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

Yu Jiang, Han Liu, Houbing Song, Hui Kong, Rui Wang, Yong Guan, and Lui Sha

***Abstract*— In this paper, we present a formal model-driven design approach to establish a safety-assured implementation of multifunction vehicle bus controller (MVBC), which controls the data transmission among the devices of the vehicle. First, the generic models and safety requirements described in International Electrotechnical Commission Standard 61375 are formalized as time automata and timed computation tree logic formulas, respectively. With model checking tool Uppaal, we verify whether or not the constructed timed automata satisfy the formulas and several logic inconsistencies in the original standard are detected and corrected. Then, we apply the code generation tool Times to generate C code from the verified model, which is later synthesized into a real MVBC chip, with some handwriting glue code. Furthermore, the runtime verification tool RMOR is applied on the integrated code, to verify some safety requirements that cannot be formalized on the timed automata. For evaluation, we compare the proposed approach with existing MVBC design methods, such as BeagleBone, Galsblock, and Simulink. Experiments show that more ambiguousness or bugs in the standard are detected during Uppaal verification, and the generated code of Times outperforms the C code generated by others in terms of the synthesized binary code size. The errors in the standard have been confirmed and the resulting MVBC has been deployed in the real train communication network.**

***Index Terms*— Multifunction vehicle bus, train communication network, model-driven development, IEC-61375.**

## I. Introduction

THE train communication network (TCN) enabling secure and fast data transmission in the entire rail vehicle [28], [32], [36], has been standardized by the International Railroad Union and the International Electrical Commission, in the international standard IEC-61375 [10]. The standard describes the main roles and communication rules of the network, where the multifunction vehicle bus controller (MVBC) is defined as a typical embedded software used for the control of data transmission among the equipment (the traction control unit, air brake electronic control unit and door control unit etc.) of each individual vehicle, and the real-time protocol (RTP) is defined as the rules (master-slave communication principle, data frame format, and timing requirements, etc.) for process data and message data transmission of MVBC.

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 modern railroad vehicles increased the functional complexity, which has been divided into class 1 to class 5 of MVBC in the standard. The higher class number of the controller means higher function complexity and difficulty to ensure the correctness through testing. For example, even the most widely used D113 MVBC of Duagon company contains some dead logic in the VHDL code for process data communication and C code for message data communication [31]. From the perspective of academia, there are many existing works for the design of MVBC, but mainly focusing on the novel implementation hardware architecture [13], [21], [25], [26]. Little research has been conducted to address the safety issue, and some failures of the communication function have been reported, resulting in the death of involved human [33], [35].

### A. Proposed Approach

In this paper, we collaborate with the researchers from China Railway Rolling Stock Corporation (CRRC), and use formal model-driven development (MDD) approach to establish a safety-assured implementation of an MVBC prototype based on the generic reference models and requirements described in the standard IEC-61375. MDD approach is a widely used software development method starting from abstract model construction, to model validation, and end in automatically code synthesis. We make the following customizations 1) the generic models and requirements in the standard are formalized as timed automata [2] and TCTL expressions,[1] respectively, 2) formally verify 92 requirements and debug the model with Uppaal [6] until the timed automata satisfies those TCTL expressions, and 3) automatically generate C code from the validated model with Times [4], which

Manuscript received June 23, 2016; revised December 28, 2016 and May 24, 2017; accepted November 11, 2017. Date of publication January 17, 2018; date of current version October 3, 2018. The Associate Editor for this paper was D. Cao. *(Corresponding author: Rui Wang.)*

Y. Jiang and H. Liu are with the School of Software, Tsinghua University, Beijing 100084, China.

H. Song is with the Department of Electrical and Computer Engineering, West Virginia University, Morgantown, WV 26506 USA.

H. Kong is with the Institute of Science and Technology Austria, 3400 Klosterneuburg, Austria. (e-mail: konghuitsinghua@126.com).

R. Wang and Y. Guan are with the College of Information Engineering, Capital Normal University, Beijing 100000, China (e-mail: rwang04@163.com).

L. Sha is with the Department of Computer Science and Technology, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2017.2778077

[1] TCTL (Timed computation tree logic) expression is used to specify properties to be checked with respect to a timed graph. It is interpretation over continuous computation trees, trees in which paths are maps from the set of nonnegative reals to system states of the graph [1].

1524-9050 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

can be compiled and synthesized into a real MVBC chip with some auxiliary code developed to interface with hardware. 4) use runtime verification to formally verify 26 implementation level safety requirements and test the consistency between the execution of the integrated system and the simulation of the verified model with RMOR [12]. Then, we set up a real platform, connecting the synthesized MVBC prototype with the worldwide mostly used MVBC D113 for comparison with existing MVBC design methods.

### *B. Main Contribution*

Overall contributions of our work are:

1) We propose a safety-assured model driven approach for the design of MVBC, where safety requirements in IEC-61375-2 are categorized as the requirements that can be verified on the abstract formal model, and the requirements that can only be verified on the synthesized underlying implementation with runtime verification technique.
2) We present a detailed case study on formalizing the generic informal model (ordinary automata, SDL diagram) of IEC-61375-1 into timed automata, validate and debug the timed automata for correctness, and synthesis and integrate code. We believe that the approach and the case study provide a guidance for the future design of MVBC complying to the standard, or the design of systems complying with similar standards.

### *C. Paper Organization*

The paper is organized as follows: the MVBC software and the corresponding safety requirements are introduced in Section II. Related works about model-driven design and MVBC design are presented in Section III. Our proposed safety-assured MDD approach is presented in Section IV, including formalization of generic model and requirements, model verification and debug, and code synthesis and integration with runtime verifier. Experiment results performed on the communication between the safety-assured prototype and the widely used MVBC D113 are given in Section V, and we conclude in Section VI.

## II. The MVBC System

### *A. TCN Topological Overview*

The TCN encompasses two buses to interconnect programmable equipment onboard rail vehicles for the support of traction and vehicle control, remote diagnostics and maintenance, and passenger information and comfort. The whole structure of TCN is presented in Figure 1. The first bus is the multifunction vehicle bus (MVB) which interconnects devices within a vehicle, a bus optimized for fast response, operating at 1.5 Mbits on twisted wires or optical fibers. The second is the wire train bus (WTB) which interconnects the vehicles of a train, a bus capable of self-configuration, operating over twisted wires at a speed of 1.0 Mbits. These two buses offer the same services: 1) cyclic, source-addressed broadcast *Process Data*, and 2) on-demand, destination-addressed *Message Data*.

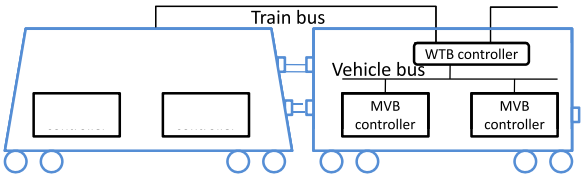



Fig. 1. Topological view of the MVBC and WTBC in the train communication network.

*Process Data* and *Message Data* is on the same bus in the devices, but they are transmitted alternately and never together. With this, the transmission time of the bus is divided into basic periods, in which it is specified which data is sent.

The multifunction vehicle bus controller (MVBC) implements the real-time protocol (RTP) to provide the information exchange of variables (distributed *Process Data*) and messages (call/reply and multicast *Message Data*) based on master-slave principle. Variables are broadcast in cycles to all devices on the bus, while the messages are transmitted without real-time demand. The wire train bus controller (WTBC) implements the same communication function but with external routing ability. The MVBC is categorized into five classes according to their capabilities. The MVBC of class 1 supports the transmission of *Process Data*, the MVBC of class 2-4 supports the transmission of both *Process Data* and *Message Data*, and the MVBC of class 5 extends them with the master ability to poll and access at least another MVB bus. In this work, we focus our research on MVBC class 5, which is more urgent in modern complex applications. Basically, the master MVBC sends a master frame with a specific address of the variable. Each participant slave MVBC checks whether the address is subscribed or not. They will compare their MVBC device address with the address contained in the master frame. If yes, the value of the requested variable is encoded into a slave frame and responded by the subscribed slave MVBC producer. Then, all other slave MVBC consumers will accept the slave frame and update the variable.

### *B. MVBC Communication Function Model*

Detailed functions of the MVBC are mainly 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. The MVBC communication function model is an abstract representation of these typical behavior rules of MVBCs, which are well defined in IEC-61375-1.

As presented in Figure 2, the abstract system architecture model contains the `User Application`, `MVBC State Controller`, and `Physical Link Bus`. The `MVBC State Controller` mainly contains three components: `Master Transfer`, `Message Sender` and `Message Receiver`. The components `Message Sender` and `Message Receiver` are responsible for transmitting the message from a producer to a consumer, and provide the flow control and error recovery from end to end through a sliding window. The transmission of message is divided into three phases with related communication primitives: connection

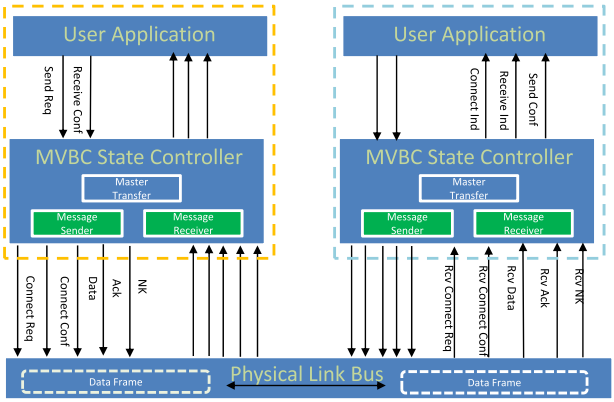



Fig. 2. System architecture model of the MVBC, where communication primitives are on the arrow.

establishment (*Connect Req, Connect Conf, etc.*), acknowledged data transmission (*Data, Ack, Rcv Data, etc.*), and disconnection (*DisConnect Req, DisConnect Conf, etc.*). The component `Master Transfer` selects a master MVBC from one of the several MVBCs with bus administrator ability at the end of a macro period. These three components embody the core function of an MVBC. The information of the MVBC, i.e. device address and master frame sequence, is initialized during deployment. Detailed description about the abstracted model can be referred to IEC-61375-1.

## III. Related Work

### A. MVBC Design

During the past decades, many scientists have paid many efforts to the design and implementation of the MVBC [13], [21], [26]. In [13], they propose to use materialization of slave nodes for MVBC on a single chip by using reconfigurable logic. In [17] and [25], they propose to use some existing tools such as Simulink to help 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 to safety assurance under dynamic physical environment. Besides, there are also some works about verifying the real-time communication protocol of TCN [15], [16], [19], [27], they describe some formal methods to verify safety properties of the communication protocol used in train control system. But they do not cope with the implementation issue, they focus on the logic correctness of train communication protocol only. The engineers from the industrial sources (the Duagon company, the China CR corporation) report that their MVBC is developed by directly writing underlying C and VHDL code manually, accompanied with some testing.

### B. Model-Driven Design

Except for the work for MVBC, there are large amounts of work and various toolkits in the literature supporting the model driven design of general systems. For example, SCADE [7] uses 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 [11], [23] is now widely used with Stateflow as its basis. It presents strong modeling, simulation and synthesis capability, but has no formal semantics for comprehensive verification on safety-critical applications even with the support of Design Verifier [23]. Except for the two famous industrial frameworks, Academic tools such as Ptolemy [9] and POLIS [5] do well in modeling and simulation of heterogeneous systems. A UML-based approach was proposed to the design of hard real-time systems, which maps the general UML notations to a platform-dependent model for code synthesis. However, those works alone are not suitable for our safety-assured MVBC design. For example, although SCADE has well-certified code generator, its verification ability can not support all safety requirements verified on the model.

### C. Safety-Assured Analysis and Design

There are lots of work for safety analysis techniques and address the safety issue in design [14], [20], [27]. For example, in [34], they describe an approach to software architecture design for safety-related systems, mainly focusing on defining an analytic model to analyze software safety at the architectural level and presenting the development of safety tactics. For research on mitigating detail safety requirements, works [29], [30] propose a novel lease based design pattern, to guarantee proper temporal embedding safety rules under arbitrary wireless communication faults. They transform the design pattern hybrid automata into specific wireless CPS designs. These works perform well at the early stage of system design, but do not cover the real implementation.

## IV. Safety-Assured Approach

### A. Design Approach Overview

The overall procedure about the proposed safety-assured model-driven development of MVBC is presented in Figure 3, which mainly relies on the formal modeling and analysis tool Uppaal,[2] code generation tool Times,[3] and runtime verification tool RMOR.[4] We integrate these tools together to validate some model verifiable safety requirements on the Uppaal model of MVBC, automatically generate code from the verified model, and verify the implementation verifiable requirement on the integrated system. Details of each step are presented as follows.

First, the timed automata model is constructed in Uppaal, according to the architecture and functional description of MVBC, such as the generic automata model and table presented in Figure 30 and Table 32 of IEC-611375-1. At the same time, the model verifiable safety requirements described in IEC-61375-2 are formalized into the timed computation tree logic formulas in the format of Uppaal. Then, the safety

[2]Uppaal is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata.

[3]Times is a tool set for modeling, schedulability analysis, synthesis of schedules and executable code with task extended timed automata.

[4]RMOR supports monitoring C programs against state machines, using an aspect-oriented pointcut language to perform program instrumentation and connect the events occurring in state machines with code fragments.

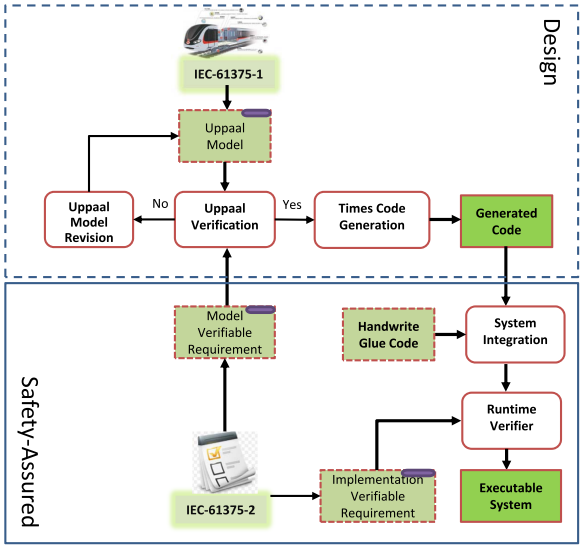



Fig. 3. Safety-Assured Design for the MVBC.

requirements can be verified on the constructed model. We need to revise the Uppaal model until the properties are satisfied.

Then, we can apply code generators for timed automata model to generate the underlying code directly. There are lots of code generators for timed automata, and we choose Times here [3], [4]. Times is co-developed by the group member of Uppaal and can generate platform-independent C code, which keeps the semantics consistency of the original Uppaal model and can be easily customized to a dedicated hardware platform with the integration of handwriting glue code. The glue code is used for the communication with the interface of the hardware, and the timing mapping and implementation of the generated code.

Finally, based on the integrated system, we can use runtime verifier to verify those implementation verifiable safety requirements related to dynamic runtime situation and uncertain environment. The requirements are formalized as event definition and state machine property definition based on the integrated code, in the input format of RMOR. Then, the original integrated code will be instructed with the generated verifier for runtime verification, after which, the final executable system can be deployed.

### *B. Timed Automata Model Construction*

We build a network of timed automata for the MVBC according to the architecture and functional description, such as the generic automata model in Figure 30, the function table presented in Table 32, and the flowchart in Figure 105 of IEC-611375-1. All the heterogeneous information is unified translated and encoded into the network of timed automata manually, which can be formally verified and automatically synthesized later. 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.

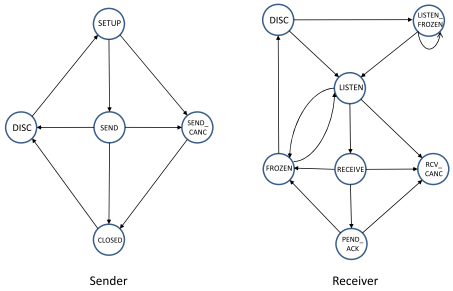



Fig. 4. Generic automata for the sender and receiver module of MVBC.

As presented in Figure 2, there are mainly three modules contained in the MVBC. We can build a single automaton for the three modules, but in this way, it is not easy for model validation. Hence, we build an automaton for each module, and those automata communicate with each other through synchronous channels and shared variables. This facilitates not only the model validation, but also the structured code generation. Besides, we also need to model some potential fault factors such as time delay and packet loss of the physical link bus. Details about the constructed automata are presented as below.

*1) Timed Automata Model for Sender-Receiver:* These two automata model the components `Message Sender` and `Message Receiver`, which implements the real-time communication protocol, and are responsible for safely transmitting the message from an MVBC (producer) to another MVBC (consumer). The generic models of the two components are in Figure. 4, and are accompanied with lots of tables and sequence diagrams in the original standard. For example, both sender and receiver are in DISC state, the sender of the producer may send a connect request and transmit to the SETUP state, and the receiver of consumer keeps listening to the connect request and transfers to the LISTEN state when the connect request confirm packet is received. When the sender of the producer transfers to the SEND state and the receiver of the consumer transfers to the RECEIVE state, the acknowledgment data transmission starts. LISTEN state represents request confirmation, SEND_CANC and RCV_CANC states represent communication cancelation, and FROZEN state represents normally acknowledgment.

Based on the generic automata and the accompanying information, we can construct the timed automata. Each state in the generic automata is mapped to an ordinary location 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. For the attached actions as described in Table 32 of the standard, they are translated into the accompanying actions of the Uppaal timed automata transition. Furthermore, the timing information defined in

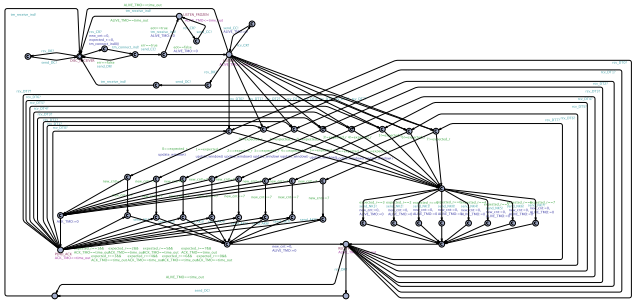


Fig. 5. Constructed timed automata for the receiver module of the MVBC.

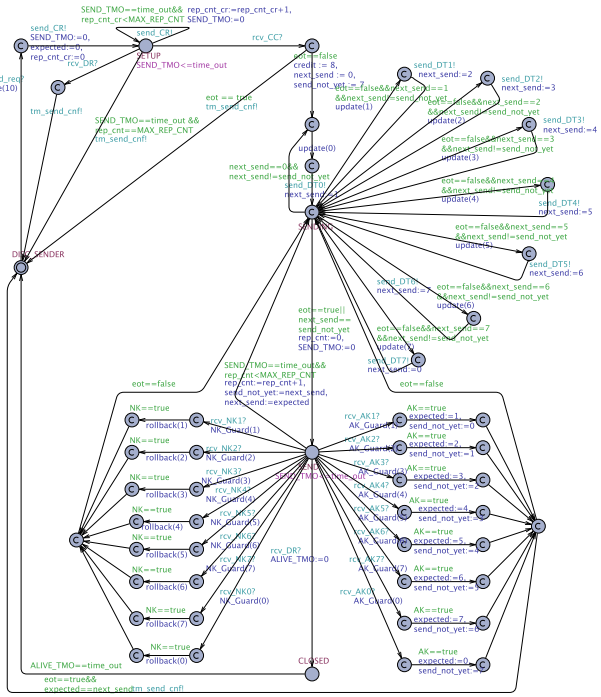



Fig. 6. Constructed timed automata for the sender module of the MVBC.

the standard such as the three timer SEND_TMO (the send timeout at the producer, initialized as 14), ACK_TMO (the acknowledge time-out at the consumer, initialized as 25), and ALIVE_TMO (alive time-out at the consumer, initialized as 95) are declared as clock variable in the timed automata, which can be started and reset on the transition. Other parameters such as MAX_REP_CNT (maximum value for the repetition count) are also initialized in the timed automata. Finally, we can get the two timed automata as presented in Figure. 5, 6.

We also need to build a channel timed automaton for the Physical Link Bus component for the communication between the sender and receiver. Within this channel timed automaton, the packet transmission delay, packet loss and packet retransmission can be captured. For example, when the sender sends the connect request packet through the send_connect_req! channel, the send_connect_req? will be synchronized in the channel timed automaton. Then, this channel automaton will further trigger a rcv_connect_req! channel with a probability and a time delay, which will then trigger the synchronous channel rcv_connect_req? in the receiver channel.

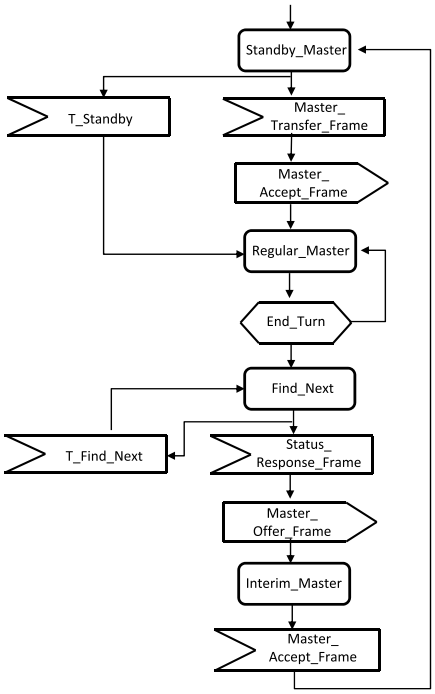



Fig. 7. Generic SDL model for the master transfer module of the MVBC.

If the packet is lost, the channel automation will not trigger the rcv_connect_req! channel. Similar mechanisms can be constructed for other events and packets, and encoded in this single channel automaton.

*2) Timed Automata Model for Master Transfer:* This timed automaton models the component `Master Transfer`, which accomplishes the task that selects a master MVBC from one of several bus administrators, and ensures mastership transfer at the end of a turn or upon the occurrence of a failure. Usually, at the end of a predefined macro period, current master MVBC will give up control ability, and a slave MVBC will be rotated in sequence as the new master to control message communications. The generic model of the master transfer is described as an SDL(Specification and Description Language) diagram as abstracted in Figure.7. For a standby master MVBC, it starts in the Standby_Master state, keeps listening to the signal of Master_Transfer_Frame and T_Standby, and transfers to the Regular_Master state. While for the current master MVBC in control, it starts in the Regular_Master state, finds a standby master MVBC for the next period at the end of this turn, and transfers to the Standby_Master state. More details can be referred to the description in the standard.

Then, we construct the timed automata based on the generic SDL diagram. Each state in the diagram is mapped to an ordinary location. Some plain C codes in the diagram are translated into the action attached on the transition of two locations. The event signal is modeled by the synchronous channel of timed automata, where receiving an event is denoted as Rcv_Channel? and sending an event is denoted as *Send_Channel*!. 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,

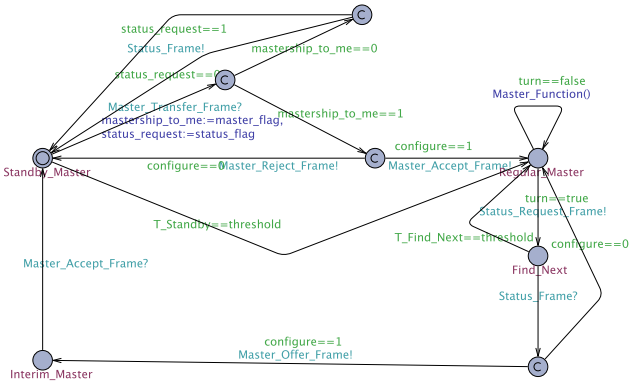



Fig. 8. Refined timed automata for the master transfer of MVBC.

it is issued by itself. Hence, we do not need to translate it into a synchronous channel. A clock variable with a predefined threshold is declared for each clock signal such as T_Standby signal (receives no master frames during a timeout T_standby). Finally, we can get the timed automata as presented in Figure. 12.

### C. Safety Requirements Formalization

The MVBC safety requirements are mainly derived from the descriptions of the MVBC conformance testing standard IEC-61375-2, accompanied with some hazard analysis in the real use of MVBC. For example, there are two safety requirements described in natural language as below:

1) ***Message Transmission:*** During the acknowledged data transfer stage, when some packets are lost in the physical link layer, they should be retransmitted.
2) ***Master Transfer:*** During the master transfer procedure, there is one and only one master MVBC contained in the train communication network.

The first requirement is related to the correctness of the acknowledged data transfer stage, where the producer sends the individual data packet of the message to the consumer. The consumer side may bundle acknowledgments. For example, the consumer may acknowledge several packets at the same time by acknowledging the packet with the highest sequence number only. When packets fail to be acknowledged, the producer shall retransmit them. Besides, the consumer may indicate to the producer that it receives an out of sequence packet. In this case, the consumer shall send a *Negative Acknowledgment* packet, indicating from which packet on it requires retransmission. For the second requirement, it is originated from the fact that the mastership can be shared by two or more MVBCs with administrator ability, which each exercises mastership for the duration of a turn. Also in case of failure of a master MVBC, mastership should be transferred to another MVBC. But it is not allowed that two MVBCs act as the master in the same time.

In the original development process, the requirements are tested through simulation as defined in the standard IEC-61375-2. Although the general methodology and procedures are well specified to test that the implemented MVBC is confirmed to the function described in IEC-61375-1, the result is highly dependent on the input patterns of the test cases, where the coverage of some extreme conditions may be ignored and hard to be enumerated due the limited number of input patterns. While in safety-assured development approach, we try to ensure that the MVBC implementation satisfies the requirements with formal verification, which is more rigorously than simulation-based testing.

*1) Formalization of Model Verifiable Requirement:* These requirements that are related to general functions of control logic and independent of the platform are categorized as model verifiable safety requirements. We formalize the requirements as timed computation tree logic formulas defined on the formal model, and verify them on the formal timed automata. Let us take the two safety requirements described in the natural language above as an example. The safety hazards of the first requirement happen during the data acknowledgment process and the data retransmission process. The safety hazard of the second requirement happens during the MVBC master rotation process.

For the retransmission process, we formalize the requirement as two properties based on the timed automata presented in Figure. 5, 6. When the SENDER automaton receives the rcv_NKi? signal, it will decide whether the sequence i lies in the legal interval or not. The decision logic is implemented on the guard of transition as (expected $<$ NK_number $\leq$ send_not_yet). The value of this decision expression is assigned to a variable $NK$. At the same time, when the RECEIVER automaton sends a legal send_NKi! signal and switches to the SEND_NK state, the SENDER automata should decide the sequence number $i$ to be true with boolean evaluation ($NK == true$) and be able to deliver the retransmission request. This property is formalized as P1 in table 1. Another property for the retransmission process is about rolling back the window of the SENDER automata in the retransmission process. After receiving the rcv_NKi? signal, the SENDER automaton will roll back the sending sliding window and retransmit the previous data packets from the sequence number i, Which means that the next_send data packet of the SENDER module must equal to the value of the NK_number. This property is formalized as P2 in table 1.

In the same way, we can formalize two properties for the data acknowledgment process of the first requirement and three properties for the MVBC master rotation process of the second requirement. All these requirements and their formalization are presented in Table I. For example, the property P6 means that in the MVB master and slave rotation process, there may be inconsistency such that two masters appear at the same time.

*2) Formalization of Implementation Verifiable Requirement:* These requirements related with dynamic runtime situation and uncertain environment are categorized as implementation verifiable safety requirements. For these safety requirements, they are not easy to be defined in the abstract timed automata level, and we use runtime verification technique and formalize the safety requirements based on the detail implementation code to verify the correctness. Let us look at the two safety requirements below, which are described in the natural language in the original standard. These two safety requirements are not easy to be captured in model level, because it is not

TABLE I
PROPERTY LIST

| Property | Formula |
|---|---|
| P1 | $A[](RECEIVER.SEND\_NK \rightarrow (NK == true))$ |
| P2 | $A[]((NK == true) \rightarrow (NK\_number == next\_send)$ |
| P3 | $A[](RECEIVER.SEND\_AK \rightarrow (AK == true))$ |
| P4 | $A[]((AK == true) \rightarrow (AK\_number == next\_send))$ |
| P5 | $A[]((MVBC(1).Regular\_Master \&\& MVBC(2).Standby\_Master)\|\| (MVBC(2).Regular\_Master \&\& MVBC(1).Standby\_Master))$ |
| P6 | $A[]not(MVBC(1).Regular\_Master \&\& MVBC(2).Regular\_Master)$ |
| P7 | $A[]not(MVBC(1).Standby\_Master \&\& MVBC(2).Standby\_Master)$ |

```
DataCenter Monitor TimeConstraints()
{
  P8 event TimeoutReply =
   ((T_Master_Receive-T_Slav_Send)<4)
  P9 event TimeoutResponse =
   ((T_Master_Send-T_Slav_Receive)<42.7)
  event Trigger =
    TimeoutReply || TimeoutResponse;
  state safe{
    When Trigger -> error;
  }
}
```

Listing 1. Runtime Property Definition for the Time Interval Between the Master and Slave Frames During the Message Transmission Process

easy to model dynamic transmission delay of data on MVB bus and dynamic processing delay of hardware platform, even with a preliminary channel model and clock variable in Uppaal timed automata.

1) ***Message Transmission(P8):*** The suggested time constraint on a slave MVBC between the finish of a master frame receiving and the start of a slave frame responding should be less than 4us.
2) ***Message Transmission(P9):*** The 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.

We formalize these two safety requirements as the runtime verification property presented in the Listing 1. 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.

In this way, we can formally verify not only the requirements related to general function of control logic but also the requirements related to the dynamic physical part. Note that, the Uppaal model verification and RMOR runtime verification cooperate together to acquire a higher safety confidence for safety critical systems. More specifically, we formalize 92 critical model verifiable safety requirements and 29 critical implementation verifiable safety requirements. The main criteria to distinguish between the two types of requirements is to find out whether the timed automata model has sufficient information for describing the requirement in TLTL format or the integrated code has sufficient information for describing the requirement in RMOR property format. The timed automata usually rely on the different level of abstraction(state and variables) and are not detailed enough to specify some requirements such as time delay restrictions on the physical bus. Which means that when all elements in the TLTL formula of the text-based requirements are described in the timed automata, it is the first type of requirement, otherwise, it is the second type of requirement. Besides, there are also several requirements that can not be formalized such as the requirement of the industrial grade type of the MVBC hardware chip. These requirements need to be manually checked or formalized with additional information from additional sensors.

### *D. Code Synthesis and Integration*

For the code synthesis, automatical code generation tools 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. Besides, the automatical code generation facilitates the traceability between the model and implementation, which results in better documentations and easier maintains. There are many code generators for timed automata, and we choose Times. Times is co-developed by the group member of Uppaal, which we believe that keep the semantics consistency and retain the verification benefits of the original Uppaal timed automata model.

*1) Code Generation Algorithm:* The code generator takes as input the XML timed automata representation to produce executable code. In the generated program, the controller automata are encoded as four correlated look-up tables (all transitions in priority order $Trans[]$, current active transitions in priority order $ActiveTrans[]$, synchronization transitions $SyncTrans[]$, out transitions from location $OutTrans[]$) and two functions (guard-function $Guard()$, assign-function $Assign()$). The second table of current active transitions $ActiveTrans[]$ is dynamic and used to hold the set of currently active edges. The execution of those lookup tables are incorporated in Algorithm 1.

The procedure in Algorithm 1 is executed by the controller thread whenever an event (such as timeout or arrival of an external event) has occurred. Initially, the list of active

edges consists of all edges leaving the initial locations. The procedure scans $ActiveTrans[]$ in priority order and evaluates the corresponding guards with $Guard(Trans)$. If a guard is found to be satisfied $Guard(Trans) == True$, there are two cases. (1) The **if** branch in line 3 is for no synchronization $Sync(Trans) == False$. The assignment is performed and the information in the table of locations is used to update the list of active edges $ActiveTrans[]$. (2) The **else** branch in line 9 is for synchronization. The information in the table of synchronization $SyncTrans[]$ is used to find an active edge belonging to another control automata with complementary action label $SyncPare(Trans)$. If such an edge is found and the guard is true, the compound transition is performed, where the assignments of the two edges are performed, and the $ActiveTrans[]$ is updated. Details about the correctness of the algorithm can be referred to the paper [3].

**Algorithm 1** Execution for the Encoded Automata

```
1: for (each Trans ∈ ActiveTrans[ ]) do
2:   if (Guard(Trans) == True) then
3:     if (Sync(Trans) == False) then
4:       Assign(Trans);
5:       Add(Trans.Dest, OutTrans, ActiveTrans[ ]);
6:       Delete(Trans);
7:       Sort(ActiveTrans[ ], Priority);
8:     else
9:       if (Guard(SyncPare(Trans) == True)) then
10:        Assign(Trans);
11:        Assign(SyncPare(Trans));
12:        Add(SyncPare(Trans).Dest, OutTrans,
           ActiveTrans[ ]);
13:        Add(Trans.Dest, OutTrans, ActiveTrans[ ]);
14:        Delete(Trans);
15:        Delete(SyncPare(Trans));
16:        Sort(ActiveTrans[ ], Priority);
17:      end if
18:    end if
19:  end if
20: end for
```

*2) Automata Model Isolation:* Before applying the code generation algorithm, we need to do some isolations on the model. The isolation is needed because we construct and initialize the timed automata template for two or more MVBCs for comprehensive verification, and now need the timed automata of a single MVBC for code synthesis. The single MVBC should receive and send packets onto the physical bus for communication with other MVBCs, and we use the synchronous channel of Uppaal timed automata to simulate the communication for the packets of sending and receiving events.

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, the synchronization channel rcv_connect_req? can be replaced by a declaration of boolean variable rcv_connect_req. At the same time, 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 are asynchronous. Besides, because the generated code is tightly coupled as presented in algorithm 1, the manual separation code of the first way 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 can formalize the implementation verifiable requirements as presented in section IV-C. We input the formalized properties 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, with the cost of increasing the storage overhead of the system.

## V. Experiment Results

In order to evaluate the proposed safety assured design approach, we apply it to the real design of MVBC with the cooperation of CRRC. The formalized 92 critical model verifiable safety requirements are verified on the timed automata and the 29 critical implementation verifiable safety requirements are verified during the runtime execution. Furthermore, we compare the proposed approach with BeagleBone [25] (an available design framework for MVBC based on Simulink) and GalsBlock [15] (a framework for general system design). We mainly compared them in the bug detection efficiency and the resource consumption of generated code.

During the requirement verification process, 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 (P1, P2, P5, P6, P7 of Table 1,

TABLE II
RESOURCE UTILIZATION OF C COMPILATION FOR MVBC, AND THE VERIFICATION EFFICIENCY

| C Compilation | Safety-Assured | BeagleBone(Simulink) | GalsBlock |
|---|---|---|---|
| Binary File Size KB | 302 | 683 | 489 |
| Bug in IEC Standard Detected | 6(verification) | 1(Simulink Design Verifier) | 3 |
| Injected Division by Zero Detected | 10/10(verification) | 4/10(Simulink Design Verifier) | 8/10 |

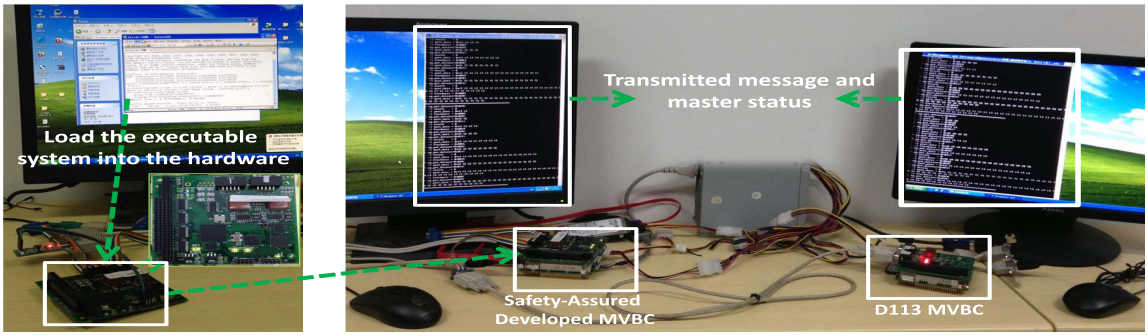



Fig. 9. Real platform communication between the safety-assured MVBC and the worldwide used D113.

P8 of Listing 1) are violated because of the error of the control logic described in the standard. 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.

Let us see the corresponding debugging and correction process for the violation of property P2. Through the counter example of Uppaal, the violation is tracked to encoded C statement *{expected := NK_seq_nr; send_not_yet := (expected + credit) % 8;}* contained in the action of transition for data retransmission, which is also located in table 33 of the standard IEC 61375-1. 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;})*. The physical problem corresponding with this bug occurs when the second packet is lost. In that case, receiver on the MVB controller (consumer) will also reply with a number asking for retransmission of the lost packet 2. However, the MVBC (producer) will mistakenly retransmit packet 3.

In a similar way, we can locate the violation of Property P1, which is due to the statement of table 33 of the standard IEC 61375-1. The guard statement should be changed to *(expected ≤ NK_number ≤ send_not_yet)* Furthermore, the violation of P5, P6 and P7 can be traced back to the handling logic of timeout event and master collision event described in SDL model of Figure 7. Actually, P5 is the combination scenario of P6 and P7. For P6, we propose to add a handshake before standby master changes to regular master because of the timeout. For P7, when a collision happens, we propose to withdraw the responsibility of MVB master controller that is the slave in the previous cycle. For P8, during the runtime verification after code synthesis and integration, the runtime monitor reports an error because of TimeoutReply event. The time is 6.4us, which is greater than 4us. We solve the problem by changing the time-consuming GPIO operation of notifying the arrival of the master frame to direct hardware interrupt, and change the arbitration mechanism for reading access of register pool for slave master data. So the slave MVB controller can response more quickly. After revision, both the model level verification and the runtime verification reports no violation. Besides, these bugs and ambiguousness have already been submitted and would be revised in the new version of IEC standard 61375-1. For the BeagleBone verification based on Simulink Design Verifier, only one bug is detected through verification on the constructed Stateflow model, and three bugs are detected through verification on the constructed GalsBlock model. Furthermore, we test the verification ability of Uppaal, Simulink Design Verifier and GalsBlock by injecting 10 division-by-zero errors into the timed automata, Stateflow and Galsblock model respectively. As presented in table II, the false negative rate of the proposed approach is zero, while the rate of Simulink Design Verifier and GalsBlock is 60% and 20%.

The generated code according to the revised model and the integrated executable system with the eCos (Embedded Configurable Operation System) is synthesized. The compiled binary file is 302 kb, 683 kb and 489 kb for the code generated by the proposed approach, BeagleBone and GalsBlock respectively. The difference is mainly derived from the fact that BeagleBone and GalsBlock generate many extra configuration files and introduces many libraries for scalability. Then, the synthesized binary files for the integrated C code can be loaded into the ARM_SRAM, and running on ARM7-STM32F407IGH6 processor. 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

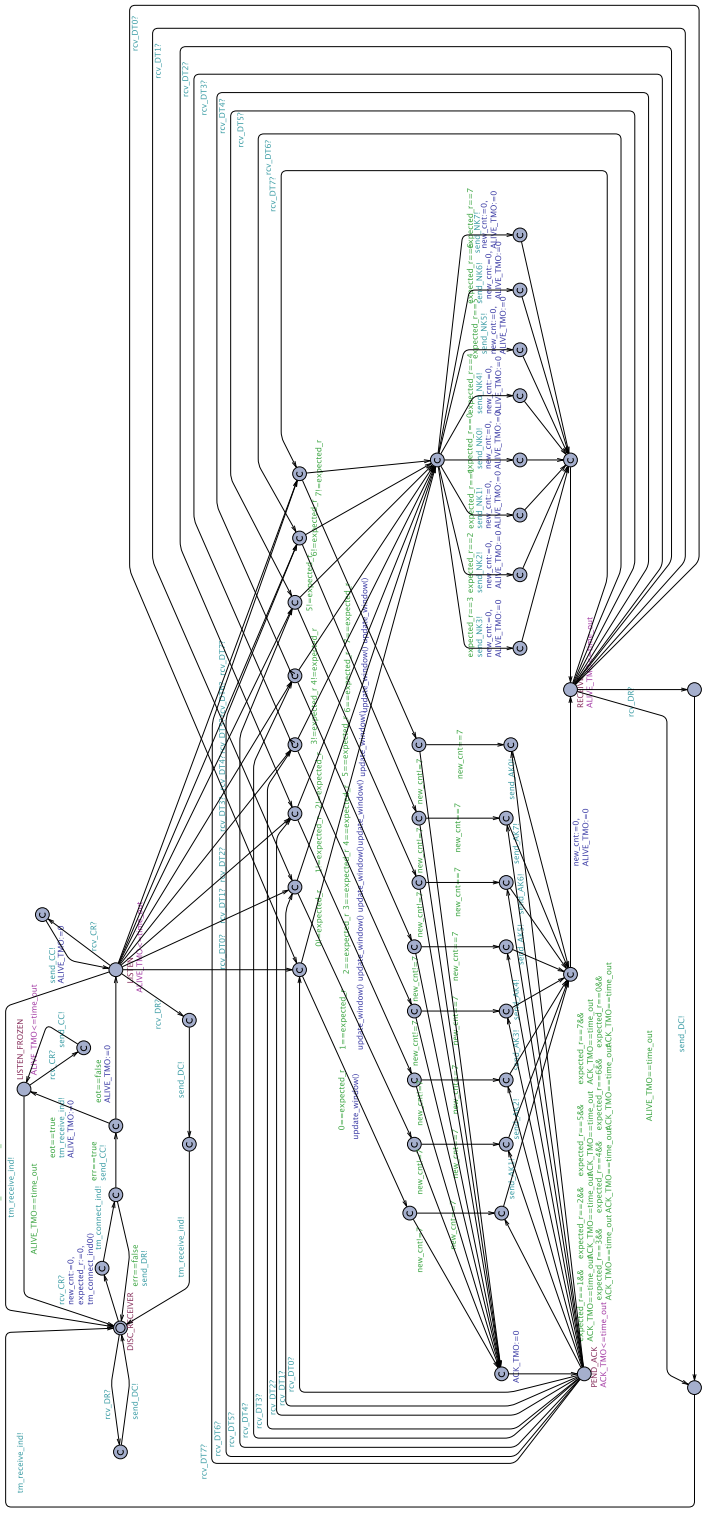


Fig. 10. Constructed timed automata for the receiver module of the MVBC [Enlarged For Review Purpose].

communication. As presented in Figure. 13, the MVBC is embedded into the industrial computer to receive instructions from a keyboard. We use the application running on the industrial computer to monitor communication, and read the message data from memory. It shows that the communications and the master transfer logic are not only executed correctly, but also satisfied with the real-time constraints, as defined in the standard IEC 61375-2. The product-level MVBC has been equipped on real test train and deployed in several subways with accompanied components by CRRC.

*Validity Discussion:* Currently, there are four limitations to the proposed safety assured approach. The first threat concerns the modeling ability of timed automata and the expression ability of the TLTL formula.

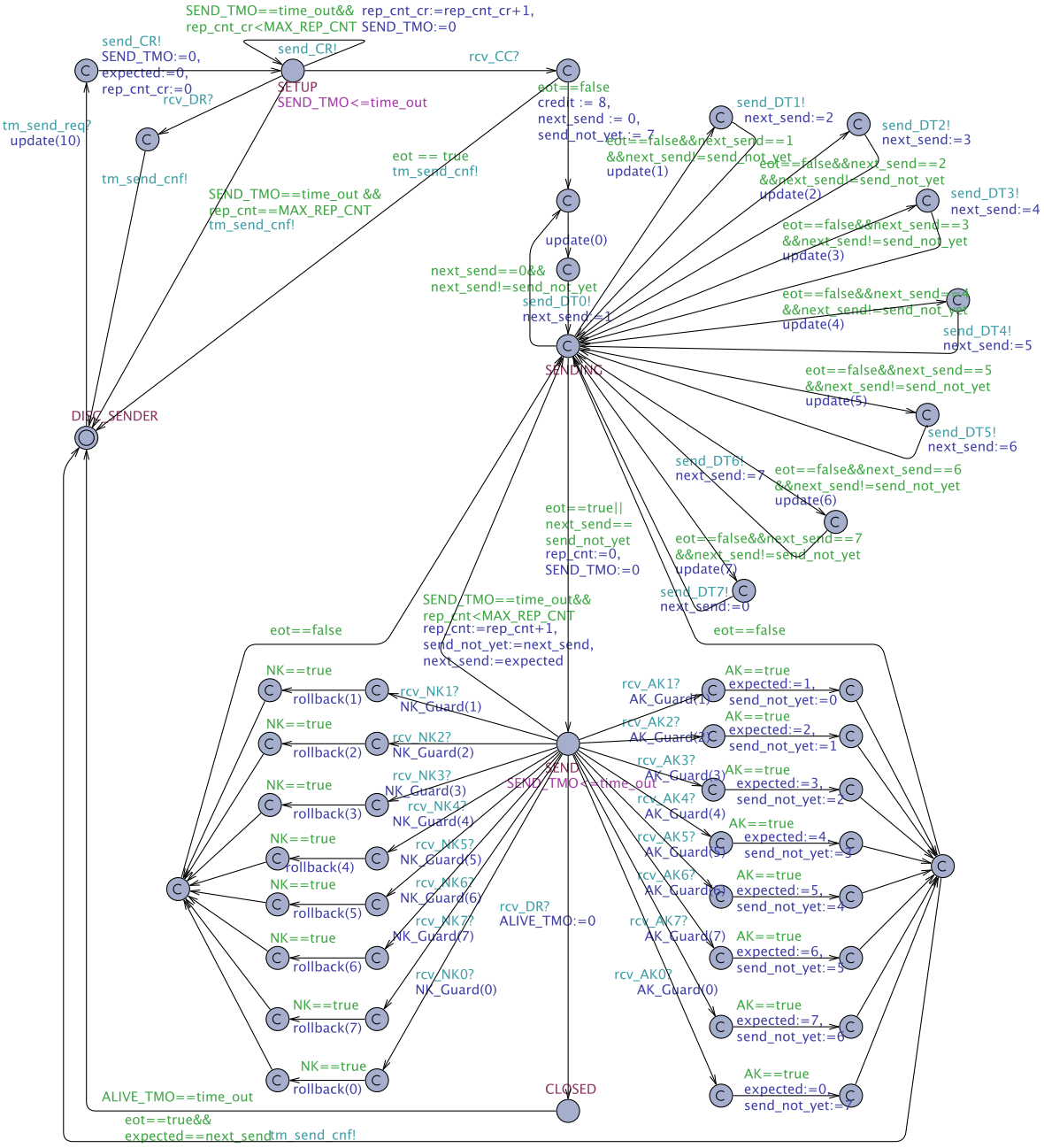



Fig. 11. Constructed timed automata for the sender module of the MVBC [Enlarged For Review Purpose].

There are several parameters and properties that are not easy to be captured in the abstract model level, such as the time delay of arriving packets through the vehicle bus. We use some estimated interval to set the value in the model, and make use of runtime verification to capture those properties that can be not formalized as TLTL formula online. The second threat concerns the verification ability of Uppaal. For complex timed automata model and property specification, it will easily lead to the well-known state explosion problem. The state explosion problem exists in most model checking tools, and can be partially solved by techniques such as symbolic algorithms and abstraction. Within our work, we can get the verification result within 12-60 seconds for all verified requirements. We choose timed automata and Uppaal because they have been well applied to the modeling and verification of embedded system, and the supporting code generation tool Times supports automatic implementation. If we choose other tools such as Spin which use Promela as its modeling language, we can do some verification but there is no automatic code generation support. The third threat concerns the quality of the code generated by Times and the glue code. Because Times has been used and tested in several real projects, we believe the generated code is correct. If not, we can apply several existing code analysis tools to enhance the quality. For example, Intel inspector, Coverity, Blast, and CMBC are widely-used C code analysis tools [8], [18], [22], [24] and can be

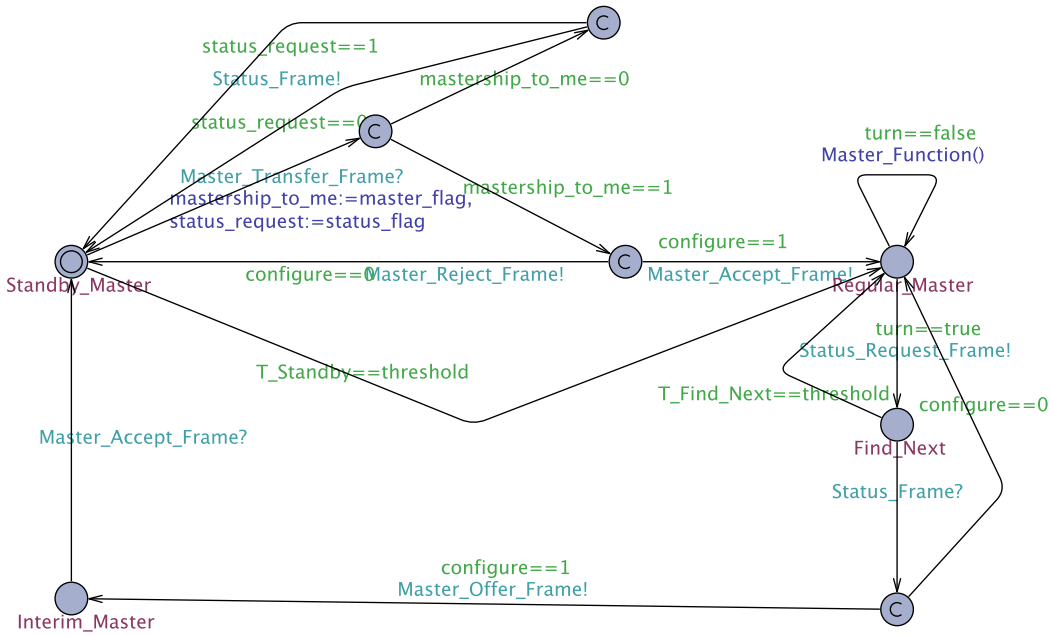



Fig. 12. Refined timed automata for the master transfer of MVBC [Enlarged For Review Purpose].

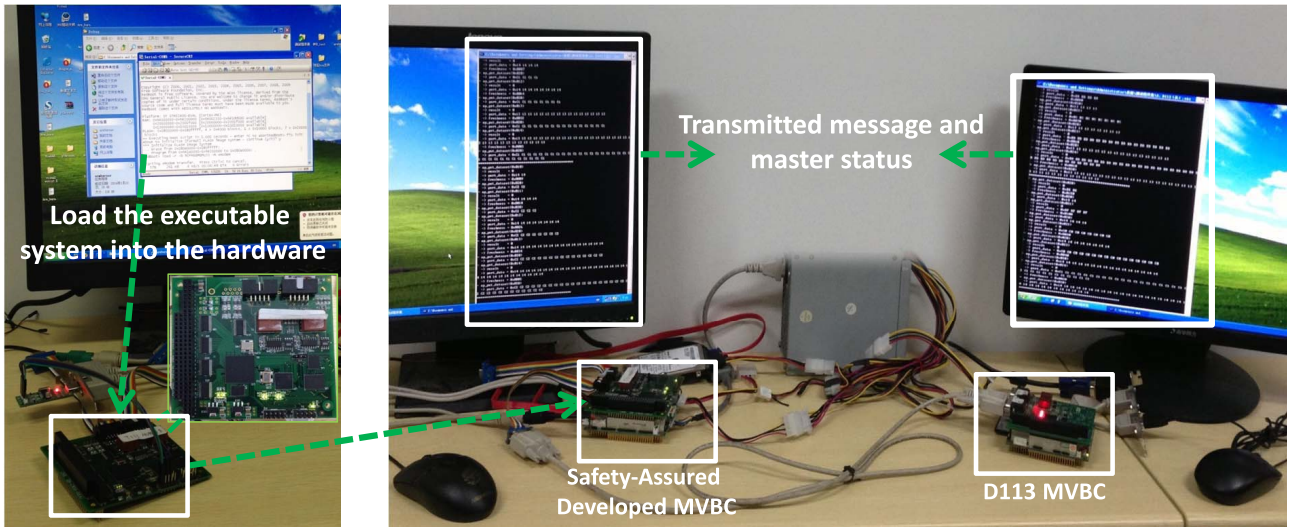



Fig. 13. Real platform communication between the safety-assured MVBC and the worldwide used D113 [Enlarged For Review Purpose].

integrated into our safety-assured approach to enhance our work.

## VI. 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. Although we address the safety assured design of MVBC in this paper, it is reasonable to apply the proposed approach to other similar system design. Our future work is to extend the code generation mechanism of timed automata, and plans to develop the tool to allow the code generation for the selected part of the network of timed automata. Another direction is to ensure the security requirement based on the formal method.

## References

[1] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking for real-time systems," in *Proc. 5th Annu. IEEE Symp. Logic Comput. Sci. (LICS)*, Jun. 1990, pp. 414–425.

[2] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[3] T. Amnell, "Code synthesis for timed automata," Dept. Comput. Sci., Uppsala Univ., Uppsala, Sweden, White Paper up-te-11102, 2003.

[4] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "TIMES—A tool for modelling and implementation of embedded systems," in *Proc. Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2002, pp. 460–464.

[5] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An integrated electronic system design environment," *Computer*, vol. 36, no. 4, pp. 45–52, Apr. 2003.

[6] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal," in *Formal Methods for the Design of Real-Time Systems*. IL, USA: Springer, 2004, pp. 200–236.

[7] G. Berry, "SCADE: Synchronous design and validation of embedded control software," in *Proc. Workshop Next Generat. Design Verification Methodol. Distrib. Embedded Control Syst.*, 2007, pp. 19–33.

[8] A. Bessey *et al.*, "A few billion lines of code later: Using static analysis to find bugs in the real world," *Commun. ACM*, vol. 53, no. 2, pp. 66–75, 2010.

[9] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *Int. J. Comput. Simul.*, vol. 4, pp. 155–182, Aug. 1994.

[10] *Train Communication Network*, IEC Standard IEC 61375-1, Int. Electrotech. Commission, Geneva, Switzerland, 2011.

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

[12] K. Havelund, "Runtime verification of c programs," in *Testing of Software and Communicating Systems*. Springer, 2008, pp. 7–22.

[13] X. Iturbe, A. Zuloaga, J. Jiménez, J. Lázaro, and J. L. Martín, "A novel SoC architecture for a MVB slave node," in *Proc. 34th Annu. Conf. IEEE Ind. Electron. (IECON)*, 2008, pp. 1455–1460.

[14] F. Jahanian and A. K.-L. Mok, "Safety analysis of timing properties in real-time systems," *IEEE Trans. Softw. Eng.*, vol. SE-12, no. 9, pp. 890–904, Sep. 1986.

[15] Y. Jiang *et al.*, "Design of mixed synchronous/asynchronous systems with multiple clocks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 8, pp. 2220–2232, Aug. 2014.

[16] Y. Jiang *et al.*, "Design and optimization of multiclocked embedded systems using formal techniques," *IEEE Trans. Ind. Electron.*, vol. 62, no. 2, pp. 1270–1278, Feb. 2014.

[17] J. Jiménez, J. Arias, J. Andreu, C. Cuadrado, and I. Kortabarria, "Design methodology for multifunction vehicle bus devices," in *Proc. 5th WSEAS Int. Conf. Syst. Sci. Simulation Eng.*, 2006, pp. 352–357.

[18] D. Kroening and M. Tautschnig, "CBMC—C bounded model checker," in *Proc. Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2014, pp. 389–391.

[19] J.-H. Lee, J.-G. Hwang, and G.-T. Park, "Performance evaluation and verification of communication protocol for railway signaling systems," *Comput. Standards Inter.*, vol. 27, no. 3, pp. 207–219, 2005.

[20] N. G. Leveson and J. L. Stolzy, "Safety analysis using Petri nets," *IEEE Trans. Softw. Eng.*, vol. SE-13, no. 3, pp. 386–397, Mar. 1987.

[21] Z. Li, F. Yang, and Q. Xing, "Design of multifunction vehicle bus controller," in *Computer and Computing Technologies in Agriculture IV*. Shenzhen, China: Springer, 2010, pp. 177–183.

[22] P. Louridas, "Static code analysis," *IEEE Softw.*, vol. 23, no. 4, pp. 58–61, Jul. 2006.

[23] *User Manual for Stateflow*, Simulink Inc., Natick, MA, USA, 2010.

[24] P. Petersen, "Intel parallel inspector," in *Encyclopedia of Parallel Computing*. CA, USA: Springer, 2011, pp. 944–949.

[25] R. Aarthipriya and S. Chitrapreyanka, "FPGA implementation of multifunction vehicle bus controller with class 2 interface and verification using beaglebone black," *Int. J. Sci. Eng. Res.*, vol. 3, no. 5, pp. 3221–3225, 2015.

[26] S. G. Shon and H. J. Byun, "Design and implementation of embedded MVB-ethernet interface," in *Proc. ACM Symp. Res. Appl. Comput.*, 2011, pp. 93–96.

[27] H. Song *et al.*, "Safety-assured formal model-driven design of the multifunction vehicle bus controller," in *Proc. 21st Int. Symp. Formal Methods*, 2016, pp. 757–763.

[28] W. Sun, F. R. Yu, T. Tang, and B. Bu, "Energy-efficient communication-based train control systems with packet delay and loss," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 9, p. 452–468, Feb. 2016.

[29] F. Tan, Y. Wang, Q. Wang, L. Bu, and N. Suri, "A lease based hybrid design pattern for proper-temporal-embedding of wireless CPS interlocking," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 10, pp. 2630–2642, Oct. 2015.

[30] F. Tan, Y. Wang, Q. Wang, L. Bu, R. Zheng, and N. Suri, "Guaranteeing proper-temporal-embedding safety rules in wireless cps: A hybrid formal modeling approach," in *Proc. 43rd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2013, pp. 1–12.

[31] D. Wang, F. He, Y. Deng, C. Su, M. Gu, and J. Sun, "Deadlock detection in FPGA design: A practical approach," *Tsinghua Sci. Technol.*, vol. 20, no. 2, pp. 212–218, 2015.

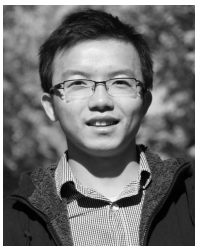
[32] Y. Wang, Y. Song, H. Gao, and F. L. Lewis, "Distributed fault-tolerant control of virtually and physically interconnected systems with application to high-speed trains under traction/braking failures," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 2, p. 535–545, Feb. 2015.

[33] WIKIPEDIA. (2015). *Philadelphia Train Derailment*. [Online]. Available: https://en.wikipedia.org/wiki/2015_Philadelphia_train_derailment

[34] W. Wu and T. Kelly, "Safety tactics for software architecture design," in *Proc. 28th Annu. Int. Comput. Softw. Appl. Conf. (COMPSAC)*, Sep. 2004, pp. 368–375.

[35] F. Yunxiao, L. Zhi, P. Jingjing, L. Hongyu, and S. Jiang, "Applying systems thinking approach to accident analysis in China: Case study of '7.23' Yong-Tai-Wen high-speed train accident," *Safety Sci.*, vol. 76, pp. 190–201, Jul. 2015.

[36] L. Zhao, B. Cai, J. Xu, and Y. Ran, "Study of the track–train continuous information transmission process in a high-speed railway," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 1, pp. 112–121, Feb. 2014.

**Yu Jiang** received the B.S. degree in software engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2010 and the Ph.D. degree in computer science from Tsinghua University, Beijing, in 2015. He is currently an Assistant Professor with the School of Software, Tsinghua university. His current research interests include domain specific modeling, formal computation model, formal verification and their applications in embedded systems.

**Han Liu** received the B.S. degree in software engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2012. He is currently pursuing the Ph.D. degree in software engineering with Tsinghua University, Beijing. His current research interests include domain specific modeling, formal verification and their applications in embedded systems.

**Houbing Song** received the M.S. degree in civil engineering from University of Texas at El Paso, El Paso, TX, USA, in 2006, and the Ph.D. degree in electrical engineering from University of Virginia, Charlottesville, VA, USA, in 2012. In 2012, he joined the Department of Electrical and Computer Engineering, West Virginia University, Morgantown, WV, USA, where he is currently an Assistant Professor. His current research interests include cyber-physical systems, intelligent transportation systems, wireless communications and networking, and optical communications and networking.

**Hui Kong** received the B.S. degree in mathematics from Wuhan University, Wuhan, China, in 2001, and the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2014. He is currently a Post-Doctoral Researcher with Institute of Science and Technology Austria, Austria. His current research interests include domain specific modeling, formal computation model, formal verification and their applications in embedded systems, safety analysis, and the assurance of cyber-physical system and hybrid system.

**Rui Wang** received the B.S. degree in computer science from Xi'an Jiaotong University, Xi'an, China, in 2004 and the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2011. She is currently a Lecturer with the College of Information Engineering, Capital Normal University, China. Her current research interests include formal verification and their applications in embedded systems.

**Yong Guan** received the Ph.D. degree from the College of Mechanical Electronic and Information Engineering, China University of Mining and Technology, China, in 2004. He is currently a Professor with Capital Normal University. His research interests include the formal verification of embedded system design. He is a member of the Chinese Institute of Electronics Embedded Expert Committee.

**Lui Sha** received the Ph.D. degree from Carnegie Mellon University, Pittsburgh, PA, USA, in 1985. He is currently a Donald B. Gillies Chair Professor of computer science with University of Illinois at Urbana–Champaign. His work on real-time computing is supported by most of the open standards in real-time computing and has been cited as a key element to the success of many national high-technology projects, including GPS upgrade, the Mars Pathfinder, and the International Space Station.