

Modeling and Formal Verification of Hardware Designs

Niusha Hakimipour Niloofar Razavi Marjan Sirjani

Department of Electrical and Computer Engineering
University of Tehran, Karegar Ave., Tehran, Iran

n.hakimi@ece.ut.ac.ir n.razavi@ece.ut.ac.ir msirjani@ut.ac.ir

Abstract

We investigate applying of an actor-based language, Rebeca, for hardware design. Rebeca is based on reactive objects with formal foundation. Hence, available tools for model checking provide us with formal verification support. In our approach, system design process is started from Rebeca in high levels of abstraction. Formal verification is then used to verify the design. We show the process of system design using Rebeca. A case study is used to show the process of modeling and verification.

Keywords: Hardware Design, Formal Verification, Model Checking, Actor Model, Rebeca.

1. Introduction

Integrating heterogeneous components and increasing complexity of microelectronic systems demand an appropriate increase in the level of abstraction in designing these systems. System-level design languages are used to start the design process from a more abstract level and apply a top-down design methodology.

Using a uniform system design language in all the stages of design process has multiple advantages, namely, the decision concerning hardware/software partitioning may be deferred to later stages, and formal verification will be possible in the earlier stages. This will decrease the probability of redesign and its time and cost overhead.

The need for a uniform and abstract design language leads us to the methodologies which are used in software development. Object-oriented modeling has been the most successful approach in software development. Different works have been done on designing System-on-Chip (SoC) using various extensions of UML [1, 2, 3]. Also, by using the various extension of C++ like SystemC [4] the designer may use the object oriented paradigm in design. Other alternative approaches are to extend

classical hardware description languages, or create new specific languages for system level design.

Existing approaches and tools for verifying system designs mainly apply non-formal approaches [5, 6] or handle only low-level specifications [7, 8, 9]. But Formal verification is getting more and more attention as a technique to verify the hardware/software designs sufficiently [10, 11].

Here, we show that the simple yet powerful event-driven computational model of the actor-based [12] language, Rebeca (*Reactive objects language*) [13, 14], is suitable for modeling systems, and its model checking tools and techniques can be used for formal verification in different stages of design.

Rebeca is a Java-like modeling language for developing object-based concurrent and distributed systems with a formal foundation. A model in Rebeca consists of a set of concurrently executing reactive objects, called rebecs (*reactive objects*). Rebecs have no shared variables and communicate asynchronously via non-blocking message passing. The Rebeca verifier tools, as front-end tools, translate Rebeca code into languages of existing model checkers, (SMV and Spin), allowing verification of the required properties [15, 16]. There is also an ongoing project on developing a direct model checker for Rebeca using state space reduction techniques [17, 18].

The main motivation in designing Rebeca was to provide an object-based language with clearly defined encapsulated units of concurrency and hence, providing a natural modular design approach, with loosely coupled modules, which makes the model suitable for applying compositional verification techniques.

Software systems typically have an asynchronous model of execution, while hardware is usually designed for synchronous execution. In the higher levels of abstraction in system-level design we generally have asynchronous models. Modeling the systems using Rebeca in the higher levels of abstraction, like the functional specification, is a natural process. So, in this work we mainly elaborate

on the modeling process for the hardware design. Rebeca is a very simple language to learn and allows the designer to start from a higher level visual design. A case study is used to show the applicability of our approach. Performance evaluation is not considered in our method.

Contribution. There are languages for system design but to the best of our knowledge formal verification of high level designs in these languages is not yet provided. Using Rebeca, we provide a language with formal verification support for system design. Here, we investigated the power of the computational model of Rebeca, for modeling hardware systems. We found it natural in higher levels of abstraction as expected and applicable in lower levels, and surprisingly very close to the behavior of real hardware components.

Outline of the paper. In the next section we have a brief overview of the related work. Section 3 explains the modeling method using Rebeca and Section 4 shows verification issues. In Section 5 a round robin arbiter is used as our case study. A short conclusion and a view of our future work are described in Section 6.

2. Related Work

There are works done on formal verification of hardware/software designs but they generally can be applied on lower levels of design.

A valuable work is presented in [7] for combining software partial order reduction and hardware BDD-based symbolic model checking techniques. In this approach, efficient techniques of model checking are put together and applied to the implementation level models.

In [10] a formal semantics for SystemC is provided and used for automatic hardware/software partitioning and simplifying formal verification problem. The authors mentioned that using uniform system-level design language keeps the verification flow integrated.

In [11] formal verification techniques targeting C-based System-on-Chip (SoC) design descriptions are discussed. Formal verification of synchronization mechanisms of a SpecC program and equivalence checking are presented.

The authors of [9] introduce a bilingual specification environment consisting of two languages SDL and S/R for modeling software and hardware respectively. The whole software/hardware co-design is converted to S/R language which can then be model checked. A

uniform language for both hardware and software is not used and formal verification can only be applied on the lower level language.

In [5], the authors presented their environment for hardware/software co-verification in C/C++ using SystemC. They suggested that modeling hardware and software in the same language together with a good object oriented design technique will make moving functionality between software and hardware easier. Their verification approach is not formal.

UML notation has achieved the status of a standard language and is applicable to wide variety of systems. The following works use UML for system level design and provide no formal foundation or verification. A UML profile for SystemC is presented in [1]. The authors conclude that UML is applicable in a wider domain area, such as SoC. A system level description mechanism based on UML is presented in [3]. The SystemC code which implements these designs can be automatically generated. In [2] a design flow to develop clocked hardware circuits using UML is outlined where statecharts and component diagrams are used.

The above mentioned works are similar to our work in one or more of the following features: formal verification support, unified language for software and hardware and different abstraction levels, object oriented approach, but none of them provides all. Also, as mentioned, the computational model used in our method is different.

A similar work is done in [19] where the authors propose an approach for design and verification of transaction level models. In their approach, they start from UML for designing a system in high level of abstraction. Then the UML model is translated to a Rebeca model which is then verified. They proposed some rules for automatic mapping of Rebeca models to SystemC transaction level models.

In this work, we may start from Rebeca models. We elaborate more on Register Transfer Level (RTL) designs as using Rebeca in high level designs is more natural. In this paper we focus on modeling hardware by Rebeca and provide the details of our approach. We elaborate how synchrony and asynchrony can be modeled by Rebeca. We also provide an optimization in our approach which can significantly reduce the size of the state space of circuits.

3. Moving through System Design Using Reactive Objects

Modularity and abstraction are used to tackle the rising complexity. Abstraction helps to defer introducing the details, and modularity helps to divide

a complex problem into simpler modules. A model which can be consistently used in all levels of abstraction increases the flexibility and reusability in design. Also, a model which leads to cohesive and decoupled modules increases maintainability and reusability.

In our method, object-based techniques and tools are used. The method also leads to the selection of modules according to the object-oriented paradigm. In this section we first introduce Rebeca and then describe the corresponding mappings from system components and their behavior to Rebeca constructs, namely reactive objects, message servers, state variables, known objects and message passing.

3.1. Rebeca Model

Rebeca models consist of concurrently executing reactive objects, called rebecs. Rebecs are encapsulated objects, with no shared variables, which can communicate via nonblocking asynchronous message passing with no explicit receive. Each rebec is instantiated from a *reactive class* and has a single thread of execution; it also has an unbounded buffer, called a queue, for arriving messages. Each message specifies a unique method to be invoked when the message is serviced. When a message at the head of a queue of a rebec is serviced, its message server is invoked and the message is deleted from the queue. During the execution of a message server, assignments and selections may take place and messages may be sent to other rebecs or to self. Each rebec knows other rebecs to which it sends messages as its *knownobjects* and has a message server named *initial* which is executed once at the beginning of the model execution.

Each Rebeca model has a part named *main* which includes the rebec instantiations and known objects relations.

3.2. Modeling Systems Using Rebeca

Hardware components run concurrently and react to the changes in their input by changing the output. This behavior naturally fits into the concept of *reactive objects*. So, each hardware component, sequential or combinational, is mapped to a rebec. Each input of a component is mapped to a state variable and a message server in the corresponding rebec. The state variable contains the value of the input and the message server is responsible to model the reaction of the component to a change in the input. The outputs of a component are provided by sending appropriate messages to the driven components (if an output of component *A* is an input

of component *B*, we call component *A* the driver and component *B* the driven component).

In the rebecs which corresponds to sequential components, there is an additional message server reacting to the clock edge and sends messages according to its outputs.

In the rebecs which corresponds to combinational components, in each message server related to the inputs, after setting the corresponding state variable the logic of the circuit is coded, and then the messages for outputs are sent.

The data path of a design is mapped to the *main* part of the Rebeca model which includes the rebec instantiations and known objects relations. Figure 1 depicts a simple conversion relation between the hardware design concepts and Rebeca constructs.

Hardware Concept	Rebeca Construct
Hardware Design	Rebeca Model
Hardware Component	Reactive Class
Component Instance	Rebec
Input of a Component	A Message Server and A State Variable in the Corresponding
Driven Component	Known Object in the Corresponding Reactive Class
Changes of Outputs	Sending Appropriate Messages to Rebecs related to the Driven

Figure 1. Conversion of Hardware to Rebeca

In order to model sequential circuits in which hardware components are synchronized by edges of a global clock signal, we need to introduce two additional rebecs in our model to perform the synchronization and clocking of the synchronous devices.

Clock rebec: has one message server in which it sends a clock message (called *AtClkEdge*), representing the positive edge of the clock signal, to all sequential rebecs. All the sequential rebecs have a send message to the *Synchronizer* rebec in their *AtClkEdge* message server, indicating the receipt of the clock.

Synchronizer rebec: has one message server in which it keeps the track of sequential rebecs (called *EndClkCycle*), and when all the messages from the sequential rebecs are received a message is sent to all of them to fire their outputs. The Synchronizer rebec ensures that all the sequential components are working synchronously, and that the next *AtClkEdge*

will not be sent before all the rebecs have fired their outputs, and in the case of a driven combinational component the circuit has reached a stable situation. In mapping a hardware circuit into a Rebeca model, an arbitrary delay for all the circuit components is modeled by the queue which holds the messages until they are processed. The *Clock* rebec can also be used to count the clock signals, and provide us with the ability to prove properties which include conditions on the number of clock signals.

3.3. High Level Designs

As mentioned, using Rebeca for modeling and verification of higher levels of abstraction when the design consists of asynchronous components is natural. Later in the design cycle, designer modifies the model and partitions the hardware and software, and specifies the interfacing logic. The advantage is that the designer is not forced to partition the design to perform the verification.

4. Formal Verification Issues

By using Rebeca language the hardware designs can be model checked. But we need techniques to overcome the state space explosion problem, one of the most challenging problems in the field of model checking. Compositional verification and abstraction techniques, weak simulation relation, symmetry and partial order reduction are applied on Rebeca models and are explained in [13, 18] which can be similarly used here. Some techniques which are specific for hardware design are proposed in the following.

An Optimization in the Proposed Approach In a combinational component, any change in the input is immediately transmitted to the output which may lead to multiple changes in output per clock. In synchronous circuits, the state of the outputs only on the edge of the clock is important.

When a combinational component is modeled as a rebec, presence of all the transient states which are not significant will cause state explosion. To avoid this, we apply an optimization technique and migrate the logic of each combinational component to the next sequential components.

When there is more than one next component, the logic is copied in all of them. We use a simple example to show the optimization technique. Figure 2, shows a pipeline. The pipeline consists of several registers (pipe registers in the sample) interconnected by combinational components (Adder and Multiplier). Changes in the register values are

propagated through the combinational components immediately. But sequential components (pipe registers) hold their output until a clock signal is issued.

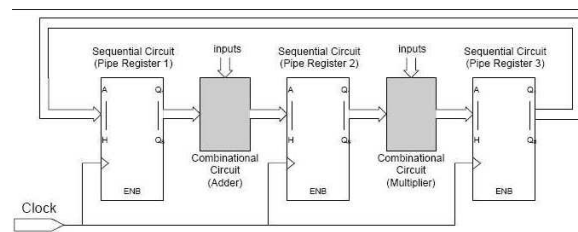


Figure 2. A Pipeline Design (to be Converted to Rebeca Model)

According to the proposed method, this hardware can be converted to a Rebeca model, which is illustrated in Figure 3. In this figure, rectangles represent rebecs, and the arrows show message delivery direction. After applying the optimization technique we will have a Rebeca model which is shown in Figure 4.

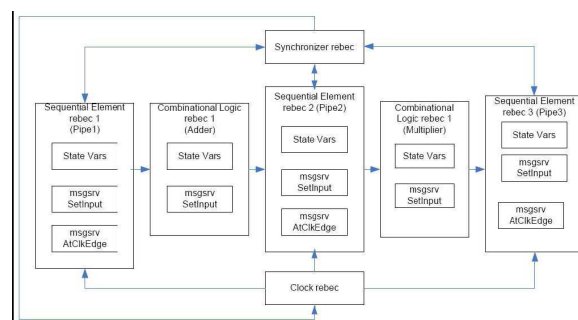


Figure 3. Rebeca Model of Pipeline in Figure 2 (Before Optimization)

The mentioned optimization technique can be applied automatically, and reduces the size of the state space of Register Transfer Level (RTL) circuits significantly.

Further optimizations are possible, like combining the sequential circuits together as a single rebec, which cannot be performed automatically, and the designer may decide for that to reduce the state space.

Formal Verification of Open Systems. In the case of open systems, we introduce an *environment* rebec to Rebeca model. The *environment* is responsible to model the different input values for each port. These values are non-deterministically chosen from the defined domain of each port and are changed once per clock cycle. For each output message to the environment a state variable is added to the driver

rebec to provide us the values of the outputs for the verification purposes.

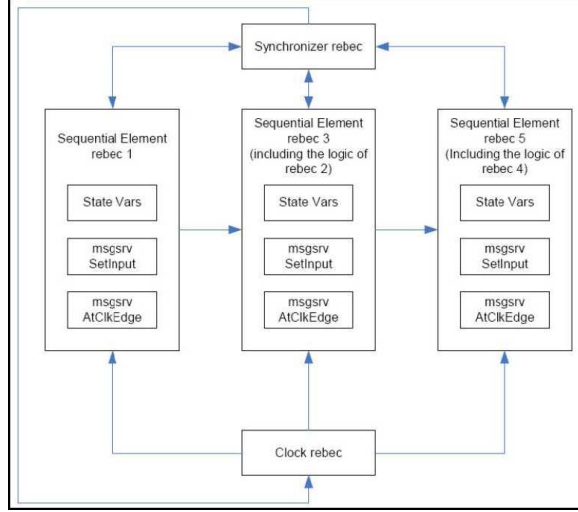


Figure 4. Rebeca Model of Pipeline in Figure 2 (After Optimization)

5. Case Study: Round robin Bus Arbiter

A round robin bus arbiter is chosen to show the applicability of our proposed method. A bus arbiter is a hardware component that determines which device is allowed to use a shared bus when multiple requests for the bus are present. A round robin bus arbiter switches the device priorities in order to give all the devices the chance to use the bus, even in the presence of a heavily loaded device.

The bus passes the requests to the arbiter and the arbiter sends the proper grant back to the bus which in turn passes it to the devices. Then the corresponding device is granted the bus for one cycle. We did our design starting from Rebeca according to the proposed method, considering the optimization issues for verification purposes. Then we verified the design using Rebeca verifier tools.

5.1. Verifying Round Robin Bus Arbiter

We use temporal logic [20] as our property specification language. A *temporal formula* is constructed out of *state formulas* (assertions) to which we apply Boolean connectives and temporal operators. State formulas are propositions defined over standard operations and relations over the set of state variables. We naturally do not consider the message queue contents in our state formulas.

System Properties. There are four properties which we want to prove for the round robin arbiter example. These properties are based on the state of the arbiter and the devices connected to it. The properties are as follows (where \square , \diamond and X mean *always*, *finally* and *next* respectively):

Safety Only one device shall have the grant to use the bus at the same time:

$$\square \left(\bigwedge_{i=0}^4 \left(\bigwedge_{j=0}^{4, j \neq i} (\overline{busy_i \wedge busy_j}) \right) \right)$$

Progress: no deadlock This property is checked directly by specifying the appropriate option in the model checking settings in Spin model checker.

Progress: no starvation If any of the connected devices requests the bus arbiter for a grant to use the bus, the device will finally receive the grant:

$$\square \left(\bigwedge_{i=0}^4 (waiting_i \rightarrow \diamond (busy_i)) \right)$$

No false grants The arbiter should not grant any device that has not requested to use the bus:

$$\square \left(\bigwedge_{i=0}^4 (\overline{waiting_i \wedge busy_i} \rightarrow X \overline{busy_i}) \right)$$

We used Rebeca verifier tools [15, 16], to translate our Rebeca models. Then we model checked the code to see if the properties are satisfied. The results of the model checking show that the properties are satisfied.

6. Conclusion and Future Work

As the size and functional complexity of the digital systems increase, higher levels of abstraction in designing these systems become inevitable. Also, formal verification of such designs become more challenging and yet necessary.

Here, we propose to use the computational model of the actor-based language Rebeca to have a unified language in design process and also gain its formal verification support.

As SystemC is popular for system design, in another work we are providing a detailed mapping between Rebeca and SystemC. We are also working on adding timing issues to the model. Tools shall be created for the mappings in the modeling part and some of the hardware specific optimization issues.

References

- [1] Riccobene, E., Rosti, A., Scandurra, P.: Improving SoC design flow by means of MDA and UML profiles. In: ACM Symposium on Applied Computing (SAC). (2004)

- [2] Sun, Z., Wong, W.F., Zhu, Y., Pilakkat, S.K.: Design of clocked circuits using UML. In: the Asia and South Pacific Design Automation Conference (ASPDAC). (2005)
- [3] Nguyen, K.D., Sun, Z., Thiagarajan, P.: Modeldriven SoC design via executable UML to SystemC. In: Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS). (2004)
- [4] Grotker, T.: System Design with SystemC. Kluwer Academic Publishers (2002)
- [5] Semeria, L., Ghosh, A.: Methodology for hardware/ software co-verification in C/C++. In: Proceedings of the 2000 conference on Asia South Pacific design automation. (2000)
- [6] Kudlugi, M., Hassoun, S., Selvidge, C., Pryor, D.: A transaction-based unified simulation/emulation architecture for functional verification. In: DAC 2001, Las Vegas, Nevada, USA. (2001)
- [7] Kurshan, R.P., Levin, V., Minea, M., Peled, D., Yenigun, H.: Combining software and hardware verification techniques. Formal Methods in System Design 21(number) (2002) 251–280
- [8] Chauhan, P., Clarke, E.M., Lu, Y., DongWang: Verifying IP-Core based system-on-chip designs. In: IEEE ASIC Conference. (1999)
- [9] Levin, V., E.Bounimova, Basbugoglu, O., Inan, K.: A verifiable software/hardware co-design using SDL and Cospan. In: COST 247 International Workshop. (1997)
- [10] Kroening, D., Sharygina, N.: Formal verification of SystemC by automatic hardware/software partitioning. In: Proceedings of MEMOCODE, IEEE (2005) 101–110
- [11] Fujita, M.: Formal verification of higher-level SoC designs in C-based descriptions. In: International Conference on Dependable Systems and Networks (Workshop on Model-Checking for Dependable Software-Intensive Systems). (2003)
- [12] Agha, G.: The structure and semantics of actor languages. In de Bakker, J.W., de Roever, W.P., Rozenberg, G., eds.: Foundations of Object-Oriented Languages. Springer-Verlag, Berlin, Germany (1990) 1–59
- [13] Sirjani, M., Movaghar, A., Shali, A., de Boer, F.: Modeling and verification of reactive systems using Rebeca. Fundamenta Informatica 63(4) (Dec. 2004) 385–410
- [14] Sirjani, M., Movaghar, A.: An actor-based model for formal modelling of reactive systems: Rebeca. Technical Report CS-TR-80-01, Tehran, Iran (2001)
- [15] Sirjani, M., Movaghar, A., Shali, A., de Boer, F.: Model checking, automated abstraction, and compositional verification of Rebeca models. Journal of Universal Computer Science 11(6) (2005) 1054–1082
- [16] Sirjani, M., Shali, A., Jaghoori, M., Iravanchi, H., Movaghar, A.: A front-end tool for automated abstraction and modular verification of actor-based models. In: Proceedings of Fourth International Conference on Application of Concurrency to System Design (ACSD'04), (IEEE Computer Society, 2004) 145–148
- [17] Jaghoori, M.M., Movaghar, A., Sirjani, M.: Modere: The model-checking engine of Rebeca. In: ACM Symposium on Applied Computing - Software Verificatin Track. (2006)
- [18] Jaghoori, M.M., Sirjani, M., Mousavi, M.R., Movaghar, A.: Efficient symmetry reduction for an actor-based model. In: 2nd International Conference on Distributed Computing and Internet Technology. Volume 3816 of Lecture Notes in Computer Science. (2005) 494–507
- [19] Kakoei, M. R., Shojaei, H., Sirjani, M., Navabi, Z.: A New Approach for Design and Verification of Transaction Level Models. In: IEEE International Symposium on Circuits and Systems (ISCAS'2007)
- [20] Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, Berlin, Germany (1992)