Safety-Assured Formal Model-Driven Design of the Multifunction Vehicle Bus Controller

Yu Jiang[1(✉)], Han Liu[1], Houbing Song[2], Hui Kong[3], Ming Gu[1], Jiaguang Sun[1], and Lui Sha[4]

[1] TNLIST, KLISS, School of Software, Tsinghua University, Beijing, China
jiangyu198964@gmail.com

[2] Department of Electrical and Computer Engineering, West Virginia University, Morgantown, USA

[3] Institute of Science and Technology Austria, Klosterneuburg, Austria

[4] Department of Computer Science, UIUC, Champaign, USA

Abstract. In this paper, we present a formal model-driven engineering approach to establishing a safety-assured implementation of Multifunction vehicle bus controller (MVBC) based on the generic reference models and requirements described in the International Electrotechnical Commission (IEC) standard IEC-61375. First, the generic models described in IEC-61375 are translated into a network of timed automata, and some safety requirements tested in IEC-61375 are formalized as timed computation tree logic (TCTL) formulas. With the help of Uppaal, we check and debug whether the timed automata satisfy the formulas or not. Within this step, several logic inconsistencies in the original standard are detected and corrected. Then, we apply the tool Times to generate C code from the verified model, which was later synthesized into a real MVBC chip. Finally, the runtime verification tool RMOR is applied to verify some safety requirements at the implementation level. We set up a real platform with worldwide mostly used MVBC D113, and verify the correctness and the scalability of the synthesized MVBC chip more comprehensively. The errors in the standard has been confirmed and the resulted MVBC has been deployed in real train communication network.

1 Introduction

The train communication network (TCN) enabling secure and fast data transmission in the entire rail vehicle has been standardized by the international railroad union and the International Electrical Commission, as presented in the international standard IEC-61375 [3]. Within the network, the multifunction vehicle bus controller (MVBC) is defined as a typical embedded software used mostly for the control of data transmission among the equipment (the traction control unit, air brake electronic control unit and door control unit etc.) onboard of each individual vehicle. Detail functions of the MVBC are based on the real-time protocol (RTP), which defines the rules (master-slave communication principle, data frame format and timing requirements, etc.) for process data and message data transmission.

J. Fitzgerald et al. (Eds.): FM 2016, LNCS 9995, pp. 757–763, 2016.
DOI: 10.1007/978-3-319-48989-6_47

Traditionally, from the perspective of industrial practice, most companies such as Siemens and Duagon develop their MVBCs by directly writing underlying C and VHDL code manually according to the description of IEC-61375, accompanied with the complex system and physical testing to avoid defects. Increasingly developed modern railroad vehicles increased the functional complexity, and are more difficult to ensure the correctness through testing. For example, even the most widely used D113 MVBC of Duagon company contains some dead logic in the C code for process data communication. From the perspective of academia, there are many existing works for the design of MVBC, but mainly focusing on the novel implementation hardware architecture [5]. In [5], they propose to use materialization of slave nodes for MVBC in a single chip by using reconfigurable logic. In [11], they propose to use BeagleBone and some existing tools such as Simulink to implement the MVBC, which is starting from model construction and ending in programming according to the validated model. Most of them focus on the functional implementation and do not pay attention on safety assurance under dynamic physical environment. Besides, there are also some works about verifying the real time communication protocol of TCN [6], they do not cope with the implementation issue neither, they focus on the logic correctness of train communication protocol only. Little research has been conducted to address the safety issue of MVBC, and some failures of the communication function have been reported to result in the accidents of the railway and trains [12], and some cases with serious injuries of human.

In this paper, we collaborate with the researchers from China Railway Rolling Stock Corporation (CRRC), and use formal model-driven development approach to establishing a safety-assured implementation of an MVBC prototype based on

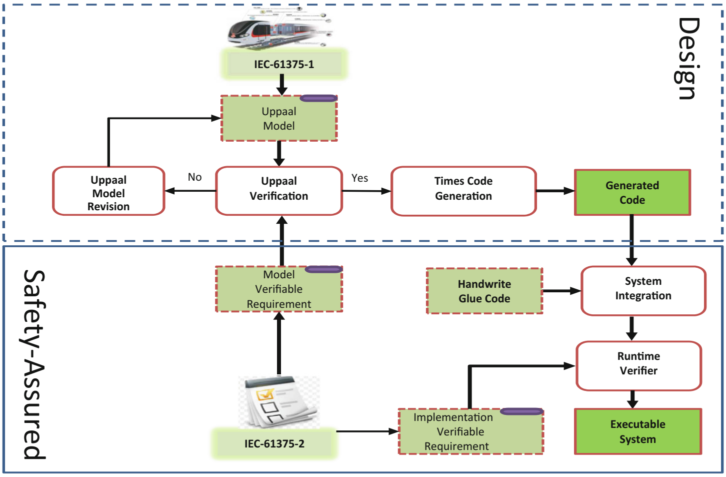

Fig. 1. Safety-assured design of MVBC. We may also replace these tools with similar functional tools such as SPIN and RV-Monitor. If we use SPIN to replace Uppaal, we need to build the SPIN Promela model instead of Uppaal timed automata model.

the standard IEC-61375, which consists of two parts: IEC-61375-1 describing the architecture and functional behaviors of MVBC, and IEC-61375-2 describing the conformance testing requirements. The overall procedure is presented in Fig. 1, where we leverage the formal modeling and verification technique as follows (1) the generic models and requirements in the standard are formalized as Uppaal timed automata and TCTL expressions [2] respectively, (2) formally verify the requirements and debug the models with Uppaal until the timed automata satisfies the TCTL expressions, (3) generate C code from the verified model with Times [1], which can be compiled and synthesized into a real MVBC chip with some auxiliary code developed to interface with hardware, and (4) use runtime verification to formally verify some implementation level safety requirements and test the consistency between the execution of the integrated system and the simulation of the verified model with RMOR [4]. Then, we set up a real platform, connecting the synthesized MVBC prototype for safety assured communication demonstration. During the practice, the errors detected in the standard has been confirmed and the synthesized prototype has been in productization and deployed in real train system control.

2 Safety-Assured Approach

Model construction and verification: First, we build a network of timed automata for the MVBC according to the architecture and functional description, such as the generic automata model, the function and action table, and the SDL (Specification and Description Language) diagram of IEC-611375-1. All these heterogeneous information are unified translated and encoded into the network of timed automata manually. Currently, it is not easy to automatically abstract the timed automata model from the text-based standard, and the whole construction procedure is manually accomplished and validated with the help of engineers from CRRC with the following modeling guidance rules.

Let us see the translation of generic automata and the accompanied function table. Each state in the generic automata is mapped to an ordinary location in Uppaal with the same name. For the packets of sending and receiving events with actual parameters specified for the control fields, we use the synchronous channel of Uppaal timed automata to simulate the communication. Because there are lots of none interrupt actions and packets associated with a single generic state while only one synchronous action is allowed to be attached in a single transition of Uppaal timed automata, we need to create a set of committed locations, where none interrupt actions are sequentially encoded into the transitions among those committed locations. Then, the attached actions described in the function table of the standard, they are translated into the accompanied actions of the Uppaal timed automata transition.

For the translation of SDL diagram, each state in the diagram is mapped to an ordinary location of Uppaal. Some plain C codes in the diagram are translated into the action attached on the transition of two locations. The event signal of SDL diagram is modeled by the synchronous channel of timed automata, where receiving an event is denoted as $Rcv\_Channel\_Name$? and sending an event is denoted as $Send\_Channel\_Name$!. In case of situations with more than two signals between two states, we need to add some intermediate locations of timed automata. Note that, for the clock signal, it is issued by itself. Hence, we do not need to translate it into a synchronous channel.

Safety Requirements Formalization: The MVBC safety requirements are mainly derived from the descriptions of the MVBC conformance testing requirement of IEC-61375-2. We divide the testing requirement into two groups: *model verifiable safety requirement* and *implementation verifiable safety requirement.*

Those requirements that are related to general functions of control logic and independent of platform are categorized as *model verifiable safety requirements*. We formalize them as timed computation tree logic formulas defined on the formal timed automata, and verify them in Uppaal. For example, the requirement that there is at most one regular master MVBC contained in the train communication network, is a typical *model verifiable safety requirements*, and can be formalized as $A[\ ]not(MVBC(1).Regular\_Master \&\& MVBC(2).Regular\_Master)$.

Those requirements related with dynamic runtime situation and uncertain environment are categorized as *implementation verifiable safety requirements.* They are not easy to be defined in the abstract timed automata level, because it is not easy to model dynamic transmission delay of data on MVBC bus and dynamic processing delay of hardware platform, even with a preliminary channel model and clock variable in Uppaal timed automata. We formalize these safety requirements as the runtime verification property of RMOR. We define some events based on the variables of the generated C code of Times, which are configured to I/O pins of the real hardware platform and will be continuously loaded by accompanied C functions. Then, the property and the accompanied C functions are transformed and input to RMOR to get the instrumented code, which can be made as an integral part of the target generated system, verifying and guiding its execution within the dynamic environment. For example, the requirement that he suggested time constraint on a master MVBC between the finish of a master frame sending and the start of a response slave frame receiving should be less than 42.7us is a typical *implementation verifiable safety requirements*, and can be formalized as below:

```
DataCenter Monitor TimeConstraints(){ ......
    event TimeoutResponse =
     ((T_Master_Send - T_Slav_Receive)<42.7)
    event Trigger = TimeoutResponse;
    state safe{When Trigger -> error;}
}
```

Listing 1.1. Runtime Property Definition for the Time Interval.

Code synthesis and verifier integration: For the code synthesis, automatical code generation tools such as Times can be applied to reduce the hard work efforts of manual implementation, which is also more human error prone. For example, the engineers from the industrial sources (the Duagon company, the China CR corporation) report that their MVBC is developed by directly writing underlying C or VHDL code manually, where there are still some bugs such as dead logic or dead code. Besides, the automatical code generation also facilitate the traceability between the model and implementation, which results in better documentations and easier maintains.

Before applying the code generation algorithm, we need to do some changes on the formal model. More specifically, we construct and initialize the timed automata template for two or more MVBCs for comprehensive verification, and now need to isolate the timed automata of a single MVBC for code synthesis. One way for isolation is to build a general environment model, which is ready to receive any output synchronization action from the isolated MVBC and send input synchronization action to the isolated MVBC. Then, we can generate execution code for both MVBC and the general environment, and manually separate the generated code. Another way for isolation is to do some reverse engineering, where the synchronization channels denoting the packets of sending and receiving events are reversed to the general variable. For example, a synchronization channel rcv_connect_req? can be replaced by a declaration of boolean variable rcv_connect_req. Meanwhile, an evaluation expression rcv_connect_req == true should be added to the guard segment, and an assignment expression rcv_connect_req := true should be added to the action segment. We use the second way, because it can be automatically accomplished by parsing and updating the XML file of the timed automata model, and the second isolation way is more closed to the real operation scenario where the sending and receiving packets from the physical bus is asynchronous. Besides, because the generated code is tightly coupled, the manually separation is more error prone.

After that, we also need to add some glue code, which is mainly used for two functionalities, the interface between the software and hardware platform, and timing implementation of the generated code on the hardware platform. For interface, we just need to initialize some configure mapping files, mapping the variable of software to the GPIO of the hardware platform. Accompanied type conversion functions may be needed. For clocks, let sc be a global system clock. For each clock x in the timed automata, let x_{reset} be an integer variable holding the system time of the last clock reset. The value of the clock is then $(sc - x_{reset})$, and a reset can be performed as $x_{reset} := sc$.

Finally, based on the generated code and the handwriting glue code, we input the formalized implementation verifiable safety requirement and the integrated code to RMOR to generate the runtime verifier, and the system integration is instrumented with the verifier for the runtime verification. The integrated verifier keeps verifying the safety requirements on the running executable system. To improve the safety confidence, we can also formalize some model verifiable requirement into verifier, but will increasing the storage overhead of the system.

Table 1. Resource utilization C compilation for MVBC, and the verification efficiency.

C compilation	Safety-Assured	BeagleBone
Binary File Size KB	302	683
Bug in IEC Standard Detected	6(verification)	1(Simulink Design Verifier)
Injected Division by Zero Detected	10/10(verification)	4/10(Simulink Design Verifier)

3 Experiment Results

To evaluate the effectiveness of the proposed approach, we apply it to the design of MVBC and compare it with BeagleBone [11], which is the most recently available design framework for MVBC based on Simulink. More specifically, we formalize 92 critical model verifiable safety requirements and 29 critical implementation verifiable safety requirements. During the verification process of the proposed approach, 11 requirement violates in the model or the implementation level. After discussion with the engineers from CRRC, 5 requirements are violated because of the error brought by our modeling behavior, and 6 requirements are violated because of the error of the control logic described in the standard. While in the verification process of BeagleBone, only one violation is detected due to the limited specification and verification of Simulink Design Verifier. For the second type of violation, we need to revise the timed automata model as well as the back end IEC standard according to analysis results of counter examples. Besides, these violations are consistent to existing works [7–10] and have already been confirmed and would be revised in the new version of IEC standard 61375. After revision, both the model level verification and the runtime verification reports no violation (Table 1).

Then, the generated code according to the revised model and the integrated executable system with the eCos (Embedded Configurable Operation System) is synthesized. Then, the synthesized binary files for the integrated C code can be loaded and run on ARM7-STM32F407IGH6 processor. The binary file is 302 kb and 683 kb for the code generated by Times and BeagleBone respectively. The difference is mainly derived from the fact that BeagleBone use Simulink C code generator to generate many extra configuration files and introduces many libraries for scalability. To test the reliability of the system as well as some requirements that can not be formalized, we connect the widely-used industrial product MVBC card D113 with our synthesized MVBC for real-time communication. We use the application running on the industrial computer to monitor communication, and read the message data from memory. It shows that the communications confirm to the requirements defined in the part two of standard IEC 61375.

4 Conclusion

In this paper, we present a formal model-driven engineering approach to establishing a safety-assured implementation of MVBC based on the generic reference models and requirements described in the International Electrotechnical Commission (IEC) standard 61375. The design part mainly includes formal model construction, code generation and integration, and the safety-assured part mainly includes model level verification and implementation level verification. During the engineering practice, several logic inconsistencies in the original standard are detected and corrected.

Acknowledgements. 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), Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE) and Z211-N23.

References

1. Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., Yi, W.: TIMES b— A tool for modelling and implementation of embedded systems. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 460–464. Springer, Heidelberg (2002). doi:10.1007/3-540-46002-0_32
2. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30080-9_7
3. International Electrotechnical Commission et al.: IEC 61375-1, Train Communication Network (2011)
4. Havelund, K.: Runtime verification of C programs. In: Suzuki, K., Higashino, T., Ulrich, A., Hasegawa, T. (eds.) FATES/TestCom -2008. LNCS, vol. 5047, pp. 7–22. Springer, Heidelberg (2008). doi:10.1007/978-3-540-68524-1_3
5. Iturbe, X., Zuloaga, A., Jiménez, J., Lázaro, J., Martín, J.L.: A novel SoC architecture for a MVB slave node. In: IECON 2008. IEEE (2008)
6. Jiang, Y., Gu, M., Sun, J.: Verification and implementation of the protocol standard in train control system. In: IEEE 37th Annual Computer Software and Applications Conference (COMPSAC), pp. 549–558 (2014)
7. Song, H., et al.: Data-centered runtime verification of wireless medical cyber-physical system. IEEE Transactions on Industry Informatics (2016)
8. Yang, Y., et al.: 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) (2016)
9. Zhang, H., et al.: Design and optimization of multi-clocked embedded systems using formal technique. IEEE Trans. Ind. Electron. **62**(2), 1270–1278 (2014)
10. Jiang, Y., et al.: Design of mixed synchronous/asynchronous systems with multiple clocks. IEEE Trans. Parallel Distrib. Syst. **26**, 2220–2232 (2014)
11. Aarthipriya, R., Chitrapreyanka, S.: FPGA implementation of multifunction vehicle bus controller with class 2 interface and verification using Beaglebone Black (2015)
12. Yunxiao, F., Zhi, L., Jingjing, P., Hongyu, L., Jiang, S.: Applying systems thinking approach to accident analysis in China: case study of "7.23" Yong-Tai-Wen high-speed train accident. Saf. Sci. **76**, 190–201 (2015)