Heterogeneous Model Merging Based on Model Transformation

Hongtian Ma, Hehua Zhang, and Ming Gu

***Abstract*—The system design and development of embedded software is under a lot of challenges. Model-based software systems are drawing more and more attentions. In our previous work we proposed a system level design language named SyncBlock and developed a toolset for the design of synchronous embedded system. Although our toolset is superior for building synchronous models, it is difficult to model the external environment of embedded system ideally, and does not support asynchronous modeling functionalities or the merging of heterogeneous models of computation. Ptolemy II is a well-known modeling platform which supports many well-defined heterogeneous models of computation. In this paper we propose a series of rules and mechanisms on model transformation from SyncBlock to the SR model of computation in Ptolemy II for heterogeneous model merging. Using our method, we can model and simulate synchronous embedded systems by SyncBlock, and then simulate the designed model further coupling with external environment modeled by other models of computation in Ptolemy II like Discrete Event Domain, and finally generate codes by the SyncBlock modeling tool. Through heterogeneous model merging by model transformation, we combine the advantages of the two modeling tools.**

***Index Terms*—Heterogeneous model merging, model transformation, simulation, SyncBlock, Synchronous Reactive model.**

I. Introduction

In order to address the challenges in modeling, simulation and implementation of the applications of synchronous embedded systems, we have designed a synchronous modeling language SyncBlock [1], which is a system level language based on automata and block diagrams. It has several advantages. Both the operational semantics and formal semantics of SyncBlock are precisely described, so that it can be used for modeling, simulation, verification and code generation. In SyncBlock, a system is modeled as a combination network of compound blocks and atomic blocks. SyncBlock supports hierarchical decomposition and concurrent process execution. Furthermore, code generation to both hardware, e.g., VHDL, and software, e.g., C is well supported, which can enable automatic generation of the efficient implementation of SyncBlock models. SyncBlock is superior in modeling of synchronous systems, but it can hardly model asynchronous systems and the external environment of embedded systems. On the other hand, Ptolemy II [2], [3] defines many models of computation like Synchronous Reactive (SR), Discrete Event (DE), and synchronous dataflow (SDF) etc. [4] It is effective for modeling the external environment and equipment information.

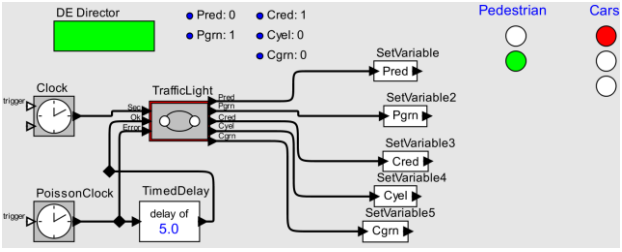


Fig. 1. A simple traffic light model in Ptolemy II.

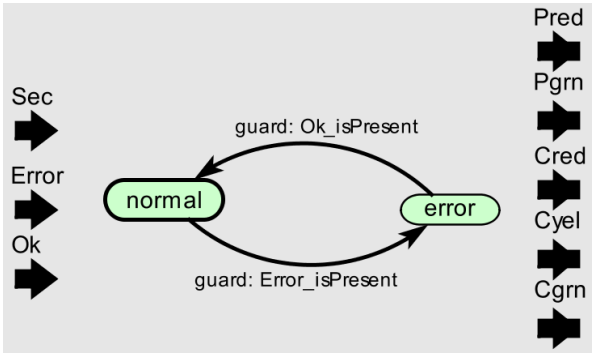


Fig. 2. Top-level model of the traffic light controller.

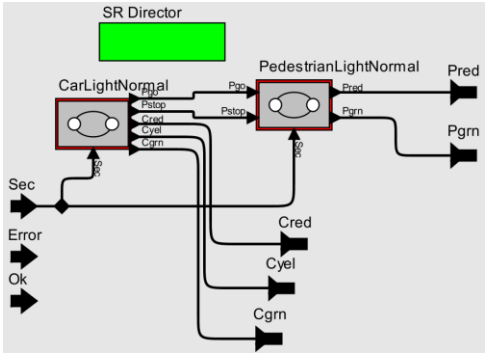


Fig. 3. The implementation of the normal state.

Let us introduce an example to denote the faithfully simulating of embedded systems by heterogeneous model fusion in Ptolemy II. Fig. 1. [5] depicts a model of traffic light. This model illustrates a typical design pattern where the top level is a DE model of the physical environment for a system

Manuscript received August 28, 2015; revised December 23, 2015. This research is sponsored in part by NSFC Program (No. 61202010, No.91218302), Tsinghua university Initiative Scientific Research Program (20131089331) and MIIT IT funds (Research and application of TCN key technologies) of China.

H. T. Ma is with the School of Software of Tsinghua University, Beijing, China (e-mail: hongtian1991@foxmail.com).

H. H. Zhang and M. Gu is with the School of Software of Tsinghua University, Tnlist, Kliss, Beijing, China (e-mail: zhanghehua@tsinghua.edu.cn, guming@tsinghua.edu.cn).

DOI: 10.7763/IJMO.2016.V6.500

under design. Opening the TrafficLight actor, we can see its implementation as presented in the Fig. 2. And looking inside the normal state, we can see the implementations as presented in the Fig. 3. The PoissonClock actor occasionally injects an Error signal. The Error condition then lasts 5 seconds, as determined by the TimedDelay actor. However, the traffic light model in SyncBlock as presented in the Fig. 4 does not contain any physical environment like the clock signal in Ptolemy II. The model should get the values of the input ports, which may be manual or file input, when a new iteration starts.

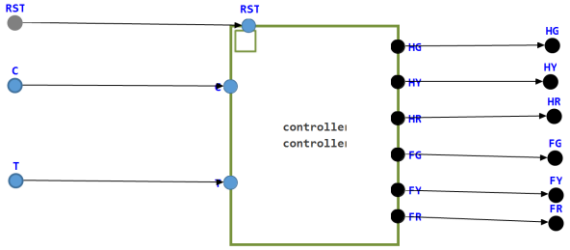


Fig. 4. TrafficLight modeled by SyncBlock.

Thus, by transforming the models to Ptolemy II, we combine the comprehensive advantages of SyncBlock with Ptolemy II's extended modeling capabilities, and the model simulation becomes more faithful for the system design.

To analyze and design synchronous embedded systems, SyncBlock is convenient and effective for modeling and simulation. It supports concurrency and code generation. However, it is infeasible to do simulation coupling with other heterogeneous modules, e.g., coupling FPGA design models with ARM programming models. After the model transformation from SyncBlock models to Ptolemy II models, the combination of simulation becomes feasible, because Ptolemy II contains models of computation, which is able to model heterogeneous modules, like DE. Through the simulation of coupling the source model and heterogeneous environment model, we can check up the completeness of the functionality and the timing correctness of the system-design model in a more faithful way. Afterwards, we use SyncBlock tool to generate reliable code, which can be synthesized and loaded into the FPGA processor directly. In the field of synchronous embedded system modeling, we put forward a complete set of solutions from modeling, simulation, validation, code generation in SyncBlock. Furthermore, we realize more faithful simulation in Ptolemy II merging the SR models and the DE models, which can model the system external information and physical environment. And we present the implementation of the transformation tool SyncToPtolemy integrated in the Tsmart platform. We can directly use SyncToPtolemy to transform the SyncBlock models to the Ptolemy II models for simulation coupling the synchronous embedded system model with the heterogeneous external environment model.

The paper is organized as follows: related work is presented in Section 2. The proposed transformation rules are presented in Section 3, including the compound blocks and atomic blocks. Section 4 presents the transformation algorithm. Case studies are given in Section V, and we conclude the paper in Section VI.

II. Related Work

A large amount of work has been contributed to reduce the complexity of the design of synchronous embedded systems. In many papers, the formal language based approach is attractive because it provides a unified basis for formal analysis to achieve expected correctness. The formal synchronous languages mainly contain Esterel [6], Lustre [7], Statecharts [8]. The Esterel and Statecharts are suitable for specifying control-oriented systems. They are easy for modeling of the complex decision control logic. But They are difficult for modeling of the signal flow and the data processing because the data communication among modules are realized by signal broadcast. The Lustre is good for specifying data-oriented systems. It is easy for modeling of the signal flow and data processing modules. But it's difficult for modeling of the decision control logic.

Based on the above system level design language, there are many corresponding tools supporting modeling, simulation of real-time, concurrent, embedded systems. The Esterel studio [9] and CEC [10] supports the design of digital systems. But the two tools are limited in code generation, many basic structures are not supported, and the simulation is not visual. The signal communication is hard to present clearly and visually. Simulink [11] is now widely used and the stateflow [12] has great modeling and simulation capability. But they have no formal semantics. Furthermore, the operational semantic of the parallel execution of automata is too complex and dependent on the relative position of the module. They are simulated in a sequence mode, from left to right, up to down. This kind of preemption is not fit for synchronous embedded system.

The Ptolemy project studies heterogeneous modeling, simulation, and design of concurrent systems. The focus is on embedded systems [13], particularly those that mix technologies, including for example analog and digital electronics, hardware and software, and electronics and mechanical devices. The focus is also on systems that are complex in the sense that they mix widely different operations, such as signal processing, feedback control, sequential decision making, and user interfaces. However, it is primarily used as a simulation environment and cannot be verified and synthesized. The semantic is based on the event queue, which may be different from the real execution of some synchronous systems.

III. Transformation Rules

The SyncBlock model is a network of blocks designed in the modeling language. The control-oriented behaviors can be captured in the parallel automata in an atomic block, and the dataflow behaviors can be captured in the data port connections among atomic blocks and compound blocks. The transformation rules from SyncBlock to Ptolemy II SR MoC is designed recursively in a top-down way. The basic elements, e.g., input and output ports, of compound blocks are transformed first, and then the communication mechanisms between sub-blocks at the same level are transformed. We then transform sub-blocks. If a sub-block is another compound blocks, we repeat this transformation

process from the first step. If it is an atomic block, the parallel automata are transformed. The advantages of our transformation strategy are: 1) the transformation process keeps the original structure of source model, which ensures the readability of the transformed model. 2) It keeps the semantics consistent with source model. The transformation rules of compound blocks and atomic blocks are presented in detail as follows.

A. Compound Block

At the top level, a compound block can be refined into several sub-blocks, each of which can be a compound block or an atomic block. Communications between sub-blocks are connected through ports. Note that the direct cycle consideration of SyncBlock and Ptolemy II SR models are different, so the semantic preserving transformation of this mechanism is one key point of the transformation rules. In the SyncBlock modeling language, the data transmission between blocks are clear and visual. The cycle is broken by the inner data communication principle that the values are updated at the end of computation in an iteration, and the updated values will be read at the start of computation in the next iteration. During the transformation, keeping the hierarchical structure is our original intention. From what has been discussed above, there are two key aspects to be manipulated.

First, we transform the block diagram of the hierarchy mapping. To preserve the original hierarchical structure of the source SyncBlock model and the readability of the transformed model, we do not flatten the hierarchical relationships in the source SyncBlock model or put all the parallel automata at one level, although this is easier than maintaining the structures. In order to achieve this goal, we use nested TypedCompositeActor in Ptolemy II to achieve the same hierarchy in the process of transformation.

Second, we transform the communications between components. The communications between sub-blocks at the same level are realized by port connections. Because of the inner data communication principle in SyncBlock that the output values are updated at the end of computation in the current iteration, and because the updated values will be read at the start of computation in the next iteration, the dataflow transfer will delay one clock cycle. When transforming the communications between sub-blocks, we add a NonStrictDelay actor as presented in Fig 5 in front of every output port of the dataflow sender so that the output value transfer will be consistent with source model which has one clock cycle delay.

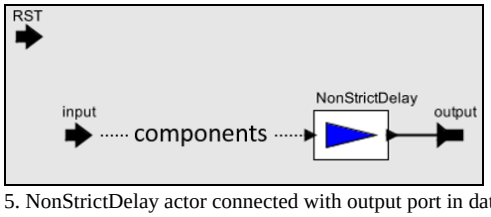


Fig. 5. NonStrictDelay actor connected with output port in data communication transformation.

B. Atomic Block

Parallel automata in atomic blocks are the most important components to express the system behavior in SyncBlock model. Thus, for the sake of equivalent conversion, we apply the parallel Modal Models in Ptolemy II to achieve the same ability of expression as parallel automata. Inside the modal model is a finite-state machine controller, and inside each state in the FSM is a refinement model. The controller is a finite-state machine (FSM) [14], which consists of states and transitions. In the process of transforming parallel automata to the modal model, three important points need to be considered specially:

First, in SyncBlock the transition actions support some basic control structure such as IF ELSE statement. For example, Fig. 6 presents two parallel automata, one of which is a self-loop with a transition whose action has some IF ELSE statements as presented in the Fig 7. However, Ptolemy II does not support IF ELSE statement.

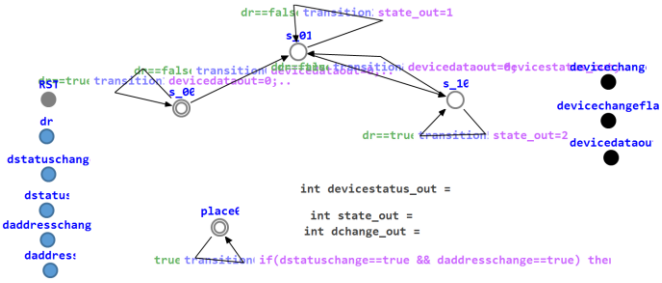


Fig. 6. two parallel automatons in SyncBlock.

```
if(dstatuschange==true && daddresschange==true) then
        dchange_out=3;
        devicestatus_out=daddress;
        devicechange=true;
        devicechangeflag=true;
else
        if(dstatuschange==true && daddresschange==false) then
                dchange_out=2;
                devicestatus_out=dstatus;
                devicechange=true;
                devicechangeflag=false;
        fi
fi

if(dstatuschange==false && daddresschange==true) then
        dchange_out=1;
        devicestatus_out=daddress;
        devicechange=true;
        devicechangeflag=true;
else
        if(dstatuschange==false && daddresschange==false) then
                dchange_out=0;
                devicechange=false;
        fi
fi
```

Fig. 7. Action of the self-loop transition.

To maintain the readability of the transformed model, we add refinement in the state which the transition's arrow points to. The basic control structure is show in the followed sheet, among BCS, statement block A, B, C, D can be empty and also be a nested BCS.

TABLE I: BASIC CONTROL STRUCTURE IN THE ACTION OF TRANSITION

Basic control structure BCS:
A
If () then
B
else
C
fi
D

The basic control structure can be refined as presented in the Fig 8 in Ptolemy II's FSM , and if A, B, C, D also contain a BCS , we would perform recursive refinement .

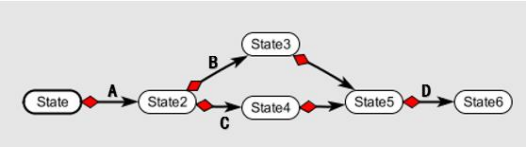


Fig. 8. BCS modeled in Ptolemy II's FSM.

The second point is the action of transition transitions. Once a transition is chosen in SyncBlock, its actions are executed in order. The format of an action is typically the operand= expressions. No matter the operand is a port name or variable name, all actions are specified by the action parameters of the transition together. But in Ptolemy II the output action is specified by the outputActions parameter of the transition for ports and the set action are specified by the setActions parameter of the transition for variables. So when we transform the transition's actions, the main idea is to separate port actions from variable actions.

The final point is about priorities. The priority is defined on the name of two transitions in a single automaton. Ptolemy II does not support priorities between transitions.

IV. Algorithm Design

algorithm 1: SyncBlock model atomic block to Ptolemy II's Modal Model
input: SyncBlock model atomic block B
output: Ptolemy II model P
for each variable vi∈B do
vi is transformed to Ptolemy II's variable
end
for each input/output port pi∈B do
pi is transformed to Ptolemy II's input/output port
end
for each parallel automaton ai∈B do
ai is transformed to Ptolemy II's Modal Model
end

algorithm 2: SyncBlock model compound block to Ptolemy II's TypedCompositeActor
input: SyncBlock model compound block C
output: Ptolemy II model P
for each input/output port pi∈C do
pi is transformed to Ptolemy II's input/output port
end
for each relation Ri between sub-blocks do
Ri is transformed to relation in Ptolemy II
end
for each sub-block Si∈C do
if Si is atomic block do
call the algorithm 1
end
if Si is compound block do
call the algorithm 2
end
end

V. Case Study

We conduct some experiments on a sub-module of train communication control system that is used in real world subway to show how to transform SyncBlock model to SR in Ptolemy II. The train control system is a safety-critical embedded system consisting of two controllers: multifunction vehicle bus(MVB) controller which interconnects devices within a vehicle, and wire train bus(WTB) controller which interconnects the vehicles of a train.

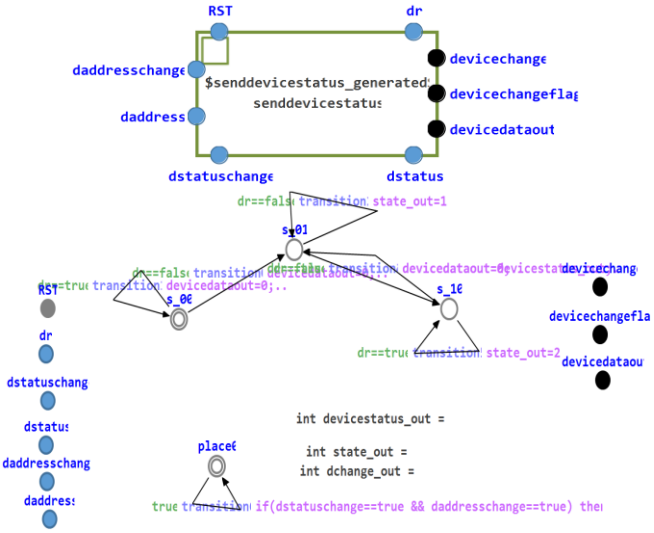


Fig. 9. An atomic block case FPGA submodule senddevicestatus.

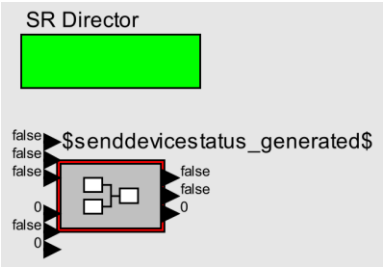


Fig. 10. Top level of the transformed model in Ptolemy II.

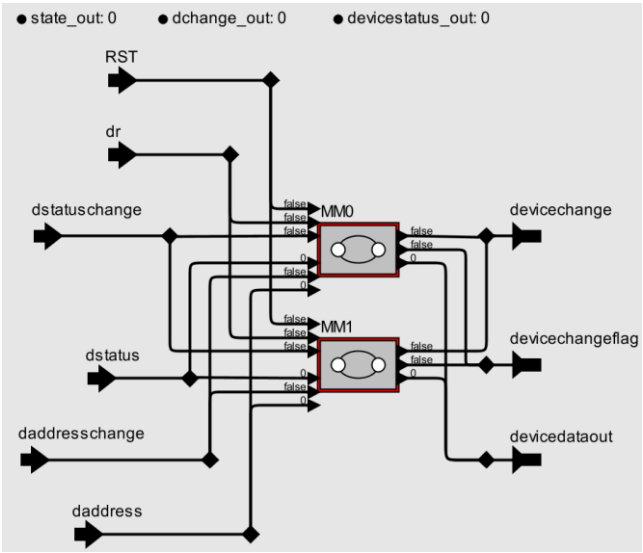


Fig. 11. submodule senddevicestatus transformed to Ptolemy II.

We focus on FPGA of the MVB controllers. As presented in the Fig 9, submodule senddevicestatus is modeled as an atomic block in SyncBlock. It has six input ports and three output ports, and two parallel automatons are inside it. in particular, the self-loop automaton has a transition with IF ELSE statement. So when transforming the model, the transition need refinement in Ptolemy II. The transformed model is presented in the Fig. 10.

We give SyncBlock 9 cycles' input value of every port with dr{true, false, false, true, true, false,f alse, true, true}, dstatuschange{true, true, true, true, true, true, true, true, true},

dstatus{4369,4369,4369,4369,4369,4369,4369,4369,4369}, daddresschange{true, true, true, true, false, false, false, false, false}, address{34952, 34952, 34952, 34952, 34952, 34952, 34952, 34952, 34952}, and we get the output value of every output port with devicechange{true, true, true, true, true, true, true, true, true},devicechangeflag{ false, true, true, true, false, false, false, false, false} devicedataout {0, 0, 0, 34952, 34952, 0, 0, 4369, 4369}

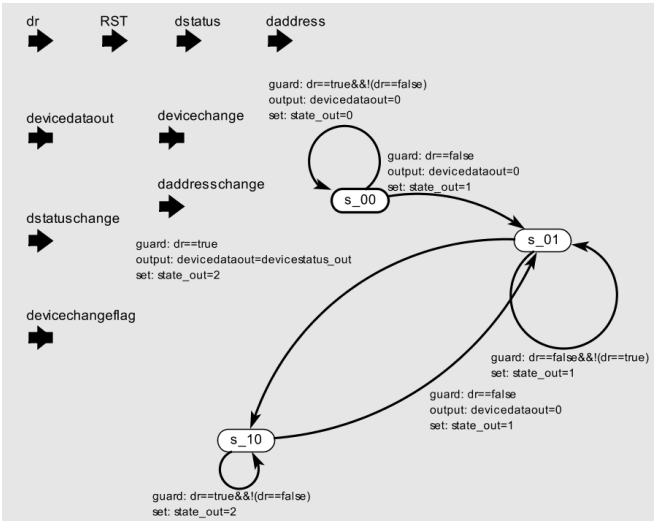


Fig.12. Automaton inside the ModalModel MM0.

Fig.13. Automaton inside the ModalModel MM1 at the left.

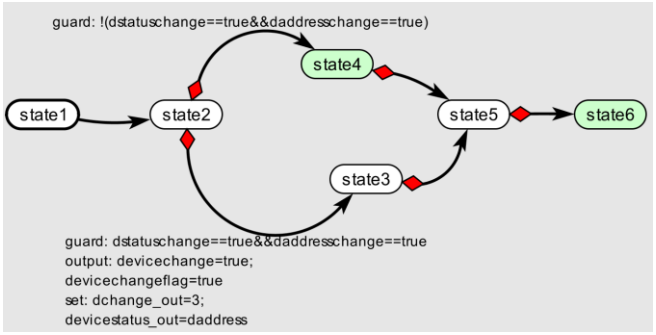


Fig. 14. Refinement inside the place0.

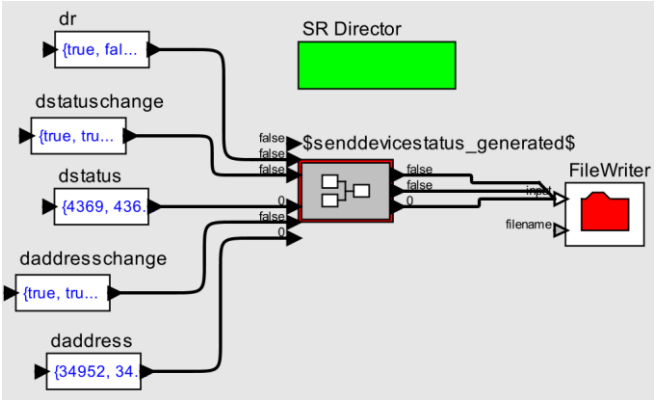


Fig.15. add input actor Sequence and output actor File Writer to transformed SR model.

As presented in the Fig. 15, we add the same input value and input cycle to the SR model, and we get the same output value as the original SyncBlock model. So we think our transformation is right.

VI. Conclusion

In this paper we present a strategy for heterogeneous model merging based on model transformation from SyncBlock to SR model of computation in Ptolemy II. Through transformation we can model synchronous embedded system by SyncBlock, simulate the pure synchronous model by SyncBlock, further more simulate the designed model coupling with modeling of external environment of embedded system by Ptolemy II, and finally generate code by using SyncBlock. This way combines the advantages of the two modeling tools for heterogeneous model merging.

In the future, we will prove the correctness of the conversion by bisimulaiton [15] verification.

References

[1] Y. Jiang. *SyncBlock: Novel Model for the Design of Synchronous Embedded System*.

[2] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, *Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems*, 1994.

[3] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity-the ptolemy approach," in *Proc. the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[4] C. Ptolemaeus, "System design, modeling, and simulation using ptolemy II," *Ptolemy*, pp. 235-265, 2014.

[5] Ptolemy. [Online]. Available: http://ptolemy.eecs.berkeley.edu/ptolemyII/ptII10.0/ptII10.0.1/ptolemy/domains/sr/demo/TrafficLight/TrafficLight/

[6] F. Boussinot and R. De Simone, "The esterel language," in *Proc. the IEEE*, vol. 79, no. 9, pp. 1293–1304, 1991.

[7] N. Halbwachs, F. Lagnier, and C. Ratel, "Programming and verifying real-time systems by means of the synchronous data-flow language lustre," *Software Engineering*, vol. 18, no. 9, pp. 785–793, 1992.

[8] D. Harel, "Statecharts: A visual formalism for complex systems," *IEEE Transactions on Software Engineering*, vol. 8, no. 3, pp. 231–274, 1987.

[9] G. Berry, "Circuit design and verication with esterel v7 and esterel studio," *IEEE International on High Level Design Validation and Test Workshop*, pp. 133–136, 2007.

[10] S. A. Edwards, *Cec: The Colombia Esterel Compiler*, 2003.

[11] M. Simulink and M. Natick, The mathworks. Inc., Natick, MA, 1993.

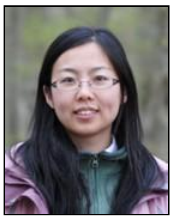
[12] G. Hamon and J. Rushby, "An operational semantics for stateflow," *Fundamental Approaches to Software Engineering*, pp. 229–243, Springer, 2004.

[13] E. A. Lee, "What's ahead for embedded software?" *IEEE Computer*, pp.18-26.

[14] E. A. Lee, "Finite state machines and modal models in Ptolemy II," *Electrical Engineering and Computer Sciences University of California at Berkeley*, 2009

[15] G. Barrett and S. Lafortune, "Bisimulation, the supervisory control problem and strong model matching for finite state machines," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 8, pp. 377–429 1998.

Hongtian Ma received the BS degree in software engineering from the Xinjiang University, Urumchi, China, in 2013. He is currently working toward the MS degree in software engineering from Tsinghua University, Beijing, China. His current research interests include domain specific modeling, formal verification and their applications in embedded systems.



Hehua Zhang received the BS and MS degrees in computer science from Jilin University, Changchun, China, in 2001 and 2004, respectively. She received the PhD degree in computer science from Tsinghua University, Beijing, China, in 2010. She is currently a lecturer in the School of Software at Tsinghua University. Her current research interests include domain specific modeling, formal verification and their applications in embedded systems.

Ming Gu received the BS degree in computer science from the National University of Defence Technology, Changsha, China, in 1984, and the MS degree in computer science from the Chinese Academy of Science at Shengyang in 1986. Since 1993, she has been working as a professor in Tsinghua University. Her research interests include formal methods, middleware technology, and distributed applications.