Model Driven Design of Heterogeneous Synchronous Embedded Systems

Huafeng Zhang[2], Yu Jiang[1], Han Liu[2], Hehua Zhang[2], Ming Gu[2], Jiaguang Sun[2]
Department of Computer Science, University of Illinois at Urbana-Champaign, Illinois, USA.[1]
School of Software, Tsinghua University,TNLIST, KLISS, Beijing, China[2]

ABSTRACT

Synchronous embedded systems are becoming more and more complicated and are usually implemented with integrated hardware/software solutions. This implementation manner brings new challenges to the traditional model-driven design environments such as SCADE and STATEMATE, that supports pure hardware or software design.

In this paper, we propose a co-design tool Tsmart-Edola to facilitate the system developers, and automatically generate the executable VHDL code and C code from the formal verified SyncBlock computation model. SyncBlock is a lightweight high-level system specification model with well defined syntax, simulation and formal semantics. Based on which, the graphical model editor, graphical simulator, verification translator, and code generator are implemented and seamlessly integrated into the Tsmart-Edola. For evaluation, we apply Tsmart-Edola to the design of a real-world train controller based on the international standard IEC 61375. Several critical ambiguousness or bugs in the standard are detected during formal verification of the constructed system model. Furthermore, the generated VHDL code and C code of Tsmart-Edola outperform that of the state-of-the-art tools in terms of synthesized gate array resource consumption and binary code size.

The abstract demo video address is :
https://youtu.be/D9ROyJmKZ4s
The tool, user manual and examples can be downloaded:
http://sts.thss.tsinghua.edu.cn/Tsmart-Edola/

CCS Concepts

•Software and its engineering → Model-driven software engineering;

Keywords

model driven development, computation model, hardware-software co-design

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

ASE'16, September 3–7, 2016, Singapore, Singapore
© 2016 ACM. 978-1-4503-3845-5/16/09...$15.00
http://dx.doi.org/10.1145/2970276.2970280

1. INTRODUCTION

With the increasing complexity of safety critical applications such as aerospace, transportation areas, etc., mixed hardware-software solutions are increasingly adopted. Such systems are commonly component-based, where some components are implemented in hardware to ensure determinacy and stability, while others without strict timing and performance constraints are usually implemented in software to save computation resource. This implementation pattern leads to an increasing heterogeneity of the final system, which challenges the traditional model driven design (MDD) environment (for example, SCADE [2] does not support partitioning and hardware synthesis, Simulink [6] does not support partition and temporal properties verification, and the semantics of their underlying computation models are complex and not easy for engineers to handle.).

Main Challenge: More specifically, the first challenge is that how to capture the heterogeneous behavior of both hardware and software modules in a unified co-design model, especially for timing consistency. The behavior of hardware module for synchronous applications is usually controlled by a hardware clock with a strict cycle, while the timing of software module usually depends on the size and complexity of the code. The second challenge is that how to ensure the correctness of the co-design model through a complete validation for the static and temporal properties. The last challenge is that how to overcome the gap between the complex co-design model and low-level mixed hardware-software parallel implementation on a dedicated hardware platform.

Proposed Toolkit: To address the above challenges, we implement Tsmart-Edola, to assist the co-design of complex safety-critical synchronous systems which embodies a mixed hardware-software solution. As presented in Figure 1, Tsmart-Edola is built on the formal computation model SyncBlock [9]. (1) First, a graphical model editor based on the syntax of SyncBlock is provided to support high-level modeling of hierarchical system decomposition, and concurrent synchronous behavior of both hardware and software components. (2) Second, a graphical simulator based on the executive semantics of SyncBlock is provided to support the interactive graphical simulations of the model under development. (3) Third, a verifier translator is provided to translate the selected graphical model to a synchronized labeled transition system that can be formally verified directly by tool Beagle [8], to get the provably exhaustive. (4) Fourth, a code generator is provided to generate VHDL code for the components partitioned into hardware implementation, and C code for software implementation. Because the timing in-

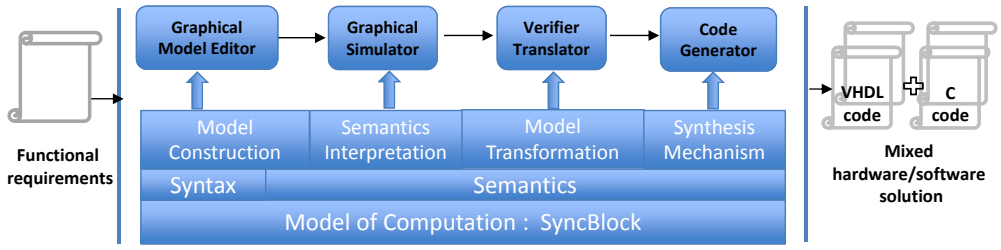

Figure 1: Tsmart-Edola design toolkit builded on SyncBlock computation model. It consists of four components: graphical model editor, graphical simulator, verifier translator, and code generator.

terval for the behavior of software modules usually depends on the complexity of statements and the speed of processor, the code generator needs to generate a scheduling mechanism to keep the timing consistency and synchronization among different software and hardware modules, which are executed in parallel and independent with each other.

With the support of Tsmart-Edola, we are able to shorten the development cycle to build and validate a SyncBlock model at high-level, compared to the traditional practice which involves implementing the sketch-like design in low-level programming languages as C and VHDL. In addition, the graphical model validation through simulation and enhanced formal verification opens a user-friendly interface for us to uncover design defects at the early development stage. Once we have verified safety-critical properties over the system model and iteratively improved the design details, executable code can be directly generated in C(software) and VHDL(hardware). The synthesized implementation is compact and has a smaller size than the results of some other VHDL and C code generators. For example, when we apply it to a multifunction vehicle bus control system design, given a function that can be modeled by both SyncBlock and Stateflow with almost the same number of states and transitions, the size of generated VHDL code and C code is reduced by up to 40% and 50% respectively, compared to Simulink code generators of Stateflow. Gate array consumption of VHDL synthesis and binary code of C compilation are also smaller by 35% and 49% respectively. Furthermore, several safety-critical bugs in the standard IEC 61375 [4] are detected during the model validation of Tsmart-Edola.

Contribution: Main contributions are summarized as:

- We have implemented Tsmart-Edola to support the lightweight formal computation model SyncBlock, to tackle the complexity of the heterogeneous embedded system design, where graphical model construction, simulation, verification, and effective partitioning and modularized resource saving code generation are implemented.
- We have applied it to a real system design, where developers successfully found critical bugs of the IEC standard during the model validation, and deployed the synthesized system in operation of real subway control. We have made it publicity free, which may motivate scientific community to use it to work on more real model-driven design practice.

Paper Organization: Related work is presented in Section 2. Some backgrounds on the formal computation model SyncBlock is presented in Section 3. Section 4 briefly introduces the tool implementation and integration of Tsmart-Edola. Experiment results on the model-driven design of a real train control system adopting the mixed software-hardware solution are presented in Section 5, and we conclude in Section 6.

2. RELATED WORK

Model-Driven Embedded System Design: There are large amounts of work and various toolkits in the industry and academia supporting the design of general synchronous reactive embedded software systems. For example, STATEMATE supports Statechart, which is good for control logic description but the support of the dataflow and structure description is limited [7]. SCADE uses safety state machine(SSM) as the formal basis, and has been successfully applied in a variety of applications. While mainly focusing on embedded software, SCADE currently has little support for the synthesis of hardware. Simulink is now widely used with Stateflow as its basis [6]. It presents strong modeling, simulation and synthesis capability, but has no formal semantics for comprehensive verification on safety-critical applications even with support of Design Verifier [12].

Except for the three famous industrial frameworks above, there are some tools for academia interests. For example, ForSyDe is implemented on a Haskell based domain specific language, served as high-level modeling and refinement of digital systems [14]. POLIS supports CFSM, and uses Esterel as the specification language for simulation and synthesis [1]. Although it supports the co-design of hardware-software, the model must be translated and use another separate tool Ptolemy for simulation [3]. Others such as GalsBlock and DDFchart focus on the applications with both synchronous and asynchronous modules implemented in pure hardware [11, 17, 15, 16].

Main Difference of Tsmart-Edola: (1) Most of the above tools can generate either C or VHDL code for the whole model, but few allows to configure some subsystems to be implemented in C and others in VHDL, such as SCADE and STATEMATE. Modular code generation is highly demanded by engineers. To guarantee the correctness of such configurable co-synthesis, synchronization among the parallel software and hardware modules is supposed to be fed with special scheduling mechanism, which is seldom implemented. (2) Tools such as Polis and Simulink address the modeling and co-synthesis problem, but the verification support needs to be increased with the increasing complexity and number of safety requirements of the system. Our lightweight semantics and modeling elements results in a easier understanding

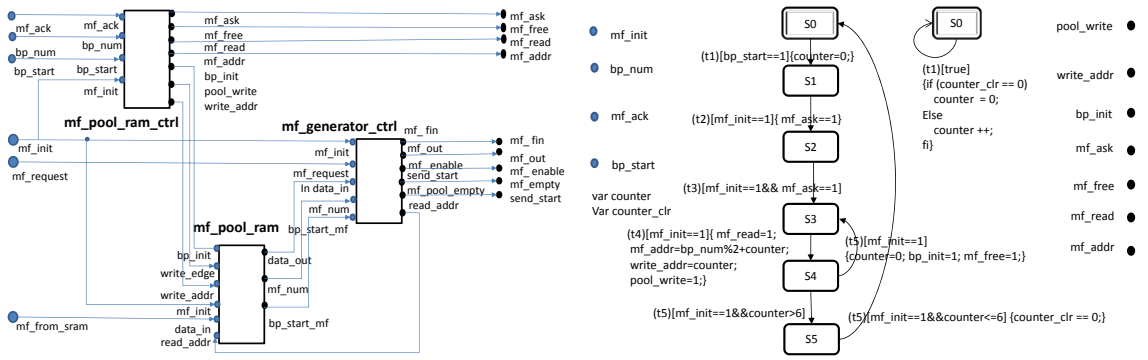

Figure 2: A visual example of SyncBlock parallel computation model, including the compound block, atom block, input data port, output data port, dataflow connection, parallel automata, local variable, and transition.

and increases the verification and synthesis ability, which makes the life of engineers easier. *In summary, Tsmart-Edola is not a subset of SCADE and Simulink, it increases the verification and synthesis ability of them with the cost of reducing the complex seldomly used features and semantics of the underlying computation model. Notice that even if we just use a subset of elements in SSM and Stateflow for modeling, the synthesis and verification ability of the constructed model in the SCADE and Simulink will not be increased.*

3. BACKGROUND ON SYNCBLOCK

In SyncBlock computation model, a system is specified as a combination of compound and atom blocks communicating through point-to-point channels. The compound block does not directly conduct computation. Instead, it presents the hierarchical decomposition of general component structure and data flow path among them. The atom block is refined as parallel automata to capture the component behaviors and detail computations.

We illustrate main features of SyncBlock with a graphical example in Tsmart-Edola as presented in Fig. 2. This module accomplishes the function of master frame generation and sending logic. At the outermost level, compound block *mf_generator* is refined as three sub-blocks (atom block *mf_pool_ram*, *mf_generator_ctrl* and *mf_pool_ram_ctrl*). Theses atom blocks cooperate to generate the master frame to be sent in the train communication network. The dots attached on the right side of each block are used to denote the output data port, while the dots on the left side are used to denote the input data port. Communications among these blocks are realized through the data port connections. The arrow on the connection indicates the signal flow direction of data exchange, and the expression of the connection facilitates the data-oriented behavior modeling. The input data port of the compound block can be connected to the input data port of the refined atom blocks (e.g. *bp_start* $\rightarrow$ *bp_start*). The output data port can be connected to the input data port of other blocks (e.g. *pool_write* $\rightarrow$ *write_edge*), and the output data port of high-level blocks (e.g. *send_start* $\rightarrow$ *send_start*).

Atom block *mf_pool_ram_ctrl* is refined by two parallel automata with two local shared variables *counter* and *counter_clr*. The transition between two states consists of three parts: name, guard, and action. All statements are defined on the updates of data ports and local variables, and all statements of an action a_i attached on a transition are executed in parallel. Automata contained in an atom block ab_i is also executed in parallel and independent of each other, but there still might be conflicts on read-write operations (e.g. *counter*). Each local variable and output data port can be written by one automaton, and the read operation is prior to the write operation. All atom blocks contained in the model are also executed in parallel. The complete rules and definitions can be referred to the manual [9].

4. TOOL IMPLEMENTATION

Tsmart-Edola is implemented on Eclipse Rich Client Platform with Eclipse Graphical Editing Framework for 57,265 lines of Java code, where 36,278 lines are inherited from Tsmart-GalsBlock [17] used for pure hardware design. The visual interfaces of most model-driven design environment are similar, but their kernel implementations for simulation, verification, and code generation are different, because their underlying computation models are totally different. Overall structure of Tsmart-Edola is presented in Figure 1.

Graphical Model Editor. Graphical model editor is based on the model construction rules presented in [9], and supports graphical modeling of hierarchical system decomposition, point-to-point channel communication, and concurrent synchronous behavior. It facilitates to capture system requirements and functional descriptions visually via the `SyncBlock` model of Figure 2, and the main interface is introduced in Figure 3.

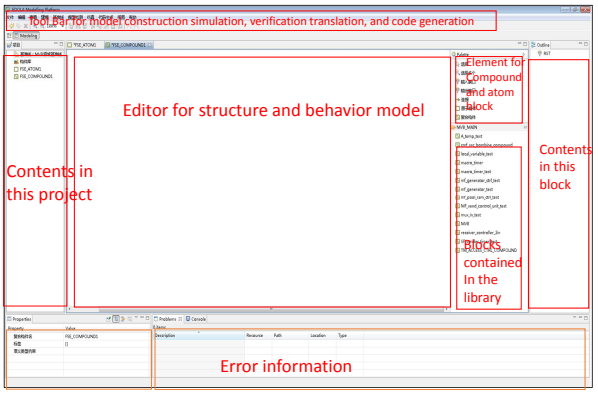

Figure 3: The overall interface of the implemented Tsmart-Edola toolkit.

The interface shows how to construct the hierarchical structure and data connections among system components. It contains six views. (0) The toolbar allows us to create, simulate, partition, verify, and synthesize code from the selected blocks in the editor view. (1) The package explorer shows the projects in the workbench and the models contained in the current project. (2) The editor view shows the diagram of a selected model and allows us to edit it by adding/removing elements. (3) The palette provides the elements (data ports, connection, idle atom block, idle compound block) that can be dragged and dropped in the model shown by the editor. (4) The palette provides a reusable and common system component models library that can be dragged into the editor. (5) The element explorer shows the contents contained in the block of the selected atom or compound block in the editor view. (6) The properties view allows us to view and edit the properties of the element selected in the editor view and some error information presentation.

A similar interface to define component behaviors is also designed. When the engineer selects an atom block in the editor view of the structure interface described above, clicks it twice, then the interface will be opened. It provides similar views of Figure 3 except two differences. Instead of the elements (data ports, connection, idle atom block, idle compound block) for structure construction, the palette view provides the elements (data ports, variables, transition, states) that can be dragged to construct parallel automata in the editor view. The properties view allows us to view and edit the properties of an element, especially for the complex actions and priorities attached on each transition.

Graphical Simulator. Graphical simulator is based on the model interpretation algorithm of SyncBlock presented in [9], and supports the interactive graphical simulation of the system model. In the semantics of SyncBlock, it should adhere to the reactions of the real system, where each system reaction step mainly involves in three phases: import inputs, compute changes and export outputs. During simulation, users can explore the system behaviors to check the correctness of the constructed model. The simulation function provides some basic functions: step-forward, step-roll back, read inputs from the input files for multi-step forward, etc. Through the interactive simulation, most functional requirements can be checked.

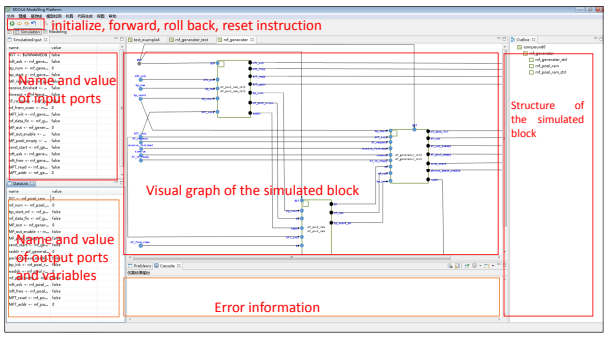

Figure 4: The overall interface of the simulation.

The simulation interface provides six views. (0) The toolbar allows us to do initialization, reset, execution forward, and roll back. (1) The input view shows the name of the input data ports, where the engineer can input the values for each computation. (2) The output view shows the name of the output data ports and the shared variables, where the engineer can check the values after each computation. (3) The editor view shows the content of the simulated block. If we click an atom block twice, each automaton contained in this block would be presented. The transitions in a computation are executed visually, and the current active states are highlighted in red. (4) The element explorer shows the contents and the structure of the simulated blocks in the editor view. (5) The properties view allows us to view some error information during simulation.

Verifier Translator. Verifier translator is based on the formal semantics of SyncBlock, and translates selected graphical model to a labeled transition system. Detail translation rules such as each parallel automata flatten, complex actions split, are described in the manual [9]. The generated transition system is described within the generated files with suffix *.elts, and can be directly invoked by tool Beagle [8] for formal verification. We just need to click the button "Export to ETLS" in the toolbar, the SyncBlock model would be automatically translated into the labeled transition system file, and efficiently verified by the tool Beagle. The formal verification makes up the incompletion of simulation to strengthen the correctness for safety critical requirements.

Code Generator. Code generator is based on the co-synthesis mechanisms presented in [9], and generates the hardware description files with suffix *.vhd for the blocks partitioned into hardware implementation, and executable software code files with suffix *.c for the blocks partitioned into software implementation. Noting that many partitioning algorithms have been proposed in the last decades, which is not the main concern and contribution of this work, and we use our previous algorithms [10] for partitioning. The key idea is to custom each atom block of SyncBlock as a node in the partitioning algorithm and find a bipartition P on a communication graph denoted as $G(V, E)$, where V is a set of nodes $\{v_1, v_2 \cdots v_n\}$ and E is a set of edges $\{e_{ij} | 1 \leq i, j \leq n\}$, and P $= (V_h, V_s)$ such that $V_h \bigcup V_s = V$ and $V_h \bigcap V_s = \emptyset$. Then, the partitioning problem can be decided by a decision vector $\mathbf{x}(x_1, x_2 \cdots x_n)$, representing implementation way of n task modules. When the value of x_i is 0 (1), the task module will be implemented in hardware (software). Objective of the problem is changed to search an n-dimensional space to find the optimal value of the decision vector on the objective function of hardware cost $H(x)$ and time constraints $T(x)$.

Then, for software synthesis, we use C sub-function call to capture the compound block, and C thread definition to capture the automata contained in the atom block. Because the automata contained in the innermost atom block are running in parallel and synchronized with the clock, the thread for each automaton needs to be synchronized with the barrier. We implement a dynamic barrier in two files named timer.h and timer.c as the synchronization scheduler of all threads. For hardware synthesis, we use the architecture description code of component map in VHDL to capture the compound block, and the behavior description code of process definition in VHDL to capture the automata contained in the atom block. Furthermore, to keep the execution cycle consistency between the reaction of hardware FPGA processor and the reaction of software ARM processor, the frequency of the FPGA processor is used to initiate the parameter of the sleep() function contained in the generated software. We just need to click the button "Export Model as C" and "Export Model as VHDL" in the toolbar, the SyncBlock model

would be automatically translated into the corresponding executable C and VHDL code. The generated code outperforms the state-of-the-art VHDL and C code generated by other tools in terms of synthesized gate array resource consumption and binary code size.

5. EXPERIMENT RESULTS

To evaluate the effectiveness of Tsmart-Edola, we apply it to the model-driven design of a multifunction vehicle bus controller(MVBC) used in the train communication network (TCN), which is standardized in IEC 61375 [18, 19]. We compare it with BeagleBone [13] (one available design framework for MVBC of class 2) and Simulink (one framework for general system design). We mainly compared them in the bug detection of the IEC standard 61375 and the resource consumption during the model-driven design of MVBC. Overall results are presented in Tabel 1 and introduced as follow.

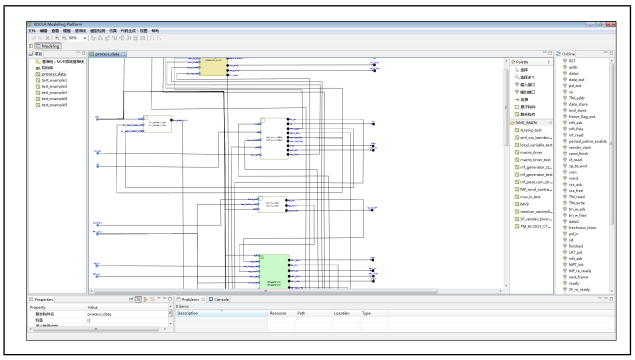

Figure 5: The model in Tsmart-Edola for the MVB controller of level 5.

Model Construction: Strictly following the specification of the standard IEC-61375, two types of communication services of MVBC are modeled as eighteen compound blocks presented in Figure. 5. Modules can be hierarchically constructed and debugged in Tsmart-Edola. The constructed SyncBlock model for the whole system is presented in Figure 5, with details can be referred to the website in the abstract. Inheriting those modeling of MVBC within class 1-4 of previous works presented in section 2, we build the model for process data communication which is later partitioned for hardware implementation. Then, according to the description of IEC-61375, we build the model for message communication of class 5 which is partitioned for software implementation. The MVBC of class 5 supports both process data and message data communication with mixed hardware-software implementation, as adopted by the most widely used MVBC D113 of Duagon company [5].

Model Validation: After preliminary graphical model simulation, the constructed model is translated through verifier translator, and safety-critical properties are formulated as logic assertions for Beagle verification. For example, you can formalize the requirement that during data packet retransmission procedure, the resent data packet number should be the next packet to be sent as the formula as:

$$[\,]((NK==true)\ \ derive\ \ (NK\_number==next\_send))$$

Unfortunately, the assertion fails to pass the verification. Through manually analysis of the counterexample provided in Beagle, we located the violation in the atom block $frame\_retransmission$ contained in the compound block $message\_service$. The bug is tracked to C statement $\{expected := NK\_seq\_nr;\ send\_not\_yet := (expected + credit)\ \%\ 8;\}$ contained in the action of transition for the data retransmission, which is located in Table 33 of the standard IEC 61375. In this buggy scenario, the system fails to update the value of packet number to be retransmitted. To fix the bug, the statement needs to be changed to $(\{ expected := NK\_seq\_nr;\ send\_not\_yet := (expected + credit)\ \%\ 8;\ next\_send := expected;\})$. As presented in Table 1, six problems about the message transmission and master rotation are verified and detected, which can be traced back to the pseudo code descriptions in Table 33-35 and Figure 105 of IEC 61375. By fixing the bugs in the standard, the modified SyncBlock model passes the verification.

Code Generation: Following the implementation style of Duagon company, the process data communication related modules partitioned for hardware implementation are synthesized with 11,000 lines of VHDL code, and the message communication related modules partitioned for software implementation are synthesized with 65,000 lines of C code.

The VHDL code for the atom block $mf\_generator\_ctrl$ of Figure 2 contains 359 lines of code with 7KB in size. If we use Stateflow to model this function and generate code by Simulink, the size grows to 12 KB, although number of the state and transition is the same for Stateflow model and SyncBlock model. Furthermore, because SyncBlock computation model uses the bounded integer, type of the interface declaration in generated VHDL code is mapped to the ranged integer. This contributes to a great reduction on resource usage when generating the RTL (Register Transfer Level) file from VHDL code. When we synthesized all VHDL code into the target device xc6slx16-3ftg256 with the xilinx ISE, it will cost 3380 number of slice registers, while for the VHDL code generated by Simulink for Stateflow model, the number is increased to 5421. The generated C code for atom block $master\_transfer$ is 893 lines with 53KB in size. But in Simulink, generated C code is almost double-sized to be 98KB with 2461 lines of code for the main logic. The generated C code can also be compiled and simulated in the visual studio development environment of Microsoft, and the binary file is 349 kb and 683 kb for the code generated by SyncBlock and Simulink respectively. The difference is mainly derived from the fact that Simulink generates many extra configuration files and introduces many libraries for scalability. Similar results hold for the comparison between Tsmart-Edolaand BeagleBone. Details about the resource utilization of VHDL synthesization and C compilation, and the bug detection results are concluded in Table 1, which proves the efficiency of the toolkit Tsmart-Edola.

Synthesized Device: Then, the synthesized binary files for the generated VHDL code and C code can be loaded into the FPGA and ARM processors on the MVBC network card, respectively. To test the reliability of the co-synthesized system, we connect the widely-used industrial product MVBC card D113 with our synthesized controller card for real-time communication. The co-synthesized system is embedded into an industrial computer to receive instructions from the keyboard. We use an application running on the industrial computer to monitor the communication, and read the message data from the memory. Sequences and contents of data frames are accurately compatible with the definition in the standard IEC-61375. In addition, the automatically co-

Table 1: Resource utilization of VHDL synthesization and C compilation for MVBC system of class 5 and 2.

VHDL Logic Utilization	Tsmart-class-5	Simulink-class-5	Tsmart-class-2	BeagleBone-class-2
Number of Slice Registers	3380	5421	2577	3728
Number of Slice LUTs	4724	6957	3915	5374
Number of fully used LUT-FF pairs	2927	4295	2009	2712
C Compilation	Tsmart-class-5	Simulink-class-5	Tsmart-class-2	BeagleBone-class-2
Binary File Size KB	349	683	0	0
Bug in IEC Standard Detected	6(verification)	1(simulation)	2(verification)	1(verification)

synthesized MVBC passes the physical and electrical test. It has been also equipped on a real test train traveling over thirty thousand kilometers without failures.

6. CONCLUSION

In this paper, we develop the toolkit Tsmart-Edola to facilitate the design of complex synchronous embedded systems with both hardware and software components. The graphical model construction, and validation through simulation and verification help us find problems in the early stage of system design. After all properties are satisfied, we can generate executable implementation from the validated model automatically, with some components in C, and others in VHDL. Furthermore, when we apply it to the model-driven design of MVBC based on the standard IEC 61375, several critical bugs and some other ambiguousness in the standard are detected during the model verification, and automatic implementation based on the bug-free model has been deployed and in operation in real subway controllers. In the future, code generators to Java and Verilog will also be added, code verifiers will also be integrated for the verification of the generated code and the toolkit itself, and different versions for different operation systems will be provided.

7. ACKNOWLEDGMENT

This research is sponsored in part by NSFC Program (No. 91218302, No. 61527812), National Science and Technology Major Project (No. 2016ZX01038101), Tsinghua University Initiative Scientific Research Program (20131089331), MIIT IT funds (Research and application of TCN key technologies) of China, and The National Key Technology R&D Program (No. 2015BAG14B01-02).

8. REFERENCES

[1] F. Balarin. *Hardware-software co-design of embedded systems : the POLIS approach*. The Kluwer international series in engineering and computer science. Kluwer Academic Publishers, 1997.

[2] Berry. Scade-synchoronous design and validation of embedded control software. In *Proceedings of the workshop Next generation design and verification methodologies for distributed embedded control systems*, pages 19–33. Springer, 2007.

[3] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. 1994.

[4] I. E. Commission et al. Iec 61375-1. *Train Communication Network*, 2011.

[5] Duagon. Mvb controller: D113. 2014.

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

[7] D. Harel and M. Politi. *Modeling reactive systems with statecharts: the STATEMATE approach*. McGraw-Hill, Inc., 1998.

[8] F. He, L. Yin, and B.-Y. Wang. *Beagle http:// sts.thss.tsinghua.edu.cn/ ceagle/*.

[9] Y. Jiang, M. Gu, and J. Sun. User manual of syncblock. In *Technical Report*, pages 1–56. Tsinghua University, 2015.

[10] Y. Jiang, H. Zhang, X. Song, W. N. Hung, M. Gu, and J. Sun. Uncertain model and algorithm for hardware/software partitioning. In *IEEE Computer Society Annual Symposium on VLSI, 2012*, pages 243–248. IEEE, 2012.

[11] H. Zhang, X. Song, W. N. Hung, M. Gu, and J. Sun. Design of mixed synchronous/asynchronous systems with multiple clocks. In *IEEE transaction on parallel and distributed systems*. IEEE, 2014.

[12] T. MathWorks. Simulink. *Inc., Natick, MA*, 2010.

[13] R.Aarthipriya and S. Chitrapreyanka. Fpga implementation of multifunction vehicle bus controller with class 2 interface and verification using beaglebone black. *International Journal of Science and Engineering Research*, 3(5):3221–3225, 2015.

[14] I. Sander and A. Jantsch. System modeling and transformational design refinement in forsyde [formal system design]. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(1):17–32, 2004.

[15] R. Wang and M. Gu. Formal modeling and synthesis of programmable logic controllers. *Computers in Industry*, 62(1):23–31, 2011.

[16] W. Wei and X. Fan. Imperfect information dynamic stackelberg game based resource allocation using hidden markov for cloud computing. 2016.

[17] H. Zhang, Y. Jiang, X. Song, W. N. Hung, M. Gu, and J. Sun. Tsmart-galsblock: A toolkit for modeling, validation, and synthesis of multi-clocked embedded systems. In *Proceedings of the 2014 Foundations of Software Engineering*. ACM, 2014.

[18] Y. Yang. From stateflow simulation to verified implementation: A verification approach and a real-time train controller design. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2016.

[19] Y. Jiang and H. Zhang. Design and optimization of multi-clocked embedded systems using formal techniques. *IEEE Transactions on Industrial Electronics*, 62(2):1270–1278, 2015.