Tsmart-GalsBlock: A Toolkit for Modeling, Validation, and Synthesis of Multi-clocked Embedded Systems

Yu Jiang[1,2], Hehua Zhang[1], Huafeng Zhang[2], Xinyan Zhao[1], Han Liu[1], Chengnian Sun[3], Xiaoyu Song[2], Ming Gu[1], Jiaguang Sun[1]
School of Software, Tsinghua University,TNLIST, KLISS, Beijing, China[1]
Department of Computer Science and Technology, Tsinghua University, TNLIST, KLISS, Beijing, China[2]
Department of Computer Science, University of California, Davis, USA[3]

ABSTRACT

The key challenges of the model-driven approach to designing multi-clocked embedded systems are three-fold: (1) how to model local synchronous components and asynchronous communication between components in a single framework, (2) how to ensure the correctness of the model, and (3) how to maintain the consistency between the model and the implementation of the system.

In this paper, we present Tsmart, a self-contained toolkit to address these three challenges. Tsmart seamlessly integrates (1) a graphical editor to facilitate the modeling of the complex behaviors and structures in an embedded system, (2) a simulator for interactive graphical simulation to understand and debug the system model, (3) a verification engine to verify the correctness of the system design, and (4) a synthesis engine to automatically generate efficient executable VHDL code from the model. The toolkit has been successfully applied to designing the main control system of a train communication controller, and the system has already been deployed and in operation. The evaluation of Tsmart on this real industrial application demonstrates the effectiveness and the potential of the toolkit.

The video demo and tool are available at the website:
https://sites.google.com/site/jiangyu198964/home

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

computer-aided software engineering

Keywords

multi-clocked embedded system, program synthesis

FSE'14, November 16–21, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3056-5/14/11...$15.00
http://dx.doi.org/10.1145/2635868.2661664

1. INTRODUCTION

Over the past decades, numerous programming models, tools and frameworks have been proposed to design and optimize embedded systems, in order to shorten the time to market and ensure the correctness of applications. Traditionally, these tools are based on abstract and rigorously defined mathematical models with a single global clock. Such an assumption of global synchronization significantly reduces the complexity of system design. The class of tools supporting global synchronous models mainly includes Esterel studio based on the Esterel model [4], SCADE suite based on the Lustre model [3], Polychrony toolset based on the Signal model [8]. These formal design tools based on well-defined mathematical models yield a rigorous methodological support for the trusted design, automatic validation, and systematic synthesis of global synchronised embedded systems.

Although those tools are effective for single-clocked system design, they provide little support for the design of systems with multiple clocks. The multi-clocked embedded systems, increasingly used in today's complex applications, usually involve concurrent behaviors with different local control clocks. This feature leads to several challenges beyond the abilities of the existing tools. The first is the modeling capability, that is, how to capture the behavior of the local synchronous component belonging to different clock domains, the asynchronous communications among components, as well as the structure of the system in a single graphical model. The second is the analytical capability, *i.e.*, how to ensure the correctness of the constructed multi-clocked model, especially to check whether the model satisfies the functional requirements or not. The last is the synthesis capability to maintain the consistency and overcome the gap between the constructed multi-clocked model and the executable implementation in the tool.

In this paper, we present a novel toolkit, named Tsmart[1], to support the model driven approach to the design of multi-clocked embedded system. The toolkit consists of four parts. **(1) Graphical Model Editor:** The editor is based on a novel computation model GalsBlock [11, 12] and supports graphical modeling of hierarchical system decomposition, synchronous concurrent process execution controlled by different clocks, and data-oriented asynchronous signal communication. It facilitates building GalsBlock models to meet the system requirement and functional descriptions. **(2) Simulator:** The simulator implements the operational semantics of GalsBlock and provides interactive graphical

[1]This demo illustrates the implementation of the techniques presented at FSE'13 [11] and TPDS'14 [12]

simulation of the system model under development. Users can explore the properties and behaviors of the system during simulation to check its correctness or debug the bugs in the model. **(3) Verifier:** The verification engine translates the graphical model to a labelled transition system (LTS), and directly invokes the formal verification tool Beagle [9] to validate the correctness of the model. Once a violation is detected, the simulator can serve as a debugger to aid in understanding the violation. **(4) Code Generator.** This component automatically generates efficient VHDL code from the graphical model, which can be directly loaded into FPGA processor for execution. These four components are seamlessly integrated together to form a complete tool chain to support the whole life cycle of the model-based approach for embedded system design.

The Tsmart toolkit facilitates the design and implementation of complex multi-clocked embedded systems. It is much easier to use Tsmart to build and validate a GalsBlock model at a high level than implement the system from scratch in low-level programming languages such as C and VHDL. The latter way usually needs more effort and is more error-prone. The graphical model validation through simulation and verification helps us find problems in the early stage of system design. After all properties are satisfied, we can generate the executable implementation from the validated model automatically. The synthesized implementation is compact and usually has a smaller size than other VHDL code generators. For example, the size of the generated code is reduced by up to 40% compared to Simulink VHDL code generator of Stateflow for a function that can be modeled by both Galsblock and Stateflow with the same number of state.

We have applied Tsmart to the design of a multifunction vehicle bus (MVB) control system used in the train communication network (TCN), according to the specification in the standard IEC 61375 [6]. Two critical bugs in the standard are detected during the model validation process, and the automatically generated implementation has been deployed and in operation in real subway control.

2. RELATED WORK AND MOTIVATION

Besides the tools introduced in Section 1 for single-clocked global synchronous system design, a large body of work has been dedicated to the design of multi-clocked systems. Esterel studio supports the design of multi-clocked systems based on MC-esterel v7 [4]. The major problem is that a designer has to work at a relatively low level and therefore the productivity is limited. Besides, the code generation capacity is limited as many basic modeling constructs are not supported, and the simulation based on a single basic clock is not intuitive or effective. Its variants such as the tool based on CRSM support graphical modeling, but the translation-based simulation and verification are not effective [13]. Similar to MC-esterel, all clocks are also defined on a single clock. Ptolemy supports modeling, and simulation of mixed synchronous and asynchronous systems [5]. However, it is primarily used as a simulation environment but not a verifier or program synthesizer. The Polis design environment based on CFSM model uses discrete events to model the interaction between software and hardware components, but does not support multi-clocked interactions [2]. Some translation based frameworks are also proposed to solve the analysis of multi-clocked systems. For example, in [7], F. Doucet *et al.* use a mixture of synchronous descriptions in Signal [1] and asynchronous descriptions in Promela [10] and provide a translation from Signal modules to Promela processes for simulation and verification. But all Sinal modules share a single clock. These toolkits and frameworks do not contain a complete tool chain and provide limited support to the model based design of multi-clocked embedded systems, motivating us to develop the Tsmart toolkit.

3. BACKGROUND ON GALSBLOCK

Fig 1 illustrates an example of GalsBlock computation model. At the top level, the compound block **Compound Top** consists of two sub-blocks (a compound block **Compound1** and an atom block **Atom1**). The clock attached to a compound block does not play a part, and just provides a virtual interface for the control clocks of its inner atom blocks. For example, the frequencies of real clock **CLK3** and **CLK4** are derived from the virtual clock **CLK1**, where the derived rules such as double and triple frequency can be configured according to different requirements. The bullet attached on the right side of each block denotes the output data port, while the bullet on the left side represents the input data port. The input data ports of **Compound Top** can be connected to the input ports of the two sub-blocks (*e.g.*, b $\rightarrow$ g, a $\rightarrow$ c), and the output ports of the sub-blocks can be connected to the both input and output data ports of other blocks (*e.g.*, e $\rightarrow$ f, h $\rightarrow$ i). The expression on the connection from port **b** to port **g** facilitates the data-oriented behavior modeling.

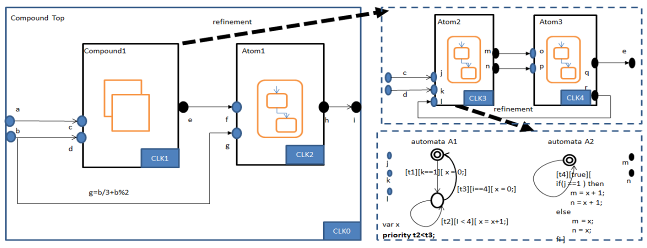

Figure 1: The example of GalsBlock computation model, including compound and atom blocks.

With the hierarchical structure and dataflow presented in the compound block, system behaviors are described by parallel automata in the atom block. For example, the atom block **Atom2** is refined as two automata controlled by clock **CLK3**, with a local shared variable and a transition priority expression. The operation of each local block is controlled and triggered by the local clock. More details about the element definition and semantic interpretation of GalsBlock are available in [11, 12].

4. OVERVIEW OF THE TOOLKIT

The toolkit provides a complete tool chain to support the design of multi-clocked embedded systems. It is implemented by 41,056 lines of Java code, and is downloadable at [14], as well as the introductive video.

4.1 Graphical Model Editor

In this graphical editor, an engineer can edit a GalsBlock model. It is implemented on Eclipse Rich Client Platform with Eclipse Graphical Editing Framework. The editor contains two interfaces, one for the construction of the system structure, and the other for defining component behavior.

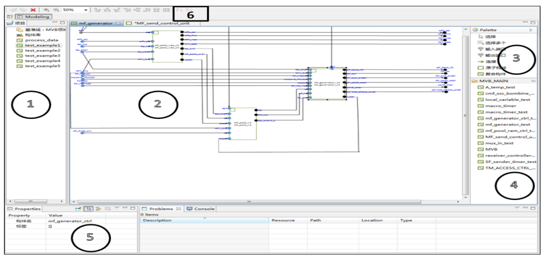

Figure 2: Structure interface for a system.

Fig 2 shows the interface to construct the hierarchical structure and data connections between system components. It contains six views. The package explorer (1) shows the projects in the workbench and the models contained in the current project. The editor view (2) shows the diagram of a selected model and allows us to edit it by adding/removing elements. The palette (3) 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. The palette (4) provides reusable and common system component models that can be dragged into the editor. The properties view (5) allows us to view and edit the properties of the element selected in the editor view, especially for the frequency of the clock attached on the idle atom block. The tool bar (6) allows us to simulate, verify, and synthesize code from the selected blocks in the editor view.

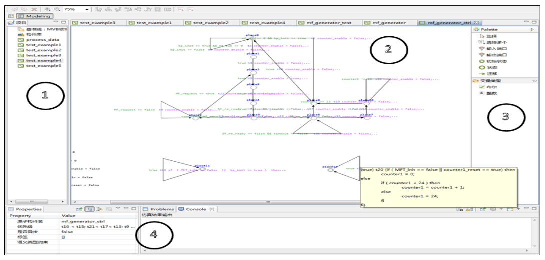

Figure 3: Behavior interface for a component.

Fig 3 shows the interface to define component behaviors. When the engineer selects an atom block in the editor view of the structure interface, double clicks it, then this interface will be opened. It provides four views, which are similar to the views of structure interface except two differences. The palette view (3) provides the elements (data ports, variables, transition, states) that can be dragged to construct parallel automata in editor. The properties view (4) allows us to view and edit the properties of an element, especially for the complex actions and priorities attached on each transition.

4.2 Graphical Model Simulator

Fig 4 shows the simulator, which interprets the constructed model based on the operational semantics defined in [11, 12]. It provides four views. The input view (1) shows the name of the input data ports, where the engineer can input the values for each computation. The output view (2) shows the name of the output data ports and the shared variables, where the engineer can check the values after each computation. The editor view (3) shows the state of each automaton contained in the selected blocks. The transitions in a computation are executed visually, and the current active states are highlighted in red. The tool bar (4) allows us to do initialization, reset, execution forward, and roll back. Through the interactive simulation, most of the functional requirements can be checked.

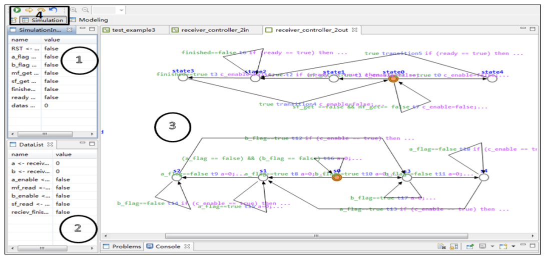

Figure 4: Editor view for component behavior.

4.3 Verifier

For safety-critical functional requirements, the incompleteness of simulation can be overcome through formal verification. Based on the formal semantics of GalsBlock model defined in [12], we translate the GalsBlock model into an equivalent LTS and invoke the formal verification tool Beagle [9] to verify its correctness. The details of translating clocks, complex actions split, data port communication from a hierarchical GalsBlock model to a flat LTS are described in [12]. Engineers can choose the block to be verified in the editor view of Fig 2, and press the button with the label *formal verification* in the tool bar. The translation engine will automatically translate the model to a set of files storing the LTS accompanied with the assertion on safety critical properties, which can be verified directly by the tool Beagle. If the assertion is violated, a counter example will be reported.

4.4 Code Synthesis Engine

The synthesis engine is based on the code generation algorithm described in [12]. Engineers can select the block in the editor view of Fig 3, and press the button *automatic implementation* in the tool bar to generate VHDL code. For each atom block, a file with suffix **.vhd* is generated, defining the behavior of the block in VHDL. For each compound block, a file with suffix **.vhd* is generated to define the structure decompositions and connections contained in VHDL component map. The set of files can be synthesized and loaded into the FPGA processor directly. In this way, the consistency between the validated GalsBlock model and the implementation of real system can be maintained better. The productivity and useability will be improved. Moreover, for a function that can be modeled by both Galsblock and Stateflow, the size of the generated code is reduced by up to 40% compared to Simulink VHDL code generator of Stateflow.

5. EVALUATION ON REAL APPLICATION

We apply the toolkit to the design of a real MVB control system in the TCN. Traditionally, the companies such as Duagon and China North Railway (CNR) develop the controller by writing VHDL code directly according to the description of IEC 61375, which is time-consuming, error-prone and difficult to verify its correctness. For example, we have found several deadlocks in the VHDL code of CNR.

First, we build the GalsBlock model in the graphical editor strictly according to the algorithm and pseudo code descriptions in the standard. The constructed model of MVB controller system can be found at [14]. Unfortunately, the first version of the constructed model can not accomplish the process data communication service during model validation. Through manual analysis, we locate the problem in the atom block *mf_pool_ram_ctrl* and *send_device_status*. The problem in the model can be further traced back to the pesudo code in Table 33 and 35 of IEC 61375. The condition in the **IF** code segment is incorrect. The bugs have been certified through our previous theoretical analysis and engineering practice. The revised GalsBlock model passes the simulation and verification.

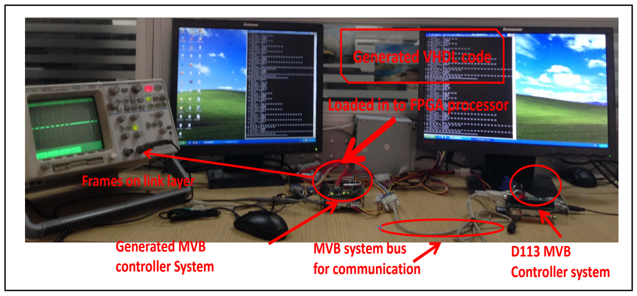
Generated VHDL code
Loaded in to FPGA processor
Frames on link layer
Generated MVB controller System
MVB system bus for communication
D113 MVB Controller system

Figure 5: Automatically Generated System Test.

Then, we generate VHDL code from the revised model through the code synthesis engine, and load them into the FPGA processor of the controller system. The generated code is efficient. For example, the size of VHDL code for the atom block *mf_generator_ctrl* is about 7 KB. If we use Stateflow to model this function and generate the code by Simulink, the size is about 12 KB, although the number of states are the same for the two models. Then, we use the most widely used MVB controller in the world – D113 from Duagon – to test the useability and reliability of the generated controller system. As presented in Fig 5, we use the application running on the industrial computer to monitor the communication. Furthermore, we also use oscilloscope to sample the data from the serial port that connected to the system bus. Both methods show that the generated controller system works well. In addition, the MVB controller designed by the Tsmart toolkit has already been deployed and in operation in railway control.

6. CONCLUSION AND FUTURE WORK

We developed the modeling, simulation, verification and code generation toolkit Tsmart to support the GalsBlock computation model for the design of multi-clocked embedded system. A prototype of the toolkit and the corresponding supporting files are available online [14]. Through the graphical model editor, we can build the structure and behavior model easily, which can be simulated graphically and verified formally to check the correctness. Another strength is the automatic code generation. This is effective because coding with low-level programming languages such as VHDL and C according to the requirements directly is more difficult and error prone. The toolkit will give a good guidance to reduce the complexity of the design of the complex embedded systems. In the future, we will extend Tsmart to generate C code, and generate test cases from the constructed model. Moreover, we will make the tool open source.

7. ACKNOWLEDGEMENT

This research is supported in part by NSFC Programs (No.61202010, No.91218302), National Key Technologies R& D Program (No.SQ2012BAJY4052) and 973 Program (No. 2010CB328003), and Tsinghua University Initiative Scientific Research Program (20131089331).

8. REFERENCES

[1] P. Amagbégnon, L. Besnard, and P. Le Guernic. Implementation of the data-flow synchronous language signal. In *Proceeding of the ACM SIGPLAN PLDI*, volume 30, pages 163–173. ACM, 1995.

[2] 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.

[3] 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.

[4] G. Berry. Circuit design and verication with esterel v7. In *HLVDT*, pages 133–136. IEEE, 2007.

[5] C. Brooks, E. A. Lee, and S. Tripakis. Exploring models of computation with ptolemy ii. In *IEEE/ACM CODES+ISSS*, pages 331–332. IEEE, 2010.

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

[7] F. Doucet, M. Menarini, I. H. Krüger, R. Gupta, and J.-P. Talpin. A verification approach for gals integration of synchronous components. *Theoretical Computer Science*, 146(2):105–131, 2006.

[8] P. L. Guernic, J. pierre Talpin, and J. christophe Le Lann. Polychrony for system design. *Journal for Circuits, Systems, and Computer*, 12:261–304, 2002.

[9] F. He, L. Yin, and B.-Y. Wang. *A Verifier for Component-Based Systems*, 11th automated technology for verification and analysis edition, 2013.

[10] G. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 23(5):279–295.

[11] Y. Jiang and etc. Design and optimization of multi-clocked embedded systems using formal technique. In *ESEC/FSE*, pages 703–706. ACM, 2013.

[12] Y. Jiang and etc. Design of mixed synchronous/asynchronous systems with multiple clocks. *IEEE Transactions on Parallel and Distributed Systems*, Accepted to appear:1–14, 2014.

[13] S. Ramesh, S. Sonalkar, V. Dsilva, N. Chandra, and B. Vijayalakshmi. A toolset for modelling and verification of gals systems. In *Proceeding of the International Conference on Computer Aided Verification*, pages 506–509. Springer, 2004.

[14] J. Yu and etc. Mvb example, vedio, user manual and toolkit download of tsmart. https://sites.google.com/site/jiangyu198964/home.