PARTITIONING AND FPGA-BASED CO-SIMULATION OF STATECHARTS

Rico Dreier
Georg Dummer
Guoxing Zhang
Klaus D. Müller-Glaser
FZI Forschungszentrum Informatik (Research Center for Information Technologies)
Department of Electronic Systems and Microsystems (ESM)
Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
E-mail: dreier@fzi.de

KEYWORDS
Partitioning, FPGA, Co-Simulation, Statecharts, VHDL Code Generation.

ABSTRACT

With the rising complexity and distribution of integrated circuits and embedded systems, the requirements for their development increase as well. Above all, the verification of a large design proves to be the bottleneck of a conventional computer-based simulation in the design flow, requiring several hours to several days. A possible approach to accelerate a simulation is to process different events on several processors at the same time. Assuming a one-processor-architecture, this allows real concurrency of the execution. In a second step the model or a part of the model is executed on reconfigurable hardware. This method is marked by the unique costs of partitioning but takes advantage of the significantly higher execution speed and real parallelism on one chip. This paper presents a framework for a Field Programmable Gate Array (FPGA) based co-simulation of electronic systems as well as an efficient partitioning strategy for Statecharts as a basis for co-simulation. An FPGA based simulation acceleration and the Statechart simulator JStateSim have been developed at FZI/ESM as well as the tool JVHDLGen that enables Very high speed integrated circuit Hardware Description Language (VHDL) code generation out of Statecharts, considering also concurrent charts.

INTRODUCTION

Dependent on the level of abstraction and the different views, a model can be described in different forms. Finite State Machines (FSM) are suitable to specify the behavior of discrete event systems. Extended FSM additionally introduce the concept of state hierarchy, concurrent states and broadcast. In order to model a complex behavior, Harel extended finite state machines by Statecharts including concepts like combination of Mealy and Moore machines (hybrid state machine), conditions, hierarchical states, history junctions, concurrent states and broadcast (Harel 1987).

The CASE tool MATLAB/Simulink/Stateflow (The Mathworks) is used to work on Statecharts, starting from the point, where the model or a part of the model shall be mapped to hardware and then embedded into the simulation. Stateflow is a graphical extension for Simulink and allows the modeling and simulation of state machines (as for example BetterState (Wind River) (Stingl and Dreier 1999)). In contrast to Simulink systems, Stateflow is event triggered, which means that a chart is only processed if a previous defined event occurs. Inside a Simulink model a Stateflow diagram is represented by a Stateflow block. A model can have several of them that can communicate through its interface. A Stateflow machine is defined as the set of all Stateflow blocks in a model. During simulation, signals can be exchanged with Simulink. The Stateflow concept extends Harel-Statecharts, e.g., by junctions (allow representing logical structures such as a `for` loop), scopes (to define the valid scope for events and data), directed event broadcasts (allows sending an event explicitly to a given state), implicit events and condition actions (executed, if the condition is evaluated as true) (Dreier et al. 2001).

The simulation acceleration is based on the integration of a hardware supported simulation (emulation) in a heterogeneous network of multiple simulators. Java Remote Method Invocation (RMI) is a mechanism hiding this from the user. It is designed as client-server architecture between local and remote objects. The communication between both is taken over and almost completely hidden by RMI. In order to accomplish a distributed simulation, the model must be partitioned into several concurrently simulated submodels. Each of them is executed by an own simulator. Goal of the partitioning is to maximize parallel execution and to minimize the communication costs at the same time. Furthermore, the static load balancing and the respective synchronization of the simulators are taken into account (Mehl 1994). Since Statecharts have a complex semantic, they cannot be converted directly into graphs, like logic circuits. Possible partitions, the cost of computation per partition and the communication between the partitions must be analyzed apriori. Then, the Statecharts are modeled in hypergraphs, so that partitioning algorithms, that have been proved useful for logic circuits, can also be applied to Statecharts.

Proceedings 15th European Simulation Symposium
Alexander Verbraeck, Vlatka Hlupic (Eds.)

ISBN 3-936150-28-1 (book) / 3-936150-29-X (CD)

PARTITIONING OF STATECHARTS

Heuristic Iterative Partitioning Algorithms

The algorithm of Kernighan and Lin (KL) (Kernighan and Lin 1969) is an algorithm of bi-partitioning. KL begins with an arbitrary partitioning and exchanges the nodes from the two partitions in pairs, as long as the cut size can be reduced. The KL algorithm treats only nodes with uniform weights, and the size of a partition is fixed. This algorithm cannot be used for partitioning of hyper graphs. The algorithm of Fiduccia and Mattheyses (FM) (Fiduccia and Mattheyses 1982) is an extension of the KL algorithm. The FM algorithm permits the movement of single nodes between partitions. Thus, the size of a partition is changeable. With the FM algorithm the selection of the node from those with the same gain, which can be moved next, is arbitrary. Krishnamurthy (Krishnamurthy 1984) introduced the concept of "level gains", in order to distinguish between such nodes according to their gains from later movements. Sanchis (Sanchis 1989) extended the concept of Krishnamurthy to a k-way partitioning (K-FM). A node is marked as free, if it is not moved in a pass of the algorithm. Otherwise it is marked as locked. The K-FM algorithm of Sanchis can be used for an arbitrary number of partitions and nodes with different weights.

All iterative methods specified above model a circuit as non-directional graph or hyper graph, without considering the direction of signals. In some situations the information about the direction of signals is useful, in order to improve the partitioning (Cong et al. 1994, Chen et al. 1997).

Partitioning of Logic Circuits

Before partitioning a logical circuit, its net list is first modeled in graphs. The partitioning of logical circuits can then be realized as partitioning of graphs or hyper graphs. The partitioning of a hyper graph is formalized in (Lengauer 1990). If a given net list contains sequential components (e.g., flip-flops), such components must be removed first. After the acyclic k-way partitioning these components are assigned to suitable partitions on the basis of the same computing load as well as the connections between sequential components and partitions.

A logical block is modeled with nodes and a net between logical blocks with directed hyper edges. The primary inputs and outputs are represented also with nodes. Figure 1 shows a logic circuit with four gates modeled in MATLAB/Simulink.

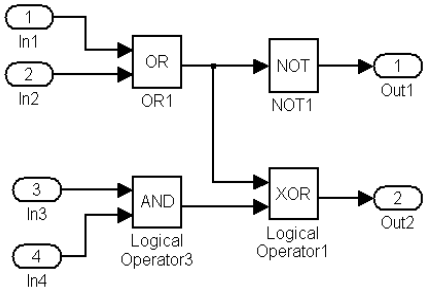

Figure 1: Logic Circuit with Four Gates

The corresponding hyper graph is illustrated in figure 2. For a hyper edge an additional node is added. Here it is marked by a black point. The directed edges between the nodes of a logic gate and the node of a hyper edge represent the input and output nodes of this hyper edge.

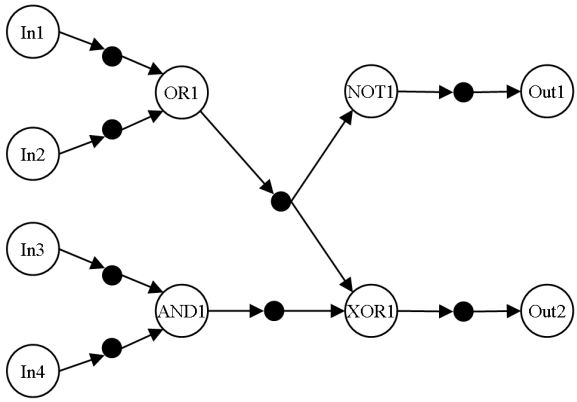

Figure 2: Directed Hyper Graph of the Circuit in Fig. 1

Analysis of a Stateflow Model

Because Statecharts have a complex semantic, they cannot be converted directly in graphs as it is possible with logic circuits. The possible partitions, the computation cost of a partition and communication between partitions must be analyzed first, which is based on the analysis results of the analyzed Statecharts modeled in hyper graphs. Partitioning algorithms for logic circuits can then be used for the partitioning of Statecharts. The concurrency of a distributed simulation depends on the concurrency of the model. A Stateflow machine can contain several charts. Within Statecharts the concurrency depends on the state hierarchy. Figure 3 shows a Stateflow machine with two chart blocks. The two charts are illustrated in figure 4 and figure 5.

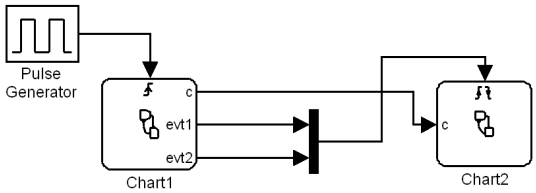

Figure 3: Stateflow Machine with two Statecharts

The state hierarchy can be represented by a tree, whereas the root node is the node for the Stateflow machine. The successors are the nodes for charts. The concurrency of a chart is defined as the number of sub-

states of the chart, if the chart is a state with AND decomposition. If the chart is a state with XOR decomposition, its concurrency is defined as 1, in other words this chart can be simulated on one simulator only.

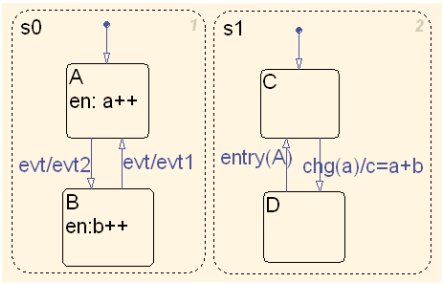

Figure 4: Chart1 of Stateflow Machine in Figure 3

The concurrency of a model is the sum of the concurrencies of all its charts. The maximum number of partitions of a model is decided by its concurrency.

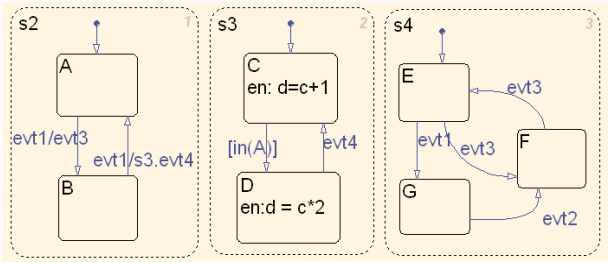

Figure 5: Chart2 of Stateflow Machine in Figure 3

Since only one substate can be active in a state with XOR decomposition at each time, the partitioning of a state with XOR decomposition will produce no parallelism. The possible partition can be either a chart or substates of a chart, depending on whether the chart has XOR or AND decomposition. Figure 6 illustrates the state hierarchy of the model shown in figure 3. The node `m` represents the Stateflow machine; the nodes `c1` and `c2` represent the chart `Chart1` and `Chart2`. The concurrency of the Statecharts `Chart1` and `Chart2` are thus 2 and 3. Therefore, the concurrency of this model is 5. Thus, this model can be divided in 5 partitions at most.

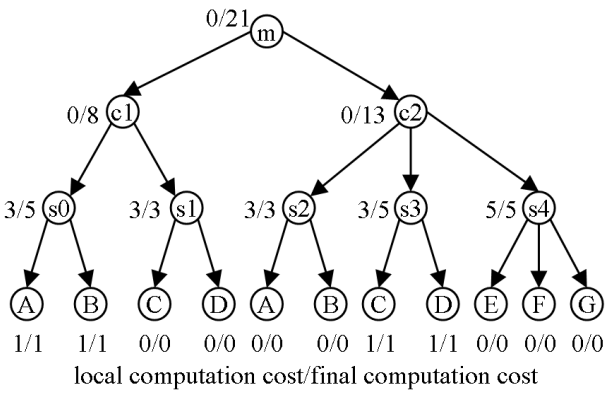

Fig. 6: State Hierarchy of a Model with Concurrency 5

The cost of computation of a model with Statecharts is mainly caused by the execution of state actions and transitions. It is assumed that all transitions and state actions are executed with the same frequency, the computation cost of each transition and state action is the same, and the computation cost of a state is defined as the sum of the number of its state actions and transitions. Since transitions between states can exist in different hierarchy levels, the computation costs are calculated in two steps. First, local computation costs are calculated. The local computation costs of a state are defined as the sum of the number of state actions and the number of transitions between direct substates. This result will be marked first as weight of a node in the state tree. The final computation costs of a state are calculated by accumulating the final costs of its direct substates and its local costs. This calculation is accomplished gradually from the leaves to the root. For the leaves the local costs are also the final costs. Figure 6 shows, how the computation costs of the model given in figure 3 are calculated. Since the root node does not contain state actions and transitions, its local costs are 0. The substates `A` and `B` of the state `s0` each contain a state action. The state `s0` contains 3 transitions including the default transition. The final computation costs of the state `s0` are the sum of the costs of the nodes `A`, `B` and its local costs 3.

Communication Costs Between Partitions

The partitions of states or charts must communicate with each other, if the events generated by a partition need to be passed on to another partition, or if two partitions access common variables. Therefore, the data objects exchanged between two partitions differ from events and variables. A data object can be accessed either by a state action or a transition. In order to determine the communication costs between partitions, a communication graph for a partitioning is defined. A communication channel between two partitions P1 and P2 exists if one of the following conditions is fulfilled:

- A variable will be changed by a state action or transition in P1, simultaneous it is operand of an expression of state action or transition in P2.
- A normal or implicit event in P1 must be sent via broadcast to P2.
- A directed event of a transition in P1 is sent to another transition in P2.
- P1 must inform P2 about the validity of a state within P1 as a state condition.

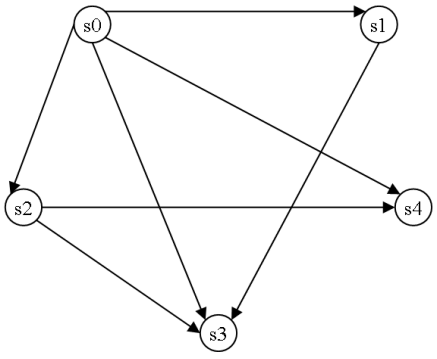

Figure 7: Communication Graph between Partitions

The nodes of the communication graph represent a possible partition. If one of the conditions specified above is fulfilled, a directed edge is added from P1 to P2 as a communication channel. It contains all data objects that shall be sent from P1 to P2. It is assumed that all data objects are exchanged with the same frequency between the partitions and the communication costs between two partitions in one direction are defined as the number of different data objects. Figure 7 shows the communication graph of the model given in the last section.

FPGA-BASED CO-SIMULATION

Conversion to VHDL Code

In order to execute a MATLAB/Stateflow model on an FPGA it must be converted to a hardware description language. After the synthesis it can then be executed on an FPGA and merged into a distributed simulation.

SF2VHD (Camera 2001) is a MATLAB M-script that uses the MATLAB API and translates graphically represented Statecharts to a textual representation in VHDL. The VHDL code is output as character string by the individual functions of an M-script. A second M-script serves for the syntax conversion of the action label into appropriate VHDL operations. A powerful language like Statecharts does not map easily into hardware. Therefore, not all possible expressions are considered in SF2VHD. An important constraint regarding Stateflow designs is that no AND decomposition is supported, i.e. states can only have XOR decompositions. In JVHDLGen, that was developed in Java at FZI/ESM, the functionality has been extended by states with AND decomposition. JVHDLGen bases on a parser that is capable to parse MATLAB (mdl) files. The objects are converted to Java objects.

In Statecharts concurrently executed states can communicate with each other by broadcast or common data. In VHDL concurrency is supported by processes. Processes can communicate through signals. In VHDL'93 the data type `shared variable` can be used for communication as well as the instruction `wait` for synchronizing processes. An alternative method is putting concurrent states into one process that is executed in one clock cycle.

An enumeration type is defined for each concurrent state. The possible values of this type are the names of the basic states. A virtual state is additionally added, in order to represent the source state of the default transition. A state inside the virtual state means that this state is inactive, too. Since the history junction is not implemented, yet, all default transitions of the concurrent states are executed when entering the appropriate state. When leaving the state with AND decomposition all concurrent substates are set to the respective virtual state. Thus, with the next entry to the state with AND decomposition, the default transitions are always executed. In order to easily identify state names, the names of all of its super states are added as a prefix. In figure 8 an example Statechart is shown that can be converted to VHDL code by JVHDLGen.

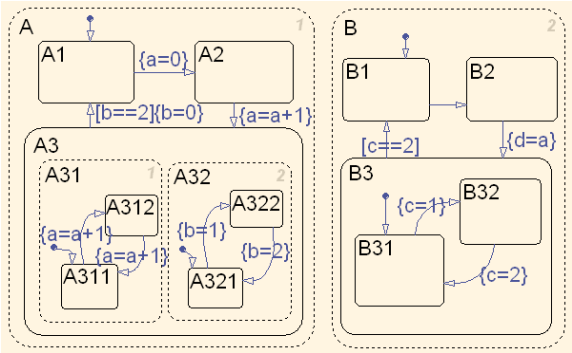

Figure 8: Chart with Nested AND Decomposition

Development Environment and Tools

The rapid prototyping system RP.2002 (Kühl 2002) is used as an evaluation platform for the development of FPGAs as well as FPGA Advantage of Mentor Graphics. The CORE generator of Xilinx supplies adjustable COREs which are optimized for Xilinx FPGAs. A feature of the RP.2002 is the extensive use of standard hardware (COTS) components. This does not only lead to low costs, the system can be also easily integrated into an existing network and is suitable for distributed simulation. A backplane integrates a PCI controller and an FPGA and allows the access of external actuators and sensors. As operating system Linux is used together with the real-time application interface RTAI.

By means of special blocksets, such as SystemGenerator (Xilinx) and AccelFPGA (AccelChip) as well as the tool JVHDLGen VHDL code can be generated from MATLAB/Simulink/Stateflow models for the configuration of an FPGA. AccelFPGA requires fixed point MATLAB or Simulink files as input. The model either has to be developed in this form or must be transformed to it manually. MATLAB models offer the opportunity to do this automatically by executing an auto quantize command. Additionally, compiler directives must be given to specify which part of the model shall be converted to VHDL. They provide model specific knowledge for the compiler and offer the opportunity for optimization. AccelFPGA generates a VHDL description on register transfer level out of the modified Simulink or MATLAB design, which can then be synthesized and simulated. MATLAB/Stateflow blocks are not supported. Similar limitations exist for SystemGenerator. Therefore, the tool JVHDLGen is used, together with the tools from Mentor Graphics and Xilinx.

JStateSim allows a distributed discrete event simulation of Stateflow charts, that are similar to Harel Statecharts (Harel 1987), analogue to MATLAB/Stateflow, but additionally offers the possibility for a distributed simulation using conservative or optimistic protocols (Schmerler et al. 1997, Fujimoto 2000). It is implemented in Java and is for that reason platform independent. The communication is realized using RMI. Simulator instances can be coupled as many as desired.

Transition notations are restricted to the following formal description:

- `if or if/then`
- `if := [condition] or event`
- `then:= (event)* and/or (equation)* (separated by ";")`
- `equation := VK ° VK`
- `VK := variable or constant`
- `° := + or –`

In order to use JStateSim as part of a distributed simulation with hardware acceleration, a part of the simulator was modified so that the hardware is completely hidden from the other simulators and the control process. In this case the simulator does not get a model as input, since it is loaded while the FPGA is configured. Otherwise the behavior is equivalent to that of an instance of the simulator of JStateSim, but can only be started with according hardware (FPGA) and software (library for accessing the FPGA). The communication between the object instances of JStateSim as well as the CPU on the FPGA rapid prototyping system with the other simulators that are involved in the simulation is taken over by Java RMI.

Simulation Structure and Design Flow

The simulation structure is shown in figure 9. The communication between the C interface of MATLAB (S function) and Java is done using the Java Native Interface (JNI). A control process communicates with the RMI part of the simulators that are implemented in Java.

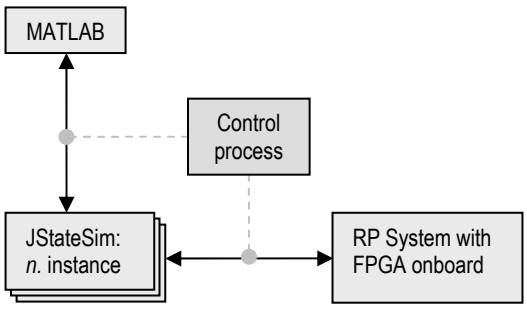

Figure 9: Simulation Structure

A bidirectional connection between MATLAB and the RP system is also possible since the interface is the same as between MATLAB and JStateSim. The JNI is used as an interface between Java and the hardware as well as for the communication between JStateSim and MATLAB. It allows Java code running in a Java Virtual Machine (JVM) to operate with libraries written in C/C++.

The VHDL design flow (figure 10) is divided into three fields: creation, implementation and verification. The starting point of each design is its specification. In VHDL this could be done on abstraction layers like algorithms and function blocks or converted to from a non-formal description. Already on this layer the correctness of the design should be tested by logic simulation. This high level description is translated to the register transfer level (RTL) and handed over to the synthesis tool. The supported language by this tool is the first of the dependencies appearing in the tool chain. The synthesis itself converts the VHDL RTL description to a mostly technology independent logic description, e.g., by using special features of a certain FPGA; even the RTL code is not target independent. This step is followed by place and route, which maps the design to the technology of a certain vendor. At this point, precise timing information is available in SDF (standard delay format) and a timing simulation is possible to verify whether all constraints are met. If an FPGA is the target platform, a bitstream for its configuration is generated. The modules are placed on the FPGA and are interconnected.

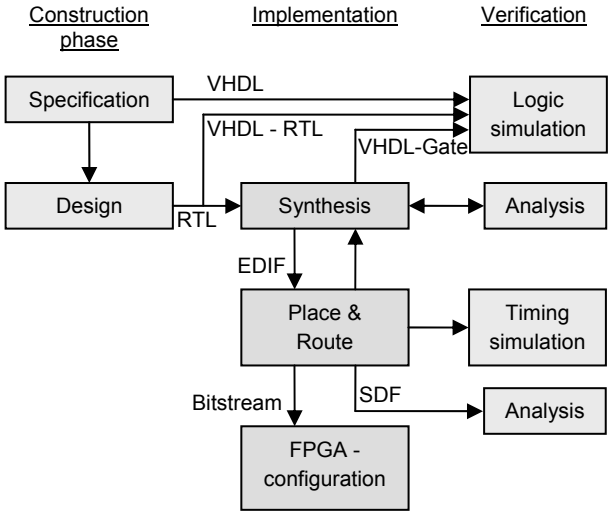

Figure 10: VHDL Design Flow

Implementation of the Interfaces

A JNI shared library provides an interface to the part of the simulator running on the FPGA. It translates and forwards the messages from JStateSim to the kernel module. Therefore, two FIFOs are installed for communication purposes and cleared at the end of a communication step. A kernel module sets up the communication between the simulator and the FPGA and uses the functionality of the PCI-driver and the FIFOs. The kernel module offers functions for the initialization and cleanup which are executed after loading and before unloading the module. The other functions provide read and write access to the dual port RAM and commands like model reset and execution of a simulation step. These recurrent operations are implemented at this low level to reduce the communication overhead. The bus interface is the link between the local bus, the part of the simulator running, and the FPGA. Basically, it consists of a clock former and a dual-port RAM, instantiated using the Xilinx CORE generator. This type of RAM is used because the local bus and the simulator have to read and write simultaneously to the RAM.

Control Unit

The control unit (figure 11) makes data available and controls the MATLAB model. Its design should be flexible and serve models without significant changes as much as possible. The simulated model is attached to the control unit. By default, it offers a clock input as well as a synchronous reset and enable input. Data inputs and outputs are dependent on the particular model. The control unit is the part of the simulator implemented in hardware. Corresponding to the data stored in the RAM it provides control signals and data to the simulated model. It is composed of different states. The initial state `wait` reads the control register (address: 0x00) until bit 0 changes. In this case state `lock` sets bit 3 in the control register to prevent the software from reading invalid values or sending new commands to the control unit. Afterwards the hierarchical state `read` reads the variables and hands them over to the model. Also, a maximum simulation time is read from a certain register in the RAM. The state `exc` simulates the model for one time step and increases the time by one. The state `continue` decides if the simulation can be carried on or not. The break conditions are the simulation time and modified variables. Afterwards the state `write` writes back the results and the simulation time. After resetting bit 3 in state `unlock` the control unit returns to the initial state and starts a new cycle. The structure of the control unit arises from the requirements of the distributed simulation and the model. JStateSim requires the control unit to simulate the model to a certain time. If the output variables change, the simulation must be canceled and the new values and time stamp must be handed over to the control process. In order to describe these requirements in VHDL finite state machines are a recommended choice. The state actions and conditions are VHDL statements. The encircled numbers at the beginning of the outgoing transitions `wait` and `continue` represent the priorities. In figure 11 only one variable is checked exemplarily. The decision to write the status of the control unit in a status bit can be replaced in the future by an implementation of an interrupt mechanism for the FPGA on the RP.2002.

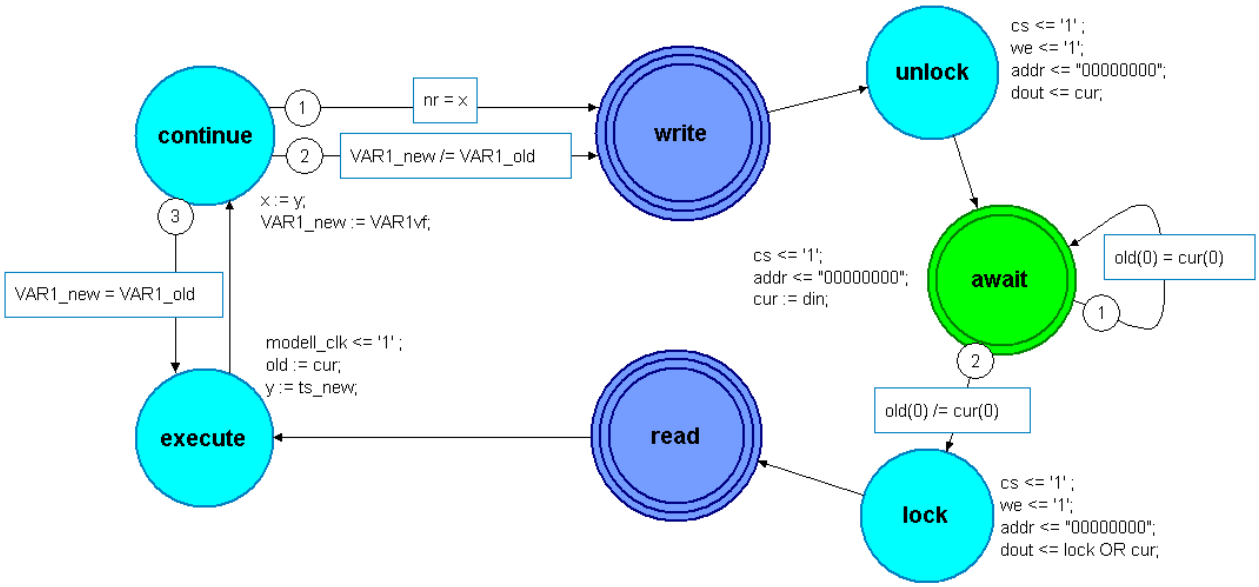

Figure 11: Control Unit

CONCLUSION

A platform independent co-simulation framework has been presented that allows for simulation acceleration by emulating a system using FPGAs. A requirement for executing a co-simulation are appropriate partitioned models, so that each participant is as efficient as possible, and the additional communication overhead induced by distributing a model remains as small as possible. Therefore, a proceeding for the partitioning of Statecharts has been suggested that integrates proven partitioning algorithms for logic circuits. Another way to increase the simulation speed is using dedicated simulators like JStateSim with optimistic synchronization protocols as an option. Since the framework has been designed in a flexible way, it can be extended to support also continuous systems. Both will be investigated in the future as well as the supported Statechart features of JVHDLGen extended.

REFERENCES

Camera, K. 2001. "*SF2VHD: A Stateflow to VHDL Translator*". Master Thesis, Department of Electrical Engineering and Computer Science, R. Brodersen, University of California, Berkeley, California, USA.

Chen, Y.; V. Jha; and R. Bagrodia. 1997. "A Multi-dimensional Study on the Feasibility of Parallel

Switch-Level Circuit Simulation". *11th Workshop on Parallel and Distributed Simulation PADS '97*.

Cong, J.; L. Zheng; and R. Bagrodia. 1994. "Acyclic Multi-Way Partitioning of Boolean Networks". *Design Automation Conference 1994*, 670-675.

Dreier, R.; E. Sax; and K.D. Müller-Glaser. 2001. "Requirements and State of the Art of Automated Software Development for Embedded Systems Based on CASE Tools". *Proc. of IEEE Design, Automation and Test in Europe Conference 2001*, Munich, Germany, 44-48.

Fiduccia C. and R. Mattheyses. 1982. "A Linear Time Heuristic for Improving Network Partitions". *ACM/IEEE Design Automation Conference 1982*, 175-181.

Fujimoto, R.M. 2000. "*Parallel and Distributed Simulation Systems*". John Wiley & Sons, Inc.

Harel, D. 1987. "A Visual Formalism for Complex Systems". *Science of Computer Programming*, No. 8, 1987.

Kernighan, B. and S. Lin. 1969. "An Efficient Heuristic Procedure for Partitioning Graphs". *The Bell System Technical Journal*, 1969, 291-307.

Krishnamurthy, B. 1984. "An improved min-cut algorithm for partitioning VLSI networks". *IEEE Transactions on Computers*, 1984, 438-446.

Kühl, M. 2002. "*Rapid-Prototyping System RP.2002*". developed at the Institute for Information Processing Technology, University of Karlsruhe, Karlsruhe, Germany.

Lengauer, T. 1990. "*Combinational Algorithms for Integrated Circuit Layout*". John Wiley & Sons, Inc.

Mehl, H. 1994. "*Methoden verteilter Simulation, Vieweg-Verlag*". Braunschweig, Germany.

Sanchis, L. 1989. "Multiple-Way Network Partitioning". *IEEE Transaction on Computers*, 1989, 62-81.

Schmerler, S.; Y. Tanurhan; and K.D. Müller-Glaser. 1997. "Predictive Time Warp". 11th *European Simulation Multiconference ESM '97*, Istanbul, Turkey.

Stingl, T. and R. Dreier. 1999. "Modellierung und Simulation mit Zustandsautomaten - Schwerpunkt BetterState". *ASIM Fachtreffen 1999*, Aachen, Germany.

AUTHOR BIOGRAPHIES

Rico Dreier was born in Sinsheim, Germany. He received the Dipl.-Ing. degree in electrical engineering in 1997 from the Technical University of Karlsruhe, Germany. Since 1998 he works at the FZI Research Center for Information Technologies, Dept. of Electronic Systems and Microsystems. He participated in projects in the area of MPEG-4 standardization, CASE methods and analysis of real time operating systems. His research interests include simulator coupling and distributed simulation of electronic systems.

Georg Dummer was born in Wittenberg, Germany. Since 1997 he studies computer science at the University of Karlsruhe, Germany. During the study he focused on the design of embedded systems. He is now completing his degree with a diploma thesis.

Guoxing Zhang was born in Shanxi, China. He received his BS degree in electrical engineering from the Tongji University in Shanghai in 1991. 1999 he joined the company NEC in Tianjin, China. Since 1999 he studies computer science at the University of Karlsruhe, Germany, and is now working on his diploma thesis.

Klaus D. Müller-Glaser received the Dipl.-Ing. and Dr.-Ing. degree in 1972 and 1977 from the University of Karlsruhe, Germany. From 1977 to 1986 he worked for Siemens AG, Synertek Inc., Honeywell Inc. and Bell Labs, Allentown, PA, before he became responsible for setting up the first commercial U.S. AT&T ASIC Design Center in Sunnyvale, CA. In 1986 he was appointed Full Professor at the University of Erlangen-Nürnberg, Germany, in April 1993 he became Full Professor and Director of the Institute for Information Processing Technologies, Department of EE and IT, University of Karlsruhe. He is a Director of the Computer Science Research Center (FZI) in Karlsruhe. From 1996 till 2002 he served as president of FZI, currently he is the dean of the department.