterel[1]. During the formalizing process, we could precisely understand some ambiguities present in the specification, and also found some violations with the specification.

Third, we describe the test results of our MMC card. Testing is done on a commercial platform, Seamless, a hardware/software co-simulation tool[11]. Due to the speed problem and reproducibility of errors, actual testing is done on an emulation environment. Test harness code produced from the formal model is combined with the code of MMC card for hardware (written in Verilog) and software (written in C). We will show the effectiveness of the experiment compared to that of brute-force random testing.

This paper is organized as follows. In Section 2, we present the background of our work. We introduce MMC specification and Esterel. We present our framework with the rationales in Section 3. In Section 4, we briefly explain our modeling method of MMC host. The testing result will be given in Section 5. Finally, we conclude the paper with some discussion and futurework.

2 Background

2.1 MultiMediaCard

An MMC system consists of an MMC*host* and MMC *cards*. Host is a device that uses the external memory cards, such as digital camera, PDA, or cell-phone. They are connected with three-wire serial data bus (*clock, command* and *data* lines). Communication between them is performed using messages, which is represented by one of the following tokens.

- a *command* is a token which starts an operation. A command is sent from the host to MMC cards and transferred serially on the command line.
- a *response* is a token which is sent from MMC cards to the host as the answer to the previously received command. It is transferred serially on the command line.
- *data* is transferred from the card to the host or vice versa via the data line.

The main functionality of an MMC card is to read and write data. Data is transferred in two ways: stream-oriented mode, where the host should explicitly stop a stream of data transfer, and block-oriented mode, where data transfer is done in terms of a block whose size is defined before the transfer. We consider block-oriented mode here, because many MMC cards, including the one made by Samsung, support only block-oriented transfer. Besides reading and writing commands, there are other additional commands, such as stop-transfer command and set-block-count command.

Like most industrial specifications, MMC specification is written mostly in English. It also includes a number of diagrams and tables, which provide much help to understand the requirements. A state diagram like Figure 1 is used to show the state transition in MMC cards, and a timing diagram like Figure 2 is used to describe the timing requirements. However, these diagrams are not sufficient to fully resolve the ambiguity or incompleteness because they are not used to *define* the requirements, but to *help* the understanding of the requirements. For example, there are many diagrams showing the timing relations between tokens on the serial bus. However, it is obvious that only small part of all possible scenarios is covered.

MMC specification identifies 11 states which a card can stay during the operation. At each state, the card accepts a command token from the host, sends back a response token if necessary, performs the required tasks, and changes its state. Note that not all commands are *legal* in each state. For

[3] Copyrighted by the MultiMediaCard Association.
[4] Copyrighted by the MultiMediaCard Association.

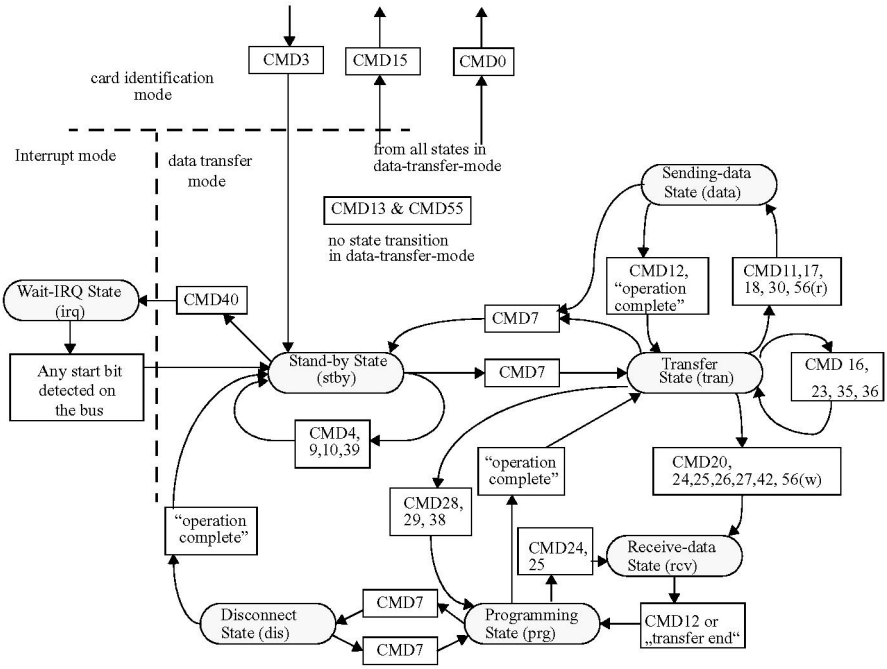

Fig. 1. An example of a state diagram for data transfer mode [3]

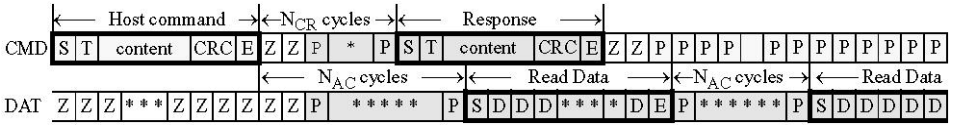

Fig. 2. An example of a timing diagram for Read Multiple Blocks [4]

example, to send a read command is considered *illegal* during the write operation. However, the consequence of an illegal command is also defined in the specification. Upon receiving an illegal command, the card should ignore the command, with sending no response to it. The set of commands is divided into several classes, and an MMC card can support only part of them. We modelled the behavior of 15 commands that are supported by Samsung MMC card.

2.2 Esterel

Esterel[5] is an imperative, textual language that supports synchronous programming paradigm. It is often used to program reactive systems where both concurrency and determinism become important issues. Unlike conventional

concurrent languages such as Java, synchronous languages can provide both the concurrency and the determinism, by *compiling* concurrent programs into the sequential ones. Therefore, Esterel is quite useful in developing complex control automata of reactive systems. Given an Esterel program, the Esterel compiler produces a C code that implements the automaton.

The execution model of Esterel is clock-driven. At each cycle, the *reaction* function of the automaton is invoked. Then, the function processes the input signals, computes the next state to proceed, and emits appropriate output signals to the environment.

An Esterel program consists of a number of *modules* which communicate with *signals*. They are present or absent at each cycle. They can also carry values.

Language constructs are divided into two classes: one that consumes time and the other that is executed instantaneously. The word 'instantaneously' means, in reality, that the execution is finished within relatively short time compared to the cycle time. The first class includes `await`, `repeat-n-times`, `loop`, etc. The second class includes assignment, C-function call, `emit`, etc.

Esterel has a number of control structures. The basic ones are sequential and parallel composition. It also has a preemption structure (`trap`) to interrupt a block of a program and direct the execution to an interrupt handler.

There are a number of introductory materials on Esterel and synchronous programming, of which we recommend [1] most.

3 Lightweight Specification-based Testing: Method

Considering the unique features of MMC systems mentioned in the introduction, we concluded that most of the existing researches on specification-based testing cannot be directly applied to validate MMC systems. We summarize the reasons as follows.

- Usually, they come with rather complex tools (ex. [3]) that analyze the formal specification and generate test cases. Because adopting such tools accompanies high learning cost, it is often unacceptable to common field engineers. Moreover, the effectiveness of such methodologies has not been fully proved.
- MMC card interfaces with the host using *serial* buses. The requirements have a form like "the data transmission stops *two clock cycles* after the end bit of the stop command". Thus, the basic unit of communcation is *bit*, not a message. Existing specification-based methods, most of which are designed for communication protocols, can not deal with this kind of

systems. We believe that this problem is due to the lack of layering in the architecture of MMC systems, as explained earlier.

- Many researches on specification-based testing are concentrated on developing test generation methods. Executing the tests has been considered as a separate issue though. It is very difficult because many test generation researches cannot resolve the feasibility problem completely.

Our method has been designed to provide many benefits we expect from specification-based testing. It is considered *lightweight* to be applicable to real-world problems with limited resource. We think it is not too radical to be accepted by the field engineers.

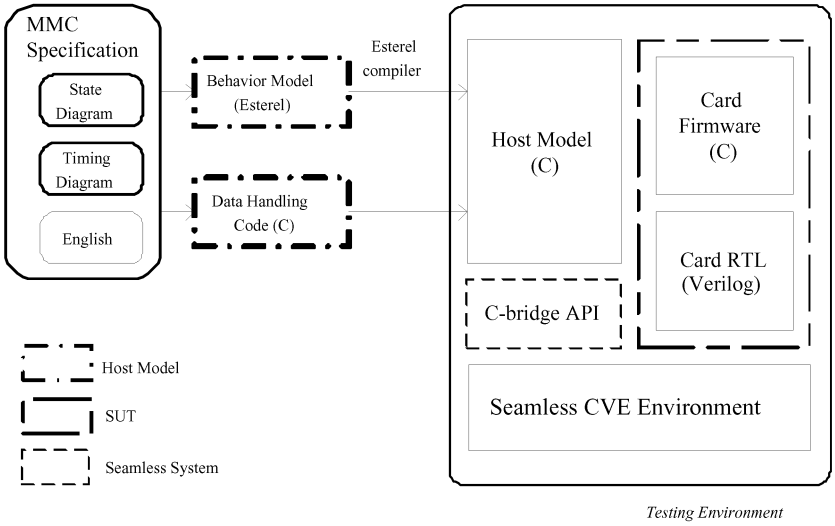

Fig. 3. An overview of our specification-based testing for MMC systems

Figure 3 is the overview of our approach. We model the environment in Esterel. It is compiled into C code, and acts as the host model in Seamless simulation environment. Random testing and scenario-based testing are used to validate the system under testing (SUT) on Seamless.

We show the decisions we made to design the approach.

3.1 Environment Modeling

Contrary to most specification-based testing methods, we build the formal behavior model of the *environment*, not the SUT. In the case of MMC systems, building the model of environment is to formalize the permitted behavior of the MMC host such as PDA, cellphone, or PC. Modeling is done using the

information from MMC specification in order to make fully general and fully nondeterministic model of the MMC host. [5]

We assume that the MMC system consists of the environment and the SUT, of which have almost the same complexity. Suppose neither of the environment and the SUT are formally modeled currently and we are to apply formal methods to them. We will need to model both of them, after all, if we are to benefit from formal verification techniques like model checking[4]. However, given this situation, we have two choices on which of them to model first.

Traditional formal methods usually emphasize the models of SUT. Thus, specification-based testing methods also assume the existence of the models of the SUT, and propose testing techniques for them. Therefore, there is a tacit agreement in the literature that users need to formalize the SUT first to apply specification-based testing.

However, on second thoughts, we have reached a conclusion that building environment models first is more suitable in many situations because we can use them directly as a *test harness*. We can exploit the executability of the host model by generating imperative codes from the model, which can be executed or emulated together with the actual codes of the SUT.

It also enables lightweight application of formal methods[6]. Since many behavioral modeling language comes with compilers that generate imperative codes (usually written in C), the learning cost of engineers is relatively small. Moreover, the adoption becomes easy because the generated code can be used in the existing testing process.

3.2 Language: Esterel

As the behavior modeling language, we choose Esterel[1]. It permits to describe complex, concurrent, communicating state machines very easily.

The rationale behind the choice of Esterel could be summarized as follows.

- It can generate efficient C codes. Many specification languages support code generation facility. However, the primary purpose of code generation is, in many cases, rapid prototyping or simulation. On the contrary, Esterel is supposed to be used as an implementation language directly, so the generated code is quite efficient.
- It is easy to learn for general programmers. Although the synchronous programming paradigm[5] might be a hurdle, programmers with a little background in hardware design can understand the paradigm and the language

[5] Nondeterminism is resolved by random choice when we execute the host model with the SUT.

easily. The imperative syntax of Esterel is also helpful.

- Cycle-based execution model of Esterel is suitable to be used together with hardware description languages. It also enables elaborate validation of hardware and software because we can control the timing of events (ex. sending a command) precisely.
- It is freely distributed. Esterel comes with free but powerful compilation and debugging facilities.

3.3 Testing Environment: Seamless

The actual testing is performed on an emulation environment Seamless, which enables co-validation of hardware and software[11]. The lack of specification of separation of functionalities between hardware and software also leads to combined testing of them.

Seamless can execute software code written in C and hardware code written in Verilog together. Because SUT is the card, we should provide the host model to Seamless simulation environment (Fig.3). We use C code compiled from Esterel program as the host model.

3.4 Test Execution

The host model we build in Esterel is maximally nondeterminstic, which means it generates all possible behaviors with no conflict to the specification. However, we should resolve the nondeterminism to use the model for testing purpose.

There are several kinds of nondeterminism in our MMC host model. In a state, the host should determine: 1) which command to issue next, 2) after having received the response, how long time to wait before sending the next command, 3) the number of data tokens to send, and so on. They correspond to the outgoing transitions from a state.

We use two methods to resolve nondeterminism. The first is to use predefined scenarios. A number of errors which have been reported for Samsung MMC card are used as scenarios. The second is random simulation. We provide Esterel code with C code that chooses a transition randomly.

During the test, we perform branch coverage analysis to measure the completeness of testing. We record the selected transitions in a state and report the coverage result for them after testing is finished.

Note that we cannot provide 100% branch coverage because the range of interval and the number of blocks are infinite. Since issuing commands with different data values or with different delays can exhibit different behaviors,

these commands should be considered as different transitions. As a compromise, we test with pre-defined sets of intervals and numbers of blocks.

4 Modeling MMC Host

The MMC host model is built using Esterel and C code. Esterel code takes the role of the control automaton and C code provides data manipulation and random test selection mechanism.

4.1 Esterel code

We manage the complexity of MMC specification by carefully dividing the functionalities into modules and designing the communication among them.

Most complexities lie in the specification of *interaction between commands*. For example, the host can issue a number of commands *during* reading or writing data, including Stop Transmission, Deselect Card, Go Inactive State, Go Idle State, etc [9]. Thus, the modules for those commands should interplay to properly handle those situations.

The set of modules is divided into three groups. They are explained below.

4.1.1 Main module

An Esterel program has a unique main module. It defines interfaces, configuration, and global state changes of the system.

Figure 4 shows the body of the main module. It is enclosed by a loop, which means the system never terminates itself.

The loop body is divided into two parts. When the control is in the first part, the card is operating normally, i.e. without any violation against the specification. Some modules are instantiated normally by `run` commands (lines 9-11), but others are instantiated within another nested loop with `abort-when` construct (lines 14-16). This distinction is needed to deal with the *soft reset* by CMD0. The module `CMD00` can emit the signal `Soft_Reset`, and it enforces restart of modules for other commands. Thus, the module `CMD00` itself should be outside of the `abort-when` construct.

As soon as a violation against the specification is detected, the signal `Violation_Found` is emitted. The first `trap` block (lines 3-18) is terminated, and the control is transferred into the second part of the loop body. Now we found a violation, we should reset the card to continue the testing, by issuing a CMD0.

```
1   loop
2     pause;
3     trap Reset_after_Violation in
4       every Violation_Found do
5         pause;
6         exit Reset_after_Violation
7       end every
8       ||
9       run CMD00 || run Scheduler || run Dataline_Moniter ||
10      run CMD_Sender || run Response_Receiver ||
11      run Data_Receiver || run Data_Sender
12      ||
13      loop abort
14          run CMD01 || run CMD02 || run CMD03 || run CMD07 ||
15          run CMD09 || run CMD10 || run CMD12 || run CMD13 ||
16          run CMD17 || run CMD18 || run CMD23 || run CMD24_25
17      when Soft_Reset end
18    end trap;     % Reset_after_Violation
19
20    % Send CMD0 for reset
21    trap CMD0_INIT in
22      loop
23        run CMD_Sender
24        ||
25        [
26          pause;
27          emit Put_cmd(0); emit Put_arg(0); pause;
28          await 48 tick;
29          exit CMD0_INIT
30        ]
31      end loop    %loop
32    end trap      %CMD0_INIT
33  end             % Loop-end
```

Fig. 4. The part of main module, `MMC_host`

4.1.2 Modules for Interface

MMC has two serial communication lines, CMD line for commands and responses and DATA line for data transfer. Modules for interface manage these lines. They take the roles of packing (resp. unpacking) tokens from (resp. to) bits, notifying other modules of arrival / sending of tokens, etc.

Interfacing with Seamless environment is also handled by these modules.

For example, the module `cmd_snd` is responsible for putting a command token on the command line. It calls the API function Seamless provides for the purpose of simulating writing a bit of data on the interface line.

Lines 9-11 in Figure 4 show the instantiation of these modules.

4.1.3 Modules for commands

MMC specification defines a number of commands. In our MMC host model, each MMC command is represented by a module that manages the control flow of the communication during the processing of that command. When a command token is issued, the module in charge of global scheduling (`Scheduler` at line 9 of Figure 4) emits a signal indicating the beginning of that command. Then, the corresponding module notices it and starts the appropriate processing.

As an example, we show a module of modest complexity. In Appendix A, we present Esterel code for processing CMD18 (Read Multiple Block) to show an example of interaction among modules. The timing diagram for CMD18 in Figure 2 would be helpful for understanding the behavior of CMD18.

CMD18 is used to read multiple blocks of data from MMC card. The number of blocks to read and the length of a block should be set by other commands before CMD18 is issued. Upon receiving CMD18, the card starts to send data blocks via the DATA line. The transfer can terminate in several ways, such as having received all the specified number of blocks, issuing a Stop Transfer command, or issuing a Card Reset command. CMD18 can be executed only at Transfer State (TRAN). When the card receives CMD18 at the state TRAN, it changes its state into Sending-data State (DATA). After finishing the transfer, it returns to the state TRAN. Refer to Figure 1 which represents the state diagram for data transfer mode.

In the module for CMD18 in Appendix A,[6] lines 10-52 are the main body. The `loop` construct means that this body should start whenever CMD18 is issued. Lines 11-16 take the role of waiting for CMD18. Whenever a command is issued (line 12), it checks whether it is CMD18 or not. If it is, it exits the `every` loop by exiting the trap (line 14). Lines 19-50 are executed when a command is issued in TRAN state (line 18). It consists of two threads, which deals with CMD line (lines 20-33), and deals with DATA line (lines 36-48).

We use two threads because reading process can be terminated in a few ways. The second thread (lines 36-48) is for processing normal transfer. It receives pre-spceified number of data blocks (lines 38-42) and emits a signal `state_changed(TRAN)` to indicate the end of transfer.

[6] Lines declaring signals and variables are omitted.

However, there are other ways to end the transfer before all the blocks are transmitted. Lines 28-31 represent such a way that the signal `state_changed (TRAN)` can terminate the transfer. Note that the signal `state_changed(TRAN)` can be emitted by other modules, too. The module for CMD12 (Stop Transmission) also emits that signal to indicate aborting of progressing transfer. Thus, lines 28-31 actually deal with two situations: normal transfer termination and termination by Stop command.

On the other hand, the first thread announces that the state is changed into DATA state when it receives the signal `end_response` (lines 20-21). The signal `end_response` is emitted by the module that is in charge of assembling bits on the CMD line into a response token.

In this way, the modules interface with the card and other modules.

4.2 C code

Like general Esterel programs, our host program needs C functions for data manipulation. We explain a few important functions.

- Function `select_next` is used to select the next command to execute and to decide how many cycles for the host to wait before issuing the command, after having received a response for the previous command. In the scenario-based mode, the selection is done by the predefined scenario which is a list of pairs of command and delay. In the random-execution mode, decision is made randomly. However, different weights are given to the commands to reflect reality. For example, the reset command CMD0 is given a low weight in order to test sequences of commands long enough.
- Function `update_coverage` has a role of calculating the branch coverage. Coverage is managed to measure the completeness of testing. It is invoked whenever a command is issued.
- A number of functions are defined to handle the actual interface with Seamless. They use C bridge interface to read or write bits on the lines. (Refer to Figure 3.)

5 Experiments

Using the host model we described above, we test the MMC card in Seamless environment.

As many advocates of formal methods have asserted, modeling process itself was revealed to be an effective means for clearing ambiguities in the informal specification. For example, Figure 3 has transitions whose triggering conditions are “operation complete” and “transfer complete”. However, exact

definitions of these terms are not directly given in the specification. During the modeling process, we had to define the precise meaning of these triggering conditions, by carefully reading of the specification and discussing with domain engineers.

After the modeling was finished, we experimented with the scenarios that were identified as *compatibility problems*. They were found by actual testing of MMC card with physical MMC hosts. Identified scenarios are modeled as a sequence of MMC commands, where a delay is associated to each command.

$$\langle(0, 8000), (1, 200), (2, 200), (3, 200), (7, 64), (24, 8)\rangle$$

For example, the above scenario includes 6 MMC commands. Each element in a list is a pair of the command and the delay; the fifth command, CMD 7, will be issued 64 cycles after the host receives the last bit of the response for CMD 3 it sent to the card before.

We received a set of three compatibility problems in the form of sequences of commands from domain engineers. However, they didn't include the timing (delay) information we thought crucial. We were able to fill the missing timing information by testing with various delays. It shows the simulation-based testing is powerful because it enables cycle-accurate testing of MMC card.

We also found other cases that were identified as violation of official specifications. For example, when we send an illegal command during the write operation, the CRC bits on the data line become abnormal under certain conditions. We have identified four abnormal cases during random testing.

When we report our findings, domain engineers usually react with an annoyed look, rather than surprise. They ask us back why they should take such odd cases into consideration. The good answer we provide is, of course, that most causes of the compatibility problems that have been found, also look ridiculous unless they are real. In most cases, we couldn't understand why the host behaves in such a strange way. However, any behavior of the host that does not violate the official specification should be supported. Therefore, the right way to prevent future errors is to conform to the official specification as much as possible.

6 Conclusion

In this paper, we introduced a lightweight specfication-based testing method for memory card systems. Instead of testing with many real MMC host devices, we proposed to build the cycle-accurate behavior model of general MMC host from the official MMC specification and to use it on a co-simulation environment. We were able to reproduce the error scenarios on the simulator, which was found to be helpful in debugging the errors. Testing method using

formal model also discovered a number of previously unknown specification-violations that have not been treated well.

Our method is not so theoretically elegant nor radically novel. However, we believe it can directly benefit many engineers facing similiar problems with us. It can be integrated into existing testing process right with little burden. Using formal modeling language is an easy, but effective way to implement the test harness precisely.

The learning cost of Esterel was found to be acceptable, especially compared to other heavy modeling languages, such as UML. One of our engineers, who has an M.S. degree in electronic engineering and only C programming experience, could learn the language and the paradigm behind it within one week. The visual simulation and debugging environment of Esterel was very helpful to build the host model quickly.

However, we also found some difficulties in conducting the project as follows.

- Though the simulation environment of Esterel, xes, proved to be excellent in stand-alone debugging, it couldn't be integrated well with other tools, such as Seamless. For example, to make time progress is possible only by clicking the 'tick' button in the simulator. If there is an API for emulating the clicking, xes could be used in connection with Seamless environment. We think adding API's would simplify the integration testing very much.
- Emulating hardware and softwre together took much more time than we expected. For example, it took about 10 minutes for the MMC card model (written in Verilog) to start normal operation after power-on. The problem can be partially solved by storing and restoring the simulation status. We have heard that the forthcoming version of Seamless will provide one solution for this problem by translating Verilog code into C. We believe that it will ease the speed problem much.

There exist dedicated commercial tools for verification of embedded systems. Two of them have reached noticeable level of industrial acceptance: Specman from Verisity[15] and Vera from Synopsys[12]. We had a chance to compare our approach with that of Specman. It uses their proprietary modeling language e, and a set of facilities that are useful for modeling and performing the verification. Constraint solver enables modelers to write parameterized test scenarios with constraints for actual data values. Coverage analyzer reports the degree of completeness after a set of test are performed. A bit primitive forms of both tools were made in our framework, but in a rather ad-hoc manner.

Although using these dedicated tools seems fruitful for many cases, we

argue that open, not proprietary language such as Esterel has its own benefits. First of all, we need not be dependent to a specific tool vendor. Moreover, we could benefit from other analysis tools freely available for Esterel, such as model checker Xeve[2].

There are a number of direction worth further research.

- Now we have a formal model of the MMC host, we could conduct a formal verification such as model checking. The biggest challenge is probably to build the model of MMC card. It would be much harder than building the host model because the model should reflect enough details of a concrete product. However, if the next version of Seamless provides the facility of abstracting Verilog code to C code, it would ease the modeling task very much.
- It will be interesting to compare the testing result from our method and those from other existing specification-based testing methods. To do this experiment, we need to build the model of MMC card, too. But, in this case, the model would be built using the information from the specification, not from the actual codes of the card. We do not think it is difficult because there is a duality between behaviors of the host and the card.
- Our approach of building host model in Esterel could be combined into the SystemC[13] framework. SystemC is rapidly becoming a de-facto standard language for system-level design of SoC systems. Testbenches of SystemC design are usually written in SystemC, too. However, our experience seems to show that Esterel is more adequate than SystemC because of its powerful control structures. We are planning to apply our approach to system-level design of MMC card when the design will be available.
- During the experiments described in Section 5, we found that there is no standard language for specifying test cases of embedded systems like memory cards. Thus, engineers should program their testbenches from scratch, using general purpose languages like Verilog or C. The situation is different for communication systems, where TTCN[14] is considered as a standard language for describing test cases. Verification engineers using TTCN can concentrate on the logical correctness of test cases, rather than having to pay attentions to irrelavant details. We strongly believe that a test language for embedded systems would easy much of the burden of verification engineers of this field.

Acknowledgements

The authors would like to thank Mentor Graphics, and especially Seok Man Jung and Soo Yong Lee for providing us invaluable supports in using Seamless CVE.

References

[1] Berry, G.: The Esterel Language Primer, version v5.91. ftp://ftp-sop.inria.fr/esterel/pub/papers/primer.pdf

[2] Bouali, A.: Xeve: an esterel verification environment, In the 10th International Conference on Computer Aided Verification, volume 1427. LNCS, 1998.

[3] Bousquet, L., Ouabdesselam, F., Richier, J.-L., and Zuanon, N.: Lutess: a specification-driven testing environment for synchronous software. In 21st International Conference on Software Engineering, ACM Press, May (1999), 267– 276.

[4] Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., and Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Proceedings of the 4th annual sympsoium on logic in computer science, June (1990)

[5] Halbwachs, N.: Synchronous Programming of Reactive Systems. (1993)

[6] Jackson, D. and Wing, J.: Lightweight Formal Methods, IEEE Computer, April (1996)

[7] Jagadeesan, L. J., Porter, A., Puchol, C., Ramming, J.C., and Votta, L.G.: Specification-based testing of reactive software: Tools and experiments. In Proceedings of the International Conference on Software Engineering, May (1997)

[8] Malaiya, Y.K., Li, N., Bieman, J., Karcich, R., and Skibbe, R.: The relationship between test coverage and reliability. in Proceedings of fifth International Symposium on software reliability engineering, (1994)

[9] The MultiMediaCard System Specification, 3.1. MMCA Technical Committee, http://www.mmca.org/tech/tech.html

[10] Paleska, J.: Test Automation for Safety-critical systems: Industrial Application and Future Developments. In Proceedings of the Formal Methods Europe Conference FME 96, (1996)

[11] Seamless Hardware/Software Co-Verification, http://www.mentor.com/seamless.

[12] Synopsys, http://www.synopsys.com/

[13] SystemC, http://www.systemc.org.

[14] ISO/IEC 9646-3:1998, Part 3: Tree and Tabular Combined Notation.

[15] Verisity, http://www.verisity.com/

[16] Zhang, L., Luo, X., Chen, M.: Comparison of Random and Partition Testing Considering the Test Profile. Journal of Testing and Evaluation, Vol 31, No. 2, (2003)

A Module for READ_MULTIPLE_BLOCK

```
module cmd18:  %% CMD 18 : READ_MULTIPLE_BLOCK
  %% Signal definitions: omitted due to space limitation
  function CMD07_is_mine (integer) : boolean;
  function get_number_blocks() : integer;

  var cmd : integer,
      nn : integer
  in
    trap E in
      loop
        trap T in        % wait for CMD18
          every end_cmd do
            cmd := ? end_cmd;
            if cmd = 18 then exit T end if
          end
        end trap;
        trap Stopped in
          if (? state_changed) = TRAN then  % Legal cmd
          [   % This thread deals with CMD line of bus
            await end_response;
            emit state_changed(DATA);    % Now, host is reading data
            loop
              await
                case end_cmd do
                  if (?end_cmd = 7 and (not CMD07_is_mine(?end_cmd_arg)))
                      or ?end_cmd = 15 then exit Stopped
                  end if
                case state_changed do
                  if ?state_changed = TRAN then  % transfer ended
                    exit Stopped
                  end if
              end await
            end loop

          ||  % this thread deals with DATA line
            nn := get_number_blocks();
            trap LoopT in loop
              abort
                await (NAC * 10) tick;
                exit E
              when start_data;
              await end_data;
              nn := nn - 1;
              if nn = 0 then exit LoopT end if
              end loop
            end trap;
            pause;
            emit state_changed(TRAN)
          ]
          end if
        end trap  % Stopped
```

```
52          end loop
53        handle E do
54          sustain Violation_Found
55        end trap
56      end var
57    end module
```