IC Design Validation Using Message Sequence Charts

Harald Vranken[1] , Tomás Garciá Garciá[2], Sjouke Mauw[2], Loe Feijs[2]

[1] Philips Research Laboratories, IC Design - Digital Design & Test
Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
harald.vranken@philips.com
[2] Eindhoven University of Technology, Eindhoven Embedded Systems Institute &
Department of Mathematics and Computing Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{sjouke, feijs}@win.tue.nl

Abstract

In this paper we describe a design validation method based on simulation of behavioral models, in which Message Sequence Charts (MSC) are used to visualize the simulation results and to aid in debugging. Thereto, we have extended a Philips proprietary tool, called TSS, with the possibility to visualize simulation traces.

1. Introduction

The design of complex digital ICs generally proceeds from high-level, behavioral models in the early design stages to RTL and gate-level models in later design stages. High-level, behavioral models allow to focus on system-level and architectural issues [1]. This paper describes a design validation method based on simulation of behavioral models, in which Message Sequence Charts (MSC) are used to visualize the simulation results and to aid in debugging.

MSC is a standardized language for displaying the communication behavior of system components [4][5][6][7][8]. MSC has already been successfully applied in the area of telecommunication systems for decades. Their main use is for the definition of system requirements, for displaying simulation runs and for expressing test cases. There are several reasons for their popularity in this field of application. First of all, the graphic and intuitive nature of MSC makes it a perfect language to communicate about the system with non-specialists. They are especially suited for capturing initial requirements, and discussing scenarios. Moreover, since the MSC language is standardized by the International Telecommunication Union, the language is being maintained actively and supported by professional tools.

We propose to use this diagrammatic language in the realm of IC validation. We believe that the virtues of the language can also help in this area to understand the interaction between the components of an IC. Rather than focusing on the application of formal verification techniques, which is also feasible with the aid of MSC, we will focus our attention on the use of MSC for displaying simulation runs in order to more easily understand the operation of a complex digital IC. In particular, MSC can help to display and analyze faulty behavior, thus supporting the debugging of ICs.

TSS (Tool for System Simulation) is a Philips proprietary tool for simulation of behavioral models, which is used in high-level design and validation of complex digital ICs. We extended TSS with the capability of capturing and visualizing simulation results using MSC. There are several research issues connected to this application of MSC. First of all, the question arises which information should be displayed in the MSC: which are the relevant processes to show and which events should be displayed in what way? After experimenting, we designed a mapping of TSS simulator output to MSC. The second question is which computer support is needed to navigate the user through the generated MSC. MSC generated by simulators such as TSS, typically contain millions of events, and tool support is needed to project onto parts of the system or parts of the behavior. In this paper, we will address both issues.

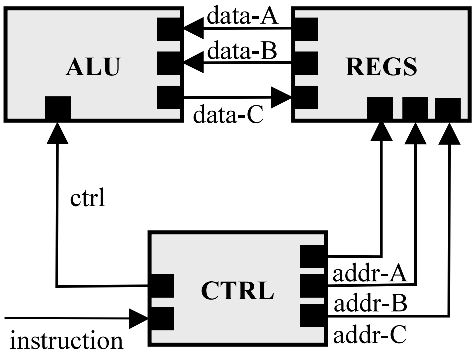
The current paper is organized as follows. The TSS simulation environment for behavioral models is described in Section 2, and MSC is introduced in Section 3. In Sections 4 and 5, it is explained how MSC is integrated into the TSS simulation environment, and how this facilitates IC design validation. Experiments are presented in Section 6, and conclusions are given in Section 7.

2. System simulation

Within Philips, TSS (Tool for System Simulation) has been developed, which provides a simulation framework for high-level, behavioral models [2][3]. A behavioral model in TSS consists of two hierarchical levels: modules and processes. The behavior of a complex IC is modeled as a network of modules that are statically connected by channels. Typical examples of such modules are a bus interface, a cache memory, or an ALU. Modules communicate by sending messages to each other over the channels. The behavior of a TSS module is modeled by a number of synchronous and/or asynchronous processes, while the behavior of these processes is expressed in the ANSI-C language.

REGS is now activated: it reads the data values as specified by the two operand addresses, and it outputs the data values to module *ALU* over the channels *data-A* and *data-B*. The module *CTRL* also sends a message on the channel *ctrl* to module *ALU* containing the operation. The *ALU* now performs the operation using the two data values as arguments. Finally, the result of the *ALU* operation is written back into the register file via channel *data-C*.

Figure 1b shows the TSS model for module *REGS*,



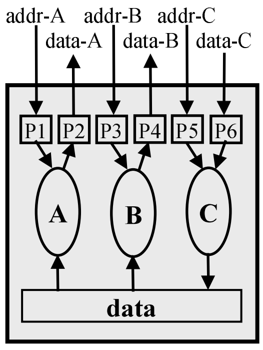


Figure 1 a) TSS behavioral model **b) TSS module *REGS***

Figure 1a shows the TSS behavioral model of an example system that is composed of three modules: *ALU*, *CTRL*, and *REGS*. The modules communicate by sending messages over channels. The channels are depicted in Figure 1a by arrows and associated to each channel is the channel name. A module is connected to a channel by means of a port, as depicted in Figure 1a. The system represents a simple CPU, containing an ALU to perform arithmetic and logic operations on two data values, a register file to store data, and a controller to control the operation of the ALU and the register file.

The typical behavior of the CPU is as follows: The system is triggered whenever the module *CTRL* receives a message on the channel *instruction* from the environment. This message corresponds to an instruction that has to be executed by the CPU. The module *CTRL* decodes the instruction and extracts an operation, the addresses of two data values in the register file on which the operation should be performed, and the address in the register file where the result of the operation should be stored. For instance, the operation may be to add two data values that are stored at address 1 and 2, and to store their sum at address 3.

Next, the module *CTRL* sends three addresses to the module *REGS* via the channels *addr-A*, *addr-B*, and *addr-C*. The messages on *addr-A* and *addr-B* specify the addresses of the two data operands, while the message on *addr-C* specifies the address for the result. The module

which consists of three processes (*A*, *B*, and *C*), six ports (*P1* through *P6*), and a data structure (*data*). The ports connect the module to the channels. The processes read data from or write data to the data structure. Process *A* is started whenever it receives a message from channel *addr-A* via input port *P1*. The process will then read a data value from the data structure *data* and send this data on channel *data-A* via output port *P2*. Process *B* operates in a similar way, using the ports *P3* and *P4* which are connected to the channels *addr-B* and *data-B*. Process *C* is started whenever it receives messages on the input ports *P5* and *P6* via the channels *addr-C* and *data-C*. The process will then write the data value into the data structure *data*.

TSS provides a simulation kernel to simulate a behavioral model composed of modules and processes. The TSS kernel implements various communication and synchronization mechanisms for interactions between modules and processes. The TSS simulation kernel and the behavioral model are compiled and linked together into a single executable. Simulation is simply achieved by running the executable.

TSS provides a user interface to control the simulation. The user interface provides an access mechanism to control and/or observe values in data structures, ports, and channels during simulation. In this way, stimuli can be offered to the behavioral model and responses can be observed. The user interface also provides control to run the simulation for a number of cycles, or until some predefined events occur. Validation of the model is performed by simulating the model with the appropriate test stimuli.

Whenever the simulation results differ from the expected results, an error in the model has been detected. Debugging is required next to find and repair the root cause of the error. The user interface is the primary aid for debugging TSS models. Via the user interface, the user can control the simulation (e.g. running for a number of cycles, setting up breakpoints), and access objects in the TSS model. For more detailed debugging, the traditional validation approach is to use a C source code debugger (e.g. GNU's *gdb*), since the TSS behavioral model is described in the ANSI-C language. However, debugging at the system level is rather difficult in the traditional validation methods, since this requires a high-level, abstract view of the system in terms of processes and their communication. This problem is relaxed by the use of MSC.

3. Message Sequence Charts

Message Sequence Charts (MSC) [4][5][6][7][8] offer a standardized, formal method to describe and visualize the interaction between processes in a system that is composed of communicating concurrent processes. MSC is primarily applied in the areas of telecommunication and software engineering for requirements specification, simulation, validation, test case specification, and documentation. This paper describes a new application area for MSC: IC design validation.

Figure 2 shows an example of an MSC for the system in Figure 1. The TSS modules are represented by the MSC instances *CTRL*, *REGS*, and *ALU*. The arrows in the figure indicate the sending and receiving of messages between MSC instances. Hence, the MSC visualizes the communication behavior. Time proceeds from top to bottom. MSC has no global notion of time and it implies a partial ordering of messages. Figure 2 shows the typical system behavior as described in the previous section.

We propose the use of MSC to visualize TSS simulation. MSC is an intuitive way to express the communication behavior of concurrent processes. In addition, MSC is standardized, which implies that syntax and semantics are well defined. The latter also implies that commercial tools can be used that operate on MSC, e.g. for visualization or formal verification.

MSC-like diagrams are often used in object-oriented design techniques, such as UML [9][10]. The MSC language as standardized in the recent release of the ITU standard (MSC2000), supports all features required to be used with UML. This new release of the language makes it possible to also express timing requirements in MSC and thus to evaluate a design with respect to performance indicators.

4. Linking TSS and MSC

We carried out two steps to use MSC for visualizing TSS simulation results:

- We defined how TSS modules, processes, and communication and synchronization behavior are expressed in MSC. We also considered how the event-based and cycle-based actions of the TSS simulation kernel are represented in MSC.
- We developed tools to automatically generate and visualize MSC during a TSS simulation run.

Figure 2 already showed how TSS simulation results can be visualized with an MSC. The MSC in Figure 2 provides an aggregate view, where each MSC instance corresponds to a TSS module. The term 'decomposed' in Figure 2 indicates that each MSC instance can be refined at a lower level. Hence, for each decomposed MSC

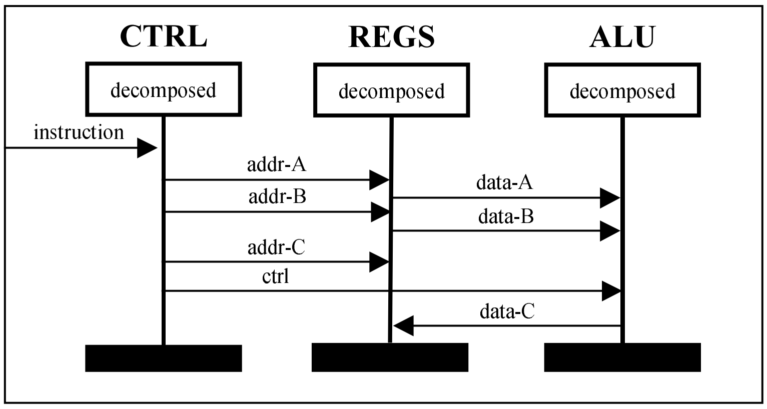


Figure 2 Aggregate MSC of example system

instance, which corresponds to a TSS module, a refined view can be given to show the communication behavior between the processes inside the TSS module.

Figure 3 shows the refined MSC view for TSS module *REGS*. As shown in Figure 1b, module *REGS* contains three processes (*A*, *B*, and *C*), four input ports (*P1*, *P3*, *P5*, and *P6*) that are connected to channels (respectively *addr-A*, *addr-B*, *addr-C*, and *data-C*), and two output ports (*P2* and *P4*) that are also connected to channels (*data-A* and *data-B*). The MSC in Figure 3 indicates that there is an MSC instance associated with each process and each output port. The MSC contains three message types (*m1*, *m2* and *m3*) which have the following meaning:

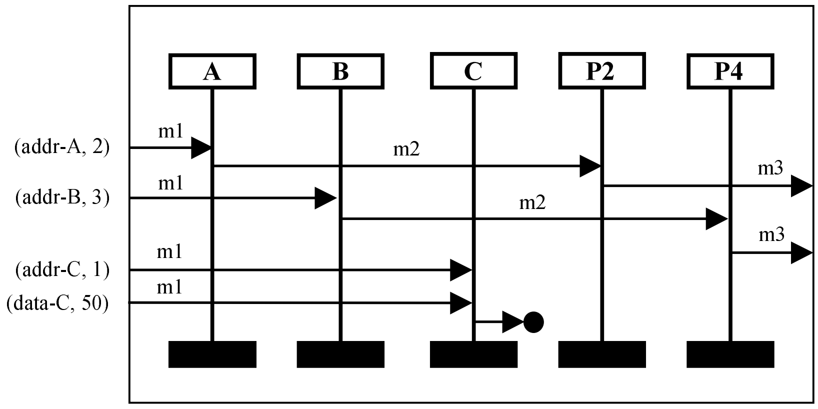


Figure 3 Refined MSC for module *REGS*

- An *m1* message indicates that a channel has been updated and that the receiving process is triggered. The label in front of the *m1* message in Figure 3 indicates the channel name and the value on this channel. E.g. the label *(addr-A, 2)* indicates that channel *addr-A* has the value *2*. The TSS process that corresponds to the receiving MSC instance, is connected to the updated channel by an input port. The process is triggered whenever it receives a message on this input port.
- An *m2* message indicates that the TSS process which corresponds to the sending MSC instance, has been executed. For example, the *m2* message from instance *A* to instance *P2* in Figure 3 indicates that the TSS process *A* has been executed. An *m2* message either goes to an MSC instance associated with an output port, which indicates that the process has updated the output port, or the *m2* message is a 'lost message', which indicates that the process has been executed and did not update any output port. The *m2* message from instance *A* to instance *P2* indicates that process *A* has updated its output port *P2*. The lost *m2* message from instance *C* indicates that process *C* has been executed without modifying an output port.
- An *m3* message indicates that a channel has been updated, i.e. the data value is copied from the output port to a channel. The label indicates the channel name and the data value. For instance, the *m3* message from instance *P2* in Figure 3 has label *(data-A, 37)*, which indicates that the value *37* has been copied from output port *P2* to channel *data-A*. An *m3* message in a particular MSC will cause an *m1* message in another MSC. For instance, the *m3* message with label *(data-A, 37)* will cause an *m1* message in the MSC for module *ALU*.

Although MSC implies a partial ordering of messages in time, obviously the simulation results obtained by a TSS simulation are totally ordered in time. Hence, the MSC obtained from TSS simulations should be interpreted as totally ordered.

We extended the TSS simulation kernel so that MSC are generated during simulation. The TSS kernel outputs a file during simulation that contains the MSC in a textual format. We extended the TSS user interface to control the generation of MSC: the modules, processes, and ports to be included in the MSC can be selected, as well as the time periods during which MSC are generated. Hence, MSC can be generated that incorporate only part of the behavior model, and that show only part of a simulation run.

The MSC file is loaded next into an additional tool, which generates a graphical representation of the MSC. After experimenting with several academic and commercially available tools for displaying MSC, we decided to start the development of a dedicated displaying tool for the MSC as generated by TSS. We experienced that MSC as generated by even small TSS examples, are considerably larger than can be handled by of-the-shelf tools. Both the width of an MSC (i.e. the number of instances) and the length of an MSC (i.e. the number of displayed events) are often too large to draw conclusions from the MSC. The MSC displaying tool which is now under development will therefore support decomposition of a large MSC into smaller ones, abstraction of sets of processes, and projection onto sets of events. Current experiments show a considerable increase of the applicability of our methodology over conventional tools.

5. Design validation

Design validation for TSS behavioral models is done through simulation. Experiences learn that for complex models it is difficult to validate simulation results and to debug errors that are identified during simulation. This is particularly the case for behavioral hardware models which incorporate lots of parallelism. The use of MSC to visualize TSS simulation results, relaxes these difficulties. The graphical representation of MSC provides an intuitive way to validate simulation results. When an error is identified in an MSC, this provides already a coarse-grain indication of which modules and processes are causing the error. C source code debugging tools (e.g. GNU's *gdb*) and debugging by means of the TSS user interface are used next to find the root cause of the error.

6. Experiments

We performed several experiments in which we used MSC to validate and debug TSS models. Initially, we used some small TSS models to demonstrate the feasibility of the approach and to develop the tools. Next, we performed experiments on the TSS model of the TriMedia CPU64, which is a complex VLIW processor designed at Philips [11][12].

Figure 4 shows a refined MSC for the module *ICACHE* in the CPU64, which represents the instruction cache (i.e. the cache memory to store instructions). The MSC shows two processes: *clk_process* which is a synchronous process triggered whenever the clock channel *CLK* is updated, and *async_process* which is an asynchronous process triggered whenever a new instruction address is received on channel *PAX*. The MSC also shows four output ports: *INSTR_FMT* to output the compression format for the current instruction, *INST_SIZE* to output the size of the instruction, *INSTR* to output the instruction itself, and *ISTALL* to output whether the cache outputs are invalid. In clock cycle 32, the cache stalls (the value 1 is output to channel *ISTALL*) which indicates that the cache cannot output the instruction at the required address. In cycle 33 the cache however is ready, and the required instruction, as well as its size and format, are output.

Our experiences are that MSC clearly aids during design validation and debug. The visualization of simulation results is a powerful aid to validate TSS models and to locate errors. Due to limitations of the visualization tool, we could not benefit from the true potential of MSC. Improved tools, offering decomposition and abstraction should improve this situation.

7. Conclusions

We presented a design validation method based on simulation of TSS behavioral models, using Message Sequence Charts. Our experiments demonstrated the added value of MSC for visualizing simulation results and debugging.

Visually displaying the information of a simulation run considerably helps in tracking down bugs. Still, even if the information is displayed in a two dimensional formalism, the large amount of data may flood the user. We dealt with this problem in two ways. First, we have introduced two levels of abstraction in our mapping from TSS to MSC, allowing the user to observe the system from two perspectives. We found that the two levels, as defined by the TSS approach, could be accommodated in the ITU recommended MSC language very well by exploiting the *decomposed* construct. Next, we are developing dedicated MSC displaying tools which support the user in (de)composing a large MSC and abstracting from abundant information.

Currently, we only use MSC to support debugging. Since MSC is a formal language, and since formal analysis techniques are available for MSC, there are still many further options for developing support for automated analysis of the resulting MSC. One could for example validate timing properties of the generated MSC.

8. References

[1] H. Vranken, *Design for Test & Debug in Hardware/Software Systems*, Ph.D. Thesis, Eindhoven University of Technology, 1998.

[2] A. Versluys, S. Koot, P. de Visser, *TSS Programmers Manual*, Philips Semiconductors, 1998.

[3] S. Koot, A. Versluys, P. de Visser, *TSS User Manual*, Philips Semiconductors, 1998.

[4] E. Rudolf, P. Graubmann, J. Grabowski, *Tutorial on Message Sequence Charts*, Computer Networks and ISDN Systems, 28(12):1629-1641, 1996.

[5] S. Mauw, *The formalization of Message Sequence Charts*, Computer Networks and ISDN Systems, 28(12):1643-1657, 1996.

[6] ITU-TS Recommendation Z.120 - Annex B, *Algebraic semantics of Message Sequence Charts*, 1995.

[7] ITU-TS Recommendation Z.120 - Annex C, *Static semantics of Message Sequence Charts*, 1995.

[8] ITU-TS. Recommendation Z.120, *Message Sequence Chart (MSC)*, 1996.

[9] G. Booch, I. Jacobson, J. Rumbaugh, *The Unified Modeling Language User Guide*, Addison-Wesley, 1998.

[10] J. Rumbaugh, I. Jacobson, G. Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1998.

[11] J. van Eijndhoven et al., *TriMedia CPU64 Architecture*, Proceedings International Conference on Computer Design, IEEE, 1999, pp.586-592.

[12] H. Vranken, *Debug Facilities in the TriMedia CPU64 Architecture*, Proceedings European Test Workshop, IEEE, 1999, pp.76-81.

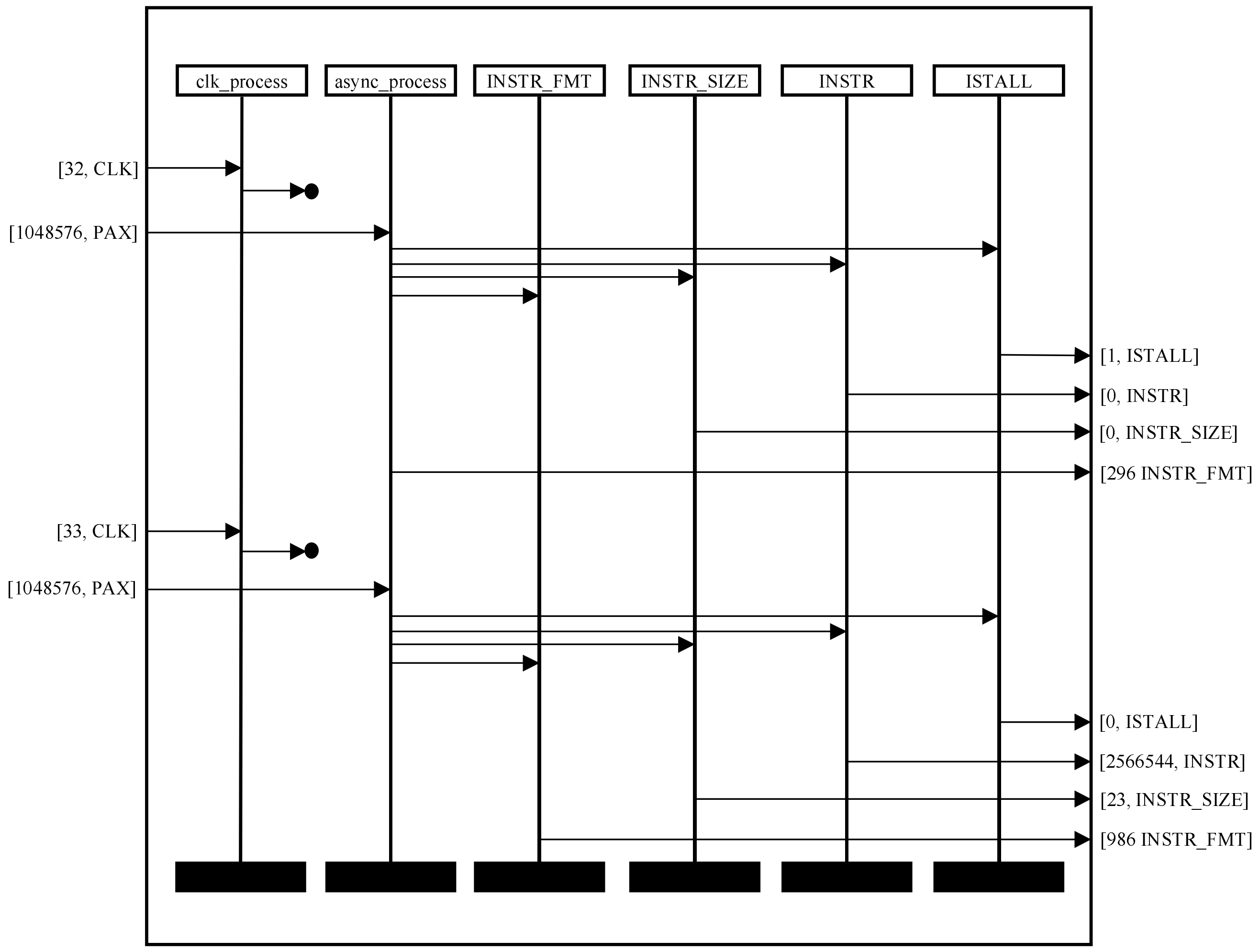


Figure 4 Example MSC from TriMedia CPU64 TSS module *ICACHE*