# Formal Design of Multi-Function Vehicle Bus Controller

Yu Jiang, Mingzhe Wang, Zhuo Su, Yixiao Yang, and Huihui Wang, *Senior Member, IEEE*

***Abstract*— Data of the train communication network(TCN) is becoming more complicated, which results in higher requirements of the data processing unit-the multifunction vehicle bus controller (MVBC) connected within the TCN. Developing an MVBC is challenging because of the integrated hardware-software solutions to support reactions in real time and dynamic environment. Hence, there is an urgent need for a rigorous design framework to facilitate the development of MVBC. In this paper, we propose a design framework TooMVBC to generate executable MVBC code from formal verified computation model. TooMVBC uses formal computation model MVBChart to capture the specification of the MVBC at high level. First, primitive syntax of MVBChart is designed to model MVBC features (*e.g.* hierarchy structure, data flow of the encoder, the control logic of communication protocol), and semantics of MVBChart is formalized for simulation and verification. Then, semantics-preserving code generation algorithms are designed to generate VHDL code for partitioned hardware implementations and C code for partitioned software implementations from verified MVBChart model. The generated code can be loaded into the proposed flexible MVBC hardware architecture directly. Finally, supporting graphical model editor, simulator, verification translator, partitioning and code generator are implemented and seamlessly integrated into TooMVBC. When we apply TooMVBC to design MVBC with the highest class 5 according to the description of the standard IEC 61375, several critical ambiguousness or bugs in the standard are detected during formal verification of the constructed system model.**

***Index Terms*— Model-driven design, formal computation model, vehicle bus controller (VBC).**

## I. Introduction

THE train communication network (TCN) was standardized in the international standard IEC 61375 [4] and adopted in most modern train control systems. For example, the data processing unit MVBC D113 developed by Duagon company complies to this standard and supports leading train communication and control for the world market.

Manuscript received April 22, 2020; revised September 12, 2020; accepted October 14, 2020. Date of publication May 20, 2021; date of current version June 2, 2021. This work was supported in part by the National Key Research and Development Project under Grant 2019YFB1706200 and Grant 2018YFB1703404, in part by the NSFC Program Grant 62022046, Grant U1911401, and Grant 61802223, and in part by the Huawei-Tsinghua Trustworthy Research Project Grant 20192000794. The Associate Editor for this article was Z. Lv. *(Corresponding author: Zhuo Su.)*

Yu Jiang, Mingzhe Wang, Zhuo Su, and Yixiao Yang are with the Key Laboratory for Information System Security, Ministry of Education (KLISS), Beijing National Research Center for Information Science and Technology (BNRist), School of Software Engineering, Tsinghua University, Beijing 100084, China (e-mail: jiangyu198964@126.com; suz18@mails.tsinghua.edu.cn).

Huihui Wang is with the Department of Engineering, Jacksonville University, Jacksonville, FL 32211 USA.

Digital Object Identifier 10.1109/TITS.2021.3078372

As an important part of the TCN, the multifunction vehicle bus controller (MVBC) is responsible for the data communication of all on-board equipment within a vehicle. With the higher requirement of the modern high-speed train, the functional complexity of the MVBC has increased from the class 1 to class 5 to support the increasing requirements, which results in more difficulties to ensure the correctness of the MVBC. Once the MVBC is deployed after system test, the train will have to work almost fully autonomously without further interactions for a long period of time. This means even small mistakes may have severe consequences, even for train crashes and lives lose. Although there are many existing works introducing how to design and implement an MVBC [11], [12], [16], [18], [20], [22], most of them focus on the manual functional implementation and do not pay attention to safety assurance under dynamic physical environment.

Besides, from the perspective of industrial practice, mixed hardware-software solutions are gaining increasing popularity in real-world reactive applications of safety-critical domains such as aerospace, so as transportation. The most widely used MVBC D113 adopts this implementation manner, where some components are implemented in hardware to ensure determinacy and stability, while others without strict timing and performance constraints are usually implemented in software to save computation resource. This implementation pattern leads to an increasing heterogeneity, which brings more difficulties in designing and validating MVBC.

In this paper, we propose a formal design framework named TooMVBC with a formal basis, MVBChart computation model, to address the safety design of MVBC and guarantee a high reliability of the safety-critical train applications. Overall structure of the design environment is presented in Fig 1, and details are briefly outlined below.

The theoretical basis of TooMVBC is a formal model of computation referred to as MVBChart. In this co-design model, hierarchical structure and data flow of MVBC such as encoder and decoder, and dynamic environment are captured by compound and atom blocks with data port connections. Functional requirements and control oriented communication protocol of each component are expressed by parallel automata contained in atom block. To strengthen the analytical ability of the computation model MVBChart, we formalize rules for interpreting the model into a labelled transition system (LTS), which can be directly analyzed and verified by a formal verifier, Beagle [5]. Furthermore, we propose resource-saving code generation algorithms to generate VHDL code for blocks assigned to hardware implementation, and C code for assigned

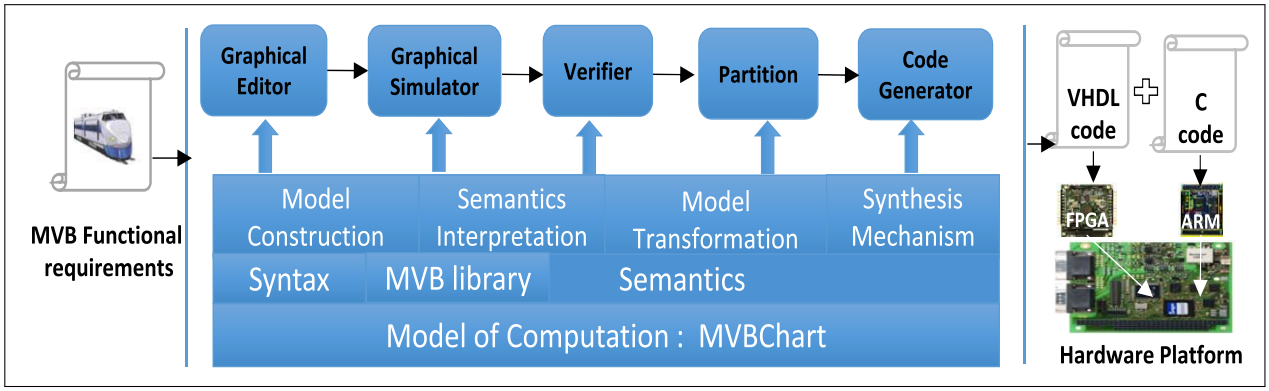



Fig. 1. Overview of the TooMVBC design framework.

software implementation, after customized automatically partitioning.

Based on the integrated design framework TooMVBC, we are able to shorten the development cycle to build and validate a MVBChart model at a high level, compared to the traditional practice which involves implementing the sketch-like design of MVBC in low-level programming languages as C and VHDL. In addition, the graphical model validation through simulation and enhanced formal verification opens a user-friendly interface for us to uncover design defects at the early development stage. Once we have verified safety-critical properties over the system model and iteratively improved the design details, executable code can be directly generated in C(software) and VHDL(hardware) after partitioning.

Furthermore, when we apply TooMVBC in the design of MVBC standardized by IEC 61375 [4], two critical bugs in the standard are detected and confirmed via modeling and validation with formal verification. Another significant strength is a better performance to manage the size of synthesized implementation than other VHDL and C code generators in the literature. The automatically co-synthesized implementation based on the bug-free model passes all physical tests, and has been deployed in real-world subways.

## II. Related Work

During the past decades, many scientists have paid many efforts to the design and implementation of the MVBC [7], [12], [16], [18], [20]. In [7], they propose to use materialization of slave nodes for MVBC in a single chip by using reconfigurable logic. In [12], [18], 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 on safety assurance under dynamic physical environment. Although many researchers have defined several types of formal semantics for Stateflow, and developed many specialized tools for translating subsets of model to pushdown automata, Lustre, PAT, and timed automata, which can be verified through the corresponding supporting tools [9], [21], most of them performs well within their own domain while abstracting some domain unrelated modeling features. For example, in timed automata based translation, they translate the common features such as timed operator, while the embedded M function code are out of their considerations. Besides, there are also some works about verifying the real time communication protocol of TCN [8], [11], [13]–[15], [24], they describe some formal methods to verify safety properties of the communication protocol used in train control system. But they do not deal with the co-design issue of hardware-software controlled behaviors of MVBC and they do not cope with the implementation issue neither, they focus on the logic correctness of train communication protocol only. Furthermore, all works above are incapable of handling the analysis or design of complex software implementation of message communication of class 5. 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, where there are still some currently unimportant bugs.

Except for the above work for MVBC, there are large amounts of work and various toolkits in the literature supporting the design of general synchronous reactive systems [17], [23]. For example, SCADE [3] uses SSM as the formal basis, and has been successfully applied in a variety of applications with 20,0000 US dollars for a single license. While mainly focusing on embedded software, SCADE currently has little support for the synthesis of hardware.

For MVBChart, we mainly eliminate the hieratical state, the composite state, during action, entry action, exit action, junction operator, M function of Stateflow, the weak abortion, strong abortion, signal communication, the composite state of SCADE. In the synchronous system design domain, those advanced features can be interpreted by the basic data port communication port, parallel automata, embedded C function, atom block and compound block of MVBChart.

## III. Theoretical Basis

In this section, we introduce theoretical basis of TooMVBC, including the construction of MVBChart computation model, semantics interpretation, formal semantics for verification, and co-synthesis mechanism.

### A. MVBChart Parallel Computation Model

In MVBChart computation model, an MVBC is specified as a combination of compound and atom blocks communicating

through point-to-point channels. The compound block does not directly conduct computation. Instead, it presents the hierarchical decomposition of general component (encoder, decoder, counter, master frame, etc.) structure and data flow path among them. The atom block is refined as parallel automata to capture the behaviors and detail computation such as the logic to generate a master frame or the computation to decode a communication frame. The complete rules and general domain features can be referred to the manual, based on which, we present its underlying formal tuple definitions for simulation and verification:

*1) Automaton:* An automaton $C_i$ is defined as a tuple $\langle L, l_0, V, G, A, E, P\rangle$, where $L$ is a set of locations $\{l_i | i \in [0, n]\}$, $l_0$ is the initial location $\{l_0 | l_0 \in L\}$, $V$ is a set of parameters $\{v_i | i \in [0, n]\}$ inherited from local variables, input data ports, and output data ports of the parent atom block, $A$ is a set of actions $\{a_i | i \in [0, n]\}$, $G(V)$ is a set of guards $\{g_i(V) | i \in [0, n]\}$,[1] $E$ is a set of transition edges $\{e_i \subseteq L \times G(V) \times A \times L | i \in [0, n]\}$ between two locations with the attached action and guard, $P$ is a set of priority valuation function $\{p_i \subseteq e_i < e_j | i, j \in [0, n], e_i \; and \; e_j \; start \; from \; the \; same \; source \; state\}$ defined on the edges that may take simultaneously.

*2) Atom Block:* An atom block $ab_i$ is defined as a tuple $\langle I, O, V, C\rangle$, where $I$ is a set of input data ports $\{I_i | i \in [0, n]\}$, $O$ is a set of output data ports $\{O_i | i \in [0, n]\}$, $V$ is a set of local variables $\{v_i | i \in [0, n]\}$, and $C$ is a set of automata $\{C_i | i \in [1, n]\}$.

*3) Compound Block:* A compound block $cb_i$ is defined as a tuple $\langle I, O, B, W, F\rangle$, where $I$ is a set of input data ports $\{I_i | i \in [0, n]\}$, $O$ is a set of output data ports $\{O_i | i \in [0, n]\}$, $B$ is a set of sub-atom blocks $\{ab_i | i \in [0, n]\}$ and sub-compound blocks $\{cb_j | j \in [0, n]\}$, $W$ is a set of point-to-point channel connections $\{w_i \subseteq port \times port | port \in \{M.I \cup M.O \cup B.I \cup B.O\}$ among the data ports within this compound block, and $F$ is a set of expressions $\{f_i | i \in [0, n]\}$ attached on the connection.

*4) MVBChart:* A MVBChart computation model $M$ is defined as a tuple $\langle CB, T\rangle$, where $CB$ is the outermost compound block, and $T$ is the synchronous trigger for computation. In fact, MVBChart model can be represented by the outermost compound block.

Based on the tuple definitions, we have built 36 general modules *(master frame, slave frame, decoder, encoder, pool ram control, counter etc.)* that can be reconfigured to model different types of MVBC.

**Algorithm 1:** Computation of Atom Block, Which Is Further Refined to Computation of Automata

**Input:** all atom blocks $ab_i$ contained in MVBChart parallel computation model $M$.
**Output:** The configuration $\mathbf{C}_i$ of each atom block.

```
// Compute the configuration for each
   atom block ab_i in parallel.
foreach atom block ab_i ∈ MVBChart M do
    // Read the initial configuration of
       ab_i: the input data ports, local
       variables, and active states.
    p_i_temp[] ← p_i[];
    var_i_temp[] ← var_i[];
    ActiveState_i_temp[] ← ActiveState_i[];
    // Do parallel computation on each
       automaton contained in ab_i based on
       the initial configuration.
    foreach automaton C_i^j ∈ the atom block ab_i do
        // Select all transitions starting
           from the current active state
           ActiveState_i_temp[].
        Trans_i^j_temp[] ←
         Get_Trans(C_i^j.E, ActiveState_i_temp[j]);
        // Sort the selected transitions
           according to the priority.
        Trans_i^j_temp[] ← Sort(Trans_i^j_temp[], C_i^j.P);
        // Trigger the transition with the
           highest priority
        foreach transition ∈ Trans_i^j_temp[] do
            if Guard of Trans_i^j_temp[k] is true then
                if Action a_i^j is not conflict then
                    (p_i_temp[], var_i_temp[],
                     ActiveState_i_temp[])
                    ←
                    Action(Trans_i^j_temp[k], p_i_temp[],
                     var_i_temp[]);
                    break ;
                else
                    Conflict Error Wraning;
    // Return the configuration of the
       atom block ab_i: Output data ports,
       local variables and active states
    p_j[] ← p_j_temp[];
    var_i[] ← var_i_temp[];
    ActiveState_i[] ← ActiveState_i_temp[];
```

## B. Model Interpretation for Simulation

According to the definition of IEC standard 61375, MVBC is a typical synchronous reactive system, which is triggered by a periodic clock. In context of synchronous systems, the system reaction step, mainly involves in three phases: import inputs, compute changes and export outputs. In the semantics of MVBChart, it should adhere to the reactions of real system and calculations of MVBChart model are based on initial status at the start of current step. This semantics liberate engineers from the puzzle of causality error during the model construction. Also, the synthesized VHDL code with this semantics solves the meta-stability problem in the same way as Moore machine.

Given those basic tuple definitions in the previous section, basic step computation of MVBChart can be interpreted as

[1] $a_i$ and $g_i(v)$ are statements supported in C. The partially supported part can be referred to the cited manual.

the configuration computation that the next active state, values of output data ports and local variables. As presented in Algorithm. 1, the step computation semantics is as follows:

1. First, when a new computation step starts, the model should import inputs from the environment through input data ports $\{CB.I\}$ of the outermost model $M$. It will read their values, and pass the values through connections until endpoint arrives, which is input data ports $\{ab_i.I\}$ of the innermost atom blocks.
2. Then, all atom blocks $ab_i$ will read updated value from input data ports and local variables, as presented in statements 2-4 of algorithm 1. The automata $\{C_i^j\}$ contained in each atom block will determines the next state, values of output data ports and local variables in parallel, as presented in statements 5-16. Because within an automaton, there may be several transitions starting from a source state with their guard satisfied, we need to choose transition with the highest priority, as presented in statements 6-7. Also, because all automata $\{C_i^j\}$ are executed in parallel and independent of each other, and there might be conflicts on read-write operations. Each local variable and output data port can only be written by one automaton. If there are two enabled transitions $e_m$ and $e_n$, from the automaton $C_i^m$ and $C_i^n$ of atom block $ab_i$ respectively, and their attached actions $a_m$ and $a_n$ attempt to update the same local variable or output data port, a conflict is reported, as presented in statement 10 and 16. After the recursive execution finishes, new configuration will be returned, as presented in the statements 17-19.
3. Finally, at the end of each computation step, returned value of output data ports should be passed through connections to the endpoints, which are input data ports $\{ab_i.I\}$ of another atom block, or output data ports $\{CB.O\}$ of the outermost block $M$. The former is used for the next computation step, and the latter is used for exporting outputs to outside environment.

### C. Formal Semantics for Verification

Formal semantics of MVBChart is determined by its equivalent labeled transition system (LTS) [6], based on which, we can formalize decision problem of MVBChart in the same way as [1], and input it to Beagle for verification directly. An LTS is defined as a tuple $\langle S, s_0, \rightarrow, A\rangle$, where $S$ is a set of states $\{s_0,\ s_1,\ \cdots,\ s_n\}$, $A$ is a set of actions $\{a_0,\ a_1,\ \cdots,\ a_n\}$, $\rightarrow \subseteq S \times A \times S$ is a set of transitions, and $s_0$ is the initial state. For any state $s_i$ and action $a_i$, if there is only one transition $s_i \xrightarrow{a_i} s_{i+1}$ contained in the transition set $\rightarrow \subseteq S \times A \times S$, the LTS is called as $deterministic$. Following description style of LTS in [8], the equivalent LTS can be constructed bottom-up, starting from the parallel automaton $C_i$ to the outermost MVBChart model $M$.

*1) LTS for Automaton:* Let $\langle L, l_0, V,\ A, E, P\rangle$ be an automaton $C_i$ as defined in the section III-A. $U$ is a set of parameter valuation functions $\{u_i \subseteq V \cup T \rightarrow B \cup N | i \in [0, n]\}$ from the parameters to bool or integer, where $T$ is the trigger.

Semantics of the automaton $C_i$ can be defined as a labeled transition system $\langle S, s_0, \rightarrow, A\rangle$, where $S$ is a set of configuration $\{s_i = (l_i, u_i) \subseteq C.L \times U(C.V) | i \in [0, n]\}$, $s_0 = (l_0, u_0)$ is the initial configuration, $A$ is a set of action $\{a_i \subseteq C_i.A \mid i \in [0, n]\}$ and $\rightarrow \subseteq S \times A \times S$ is the transition rule for an automaton reaction step such that:

$$\frac{\{(l, g, a, l') | max\{p(e_i) | g(v) = true\}\} \neq \emptyset \bigwedge g(T) = true}{(l, u) \xrightarrow{a} (l', u')}$$

$$\frac{\{(l, g, a, l') | max\{p(e_i) | g(v) = true\}\} = \emptyset \bigwedge g(T) = true}{(l, u) \xrightarrow{\emptyset} (l, u')}$$

where each automaton is allowed to take the transition with the highest priority (rule 1), or staying in the current state (rule 2). For each execution, the clock $T$ will increase with a minimum $\triangle\ t$ to break the guard $g(T')$, so that the next execution will be enabled on the next period of clock $T$. These rules break the output from the internal input changes through the clock, similar to the rules of Moore machine.

*2) LTS for Atom Block:* Let $\langle I, O, V,\ C\rangle$ be an atom block $ab_i$ as defined in section III-A, consisting of n parallel automata $\bigcup_{i=1}^{n} C_i$. Automata $C_i$ equals $\langle L^i, l_0^i, V^i, G^i, A^i, E^i, P^i\rangle$, where $V^i \subseteq I \cup O \cup V$ is inherited from the atom block. Then, location set $\overline{L}$ for the atom block is defined as $\{\overline{l}_i = (l_i^1, l_i^2, \cdots, l_i^j, \cdots, l_i^n) \subseteq (L^0 \times L^1 \cdots \times L^n) | j \in [0, n], l_i^j \in L^j\}$, and $\overline{l}_0$ equals $(l_0^1, l_0^2, \cdots, l_0^n)$. Action set $\overline{A}$ is defined as $\{\overline{a}_i = (a_i^1, a_i^2, \cdots, a_i^j, \cdots, a_i^n) \subseteq (A^0 \times A^1 \cdots \times A^n) | i \in [0, n], a_i^j \in A^j\}$. Parallel evaluation function $\overline{U}$ is defined as $\{\overline{u}_i = (u_i^1, u_i^2, \cdots, u_i^j, \cdots, u_i^n) \subseteq (U^0 \times U^1 \cdots \times U^n) | j \in [0, n], u_i^j \in U^j\}$, where $U^j$ is the parameter evaluation function $\{u_0^j,\ u_1^j,\ \cdots,\ u_n^j\}$ defined for automaton $C_j$, and $\{u_i^j \subseteq (V^j \cup T \leftarrow B \cup N) | i \in [0, n]\}$. The initialization $\overline{u}_0$ equals $(u_0^1, u_0^2, \cdots, u_0^n)$.

Semantics of the atom block $ab_i$ can be defined as a labeled transition system $\langle S, s_0, \rightarrow, A\rangle$, where $S$ is the set of configuration $\{s_i = (\overline{l}_i, \overline{u}_i) \subseteq \overline{L} \times \overline{U} | i \in [0, n]\}$, $s_0 = (\overline{l_0}, \overline{u_0}) = ((l_0^1, l_0^2, \cdots, l_0^n), (u_0^1, u_0^2, \cdots, u_0^n))$ is the initial configuration, $A$ is a set of action $\{a_i = \overline{a}_i \subseteq \overline{A} \mid i \in [0, n]\}$ and $\rightarrow \subseteq S \times A \times S$ is the transition rule for an atom block reaction such that:

$$\frac{g(T) = true}{(\overline{l}, \overline{u}) \xrightarrow{\overline{a}} (\overline{l'}, \overline{u'})}$$

where the compound transition can be refined to the transition of each automaton $C_j$. $\forall j \in [1, n]$, $(\overline{l}.l^j, \overline{u}.u^j)$ is the configuration of $C_j$, and $T^j$ is inhered from $T$ to ensure synchronization among different automata, as shown at the bottom of the next page.

*3) LTS for Compound Block:* Let $\langle I, O, B, W, F\rangle$ be a compound block $cb_i$ as defined in section III-A, consisting of several sub-atom and sub-compound blocks. The compound block can be re-flattened as $\langle I, O, B', W', F'\rangle$, where $B' = \{\bigcup_{i=1}^{n} ab_i | i \in [0, n]\}$ is a set of all atom blocks in bottom level, $W' = \{w_i' \subseteq port \times port | port \in \{cb_i.I \cup cb_i.O \cup B'.I \cup B'.O\}\}$ is a set of all flatted connections among the data ports of atom block in the bottom level and data ports of the outermost compound block $cb_i$, and $F' = \{f_i' | i \in [0, n]\}$ is a

set of cascaded expressions attached on the flatted connection. Then, location set $\overline{CL}$ for the compound block is defined as $\{\overline{cl}_i = (\overline{l}_i^1, \cdots, \overline{l}_i^j, \cdots, \overline{l}_i^n) \subseteq (\overline{L}^0 \times \overline{L}^1 \times \cdots \times \overline{L}^n) | j \in [0,n], \overline{l}_i^j \in \overline{L}^j, \overline{L}^j = (L^0 \times L^1 \cdots \times L^n)\}$, where $\overline{L}^j$ is the location set for the atom block $ab_i$ of $B'$. The initial location vector $\overline{cl}_0$ equals $(\overline{l}_0^1, \cdots, \overline{l}_0^n)$. The action set $\overline{CA}$ is defined as $\{\overline{ca}_i = (\overline{a_i}^1, \cdots, \overline{a_i}^j, \cdots, \overline{a_i}^n) \subseteq (\overline{A}^0 \times \overline{A}^1 \times \cdots \times \overline{A}^n) | j \in [0,n], \overline{a_i}^j \in \overline{A}^j, \overline{A}^j = (A^0 \times A^1 \times \cdots \times A^n)\}$. The parallel evaluation function vector $\overline{CU}$ is defined as $\{\overline{cu}_i = (\overline{u_i}^1, \cdots, \overline{u_i}^j, \cdots, \overline{u_i}^n) \subseteq (\overline{U}^0 \times \overline{U}^1 \times \cdots \times \overline{U}^n) | j \in [0,n], \overline{u_i}^j \in \overline{U}^j, \overline{U}^j = (U^0 \times U^1 \cdots \times U^n)\}$, where $\overline{U}^j$ is a set of valuation function vector defined on atom block $ab_i$ of $B'$, and the initialization $\overline{cu}_0$ equals $(\overline{u}_0^1, \overline{u}_0^2, \cdots, \overline{u}_0^n)$.

Semantics of the compound block $cb_i$ is defined as a labelled transition system $\langle S, s_0, \rightarrow, A \rangle$, where $S$ is the of configuration $\{s_i = (\overline{cl}_i, \overline{cu}_i) \subseteq \overline{ML} \times \overline{MU} | i \in [0,n]\}$, $s_0 = (\overline{ml}_0, \overline{mu}_0) = ((\overline{l}_0^1, \overline{l}_0^2, \cdots \overline{l}_0^n), (\overline{u}_0^1, \overline{u}_0^2, \cdots \overline{u}_0^n))$ is the initial configuration, $A$ is a set of action $\{a_i = \overline{ca}_i \subseteq \overline{CA} \mid i \in [0,n]\}$ and transition $\rightarrow \subseteq S \times MA \times S$ is the transition rule for a compound reaction step such that:

$$\frac{g(T) = true}{(\overline{ml}, \overline{mu}) \xrightarrow{\overline{ma}} (\overline{ml'}, \overline{mu'})}$$

where the compound transition can be refined to the sub-compound transition of each atom block $ab_i$ of $B'$. $\forall i \in [1,n]$, $(\overline{ml'}.\overline{l}^i, \overline{mu'}.\overline{u}^i)$ is the configuration of $ab_i$, and $\forall (port1 \times port2) \in W'$ is the flatted connection with the cascaded expression $f_i'$:

$$\frac{g(T) = true}{(\overline{ml'}.\overline{l}^i, \overline{mu'}.\overline{u}^i) \xrightarrow{\overline{ma}.\overline{a}^i} (\overline{l}^{i\prime}, \overline{u}^{i\prime}) \bigwedge u(port2) \xrightarrow{f_i'} u'(port1)}$$

*4) LTS for MVBChart Model:* As defined in section III-A, a MVBChart parallel computation model $M$ is defined as a tuple $\langle CB, T \rangle$, where $CB$ is the outermost compound block, and $T$ is the synchronous trigger for computation. Hence, the labelled transition system of MVBChart model is the same as the definition for the outermost compound block.

### D. Co-Synthesis of MVBChart to MVBC

Generally speaking, the MVBChart model and the MVBC do not pose any constraints on architecture or programming language for implementation. It is desirable to keep the features of the hierarchical structure, parallel processing, and data flow of modern MVBC in the hardware and software implementations, automatically. As presented in the Fig. 2, we propose a platform including two processing units: an ARM (Advanced RISC Machines) processor for software modules and an FPGA (Field Programmable Gate Array) processor for hardware modules. The communication is realized via the connection between the pins of FPGA to the GPIO (General Purpose Input/Output) of ARM.

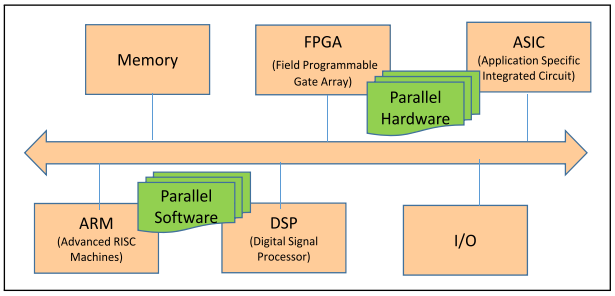



Fig. 2. Flexible architecture for the code synthesis from MVBChart model to MVBC system.

*1) Software-Hardware Partitioning:* Noting that many efficient partitioning algorithms have been proposed in the last decades, we can customize existing algorithms into TooMVBC, and focus more on the design of modeling, simulation, co-verification and code generation. To the best of our knowledge, this is the first time to customize and implement the partitioning task into a design framework, and we incorporate the most recently developed partitioning algorithm directly [10].

The key idea is to custom each atom block of MVBChart as a node in the partitioning algorithm and find a bipartition P on a communication graph denoted as $G(V,E)$, where $V$ is a set of nodes $\{v_1, v_2 \cdots v_n\}$ and $E$ is a set of edges $\{e_{ij} | 1 \leq i,j \leq n\}$, and $P = (V_h, V_s)$ such that $V_h \bigcup V_s = V$ and $V_h \bigcap V_s = \emptyset$. Then, the partitioning problem can be decided by a decision vector $\mathbf{x}(x_1, x_2 \cdots x_n)$, representing implementation way of n task modules. When the value of $x_i$ is 0 (1), the task module will be implemented in hardware (software). Objective of the problem is changed to search an n-dimensional space to find the optimal value of the decision vector on the objective function of hardware cost $H(x)$ and time constraints $T(x)$ as below:

$$P_0 : \begin{cases} minimize H(\mathbf{x}) \\ subject\ to T(\mathbf{x}) \leq M \\ \mathbf{x} \in \{0,1\}^n \end{cases} \tag{1}$$

Accompanied with the partitioning algorithm, we provide more flexibility for the implementation of MVBC according to different partitioning strategies on time and resource. Noting that many partitioning algorithms have been proposed in the last decades, which is not the main concern and contribution of

$$\frac{\{(l^j, g^j, a^j, l^{j\prime}) | max\{p(e_i^j) | g^j(v) = true\}\} \neq \emptyset \bigwedge g(T^j) = true}{(\overline{l}.l^j, \overline{u}.u^j) \xrightarrow{\overline{a}.a_j} (l^{j\prime}, u^{j\prime})}$$

$$\frac{\{(l^j, g^j, a^j, l^{j\prime}) | max\{p(e_i^j) | g^j(v) = true\}\} = \emptyset \bigwedge g(T^j) = true}{(\overline{l}.l^j, \overline{u}.u^j) \xrightarrow{\emptyset} (l^j, u^{j\prime})}$$

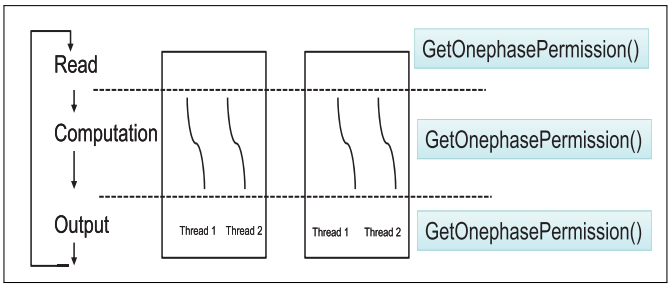



Fig. 3. Synchronization of parallel threads.

```
#ifndef TIMER_H
    #define TIMER_H
    int prephasecleared = 1;
    int totalthreads = 0;
    int onephaserealthreads = 0;
    CRITICAL_SECTION sync_section;
    void regist();
    void GetOnePhasePermission();
#endif
```

Listing 1. The header file timer.h for the scheduler.

```
void regist()
{
   EnterCriticalSection(&sync_section);
   totalthreads++;
   onephaserealthreads = totalthreads;
   LeaveCriticalSection(&sync_section);
}
```

Listing 2. The registration entrance in the file timer.c.

```
void GetOnePhasePermission()
{
  EnterCriticalSection(&sync_section);
  onephaserealthreads--;
  LeaveCriticalSection(&sync_section);

  while (1 == prephasecleared) {
   EnterCriticalSection(&sync_section);
   if (0 == onephaserealthreads) {
     prephasecleared = 0;
   }
   LeaveCriticalSection(&sync_section);
  }
   EnterCriticalSection(&sync_section);
   onephaserealthreads++;
   LeaveCriticalSection(&sync_section);

   while (0 == prephasecleared){
    EnterCriticalSection(&sync_section);
    if(onephaserealthreads==totalthreads){
   prephasecleared = 1;
    }
    LeaveCriticalSection(&sync_section);
   }
}
```

Listing 3. The consistency scheduler in the file timer.c.

this work, and we use our previous algorithms for partitioning [10]. It proposes a heuristic based on genetic algorithm and simulated annealing to solve the problem near-optimally, even for quite large systems. Based on the predefined libraries consisting of 36 general modules of MVBC domain, we can package the graphical MVBChart model for different MVBC classes easily with the addition of some basic modeling primitives. This changes the development into a more flexible style.

*2) Software Synthesis:* For software synthesis, previous work such as in [2], [19] usually flat their model and generate a single C function for the flatted model, which has no structure information and the code cannot be traced back to the original model. This is very difficult for the understanding and further update of developer. To overcome this inconvenience, we keep the structure and traceability and implement a more user friendly synthesis algorithm. We use C sub-function call to capture the compound block, and C statement of thread definition to capture the automata contained in atom block. Because the automata contained in the innermost atom block are running in parallel and synchronized with clock, thread for each automaton needs to be synchronized with the barrier, and we implement the dynamic barrier in two files named timer.h and timer.c, as the synchronization scheduler of all threads. Correlation between thread implementation and scheduler function is presented in Fig 3.

In the file timer.h, variable $totalthreads$ is used to denote the number of threads for all automata, $onephaserealthreads$ denotes the number of threads that are not in the scheduler, and $prephasecleared$ is used to control synchronization stage of all threads. In order to avoid operation conflicts on those variables, a system critical resource $sync\_section$ is declared to ensure exclusive access. Function declarations void regist() and void GetOnePhasePermission() are used for thread registration and synchronization, respectively.

In the file timer.c, function void regist() and void GetOnePh asePermission() are implemented on system access function to help to keep the timing consistency of all parallel threads. When thread for each automaton starts, thread register function void regist() is called to revise the number of total threads, and the number of threads that have not entered synchronization function. The consistency scheduler function void GetOnePhasePermission() is called multiple times during execution of threads. There are two while loops contained in the scheduler function, where the first is used to ensure that all threads have entered, and the second is used to ensure that all threads are synchronized to go out of suspension. In this way, all threads are dynamically synchronized to the system reaction step of importing inputs, updating changes and exporting outputs.

For the atom block $ab_i$, based on the element definition of atom block and the function interface contained in scheduler timer.c, synthesis engine parses those elements to get software description as below, with details presented in algorithm 2:

1. **Interface Definition.** Generate the variable declaration in header file from input ports $\{I_i | i \in [0, n]\}$ and output ports $\{O_i | i \in [0, n]\}$ of atom block $ab_i$.
2. **Type Generation.** Generate the type declaration in header file for each automaton $\{C_i | i \in [1, n]\}$ of atom block $ab_i$, and the location set $\{l_i | i \in [0, n]\}$ of the automaton is value space of the generated type. Based on the type, variable for state is declared. An accompanied variable for each local variable $\{v_i | i \in [0, n]\}$ and output data port is declared for parallel calculation and update.
3. **Thread Construction.** Generate a thread for each automaton $\langle L, l_0, V, A, E, P, T \rangle$ servicing value computation and state transition. First, we add the void regist() function call into thread. Then, the thread is divided into three segments in reaction manner: (1) read the variables declared for local variables and input data ports, (2) calculate the value of state and

variables within the SWITCH statement, where different transitions $\{e_i|i \in [0,n]\}$ starting from a state can be captured by the IF-ELSE statement inside a branch of CASE. Along with the use of accompanied signal, implementation avoids the side effects of sequential statements in C to ensure that value used for computation is the initial value of each step, (3) update the variables declared for local variables and output data ports. Before the implementation code of each segment, we add the function call of void GetOnePhasePermission() for global synchronization of all threads, and the function call of sleep() for synchronization between hardware and software.

The software implementation for the compound block is almost the same, with interface generation and function call generation of each sub-compound block or atom block. For the compound block $cb_i$, it is defined as a tuple $\langle I, O, B, W, T\rangle$, where $I$ is a set of input data ports $\{I_i|i \in [0,n]\}$, $O$ is a set of output data ports $\{O_i|i \in [0,n]\}$, $B$ is a set of sub-atom blocks $\{ab_i|i \in [0,n]\}$ and sub-compound blocks $\{cb_j|j \in [0,n]\}$. The synthesis engine parses those elements to get software description as below, with details presented in algorithm 3:

1. **Interface Definition.** Generate the variable declaration in header file from input data ports $\{I_i|i \in [0,n]\}$ and output data ports $\{O_i|i \in [0,n]\}$ of compound block $cb_i$.
2. **Function Call Generation.** Declare sub-function call for each atom block $\{ab_i|i \in [0,n]\}$ and sub-compound block $\{cb_j|j \in [0,n]\}$ of $cb_i$.

*3) Hardware Synthesis:* In terms of hardware synthesis framework, we use the architecture description code of component map in VHDL to capture the compound block, and the behavior description code of process definition in VHDL to capture the automata contained in atom block. The core synthesis algorithm for them is similar to the implementation algorithm presented in our previous work [8],.

For the atom block $ab_i$ contained in the selected block, it is defined as a tuple $\langle I, O, V, C, T\rangle$, where $I$ is a set of input data ports $\{I_i|i \in [0,n]\}$, $O$ is a set of output data ports $\{O_i|i \in [0,n]\}$, $V$ is a set of local variables $\{v_i|i \in [0,n]\}$, $C$ is a set of automata $\{C_i|i \in [1,n]\}$ refined as a tuple $\langle L, l_0, V, A, E, P, T\rangle$. The synthesis engine parses those elements to get hardware description as below:

1. **Interface Definition.** Generate the port declaration in the ENTITY definition of VHDL module, from input data ports $\{I_i|i \in [0,n]\}$ and output data ports $\{O_i|i \in [0,n]\}$. The bool type needs to be changed to std_logic, and integer needs to be changed to ranged integer type of VHDL for more efficient synthesis.
2. **Type Generation.** Generate the type declaration of VHDL module for each automaton $\{C_i|i \in [1,n]\}$, and location set $\{l_i|i \in [0,n]\}$ of the automaton is the value space. Based on the type, two accompanied signals used in process construction are declared for current active state and next active state. Two accompanied signals for each local variable $\{v_i|i \in [0,n]\}$ are declared for calculation, and an accompanied signal for each output data port $\{O_i|i \in [0,n]\}$ is declared for update.

**Algorithm 2:** Thread Implementation for Each Automaton Contained in Atom Block

**Input:** All parallel automata $C_i$ contained in the atom block $ab_i$.

**Output:** The threads for each automaton.

```
1  StringBuilder buffer = new StringBuilder();
   // Generate a thread for each automaton
   // Each thread is divided into three
      steps: read, computation, output.
      Each step needs to be synchronized
      with all the other threads.
2  for Each automaton C_i^j ∈ the atom block ab_i do
3      Append(buffer, void thread_ab_i_C_i^j());
4      Append(buffer, regist());
5      Append(buffer, DWORD start = timeGetTime());
       // The first step is for variable
          assignment.
6      Append(buffer, GetOnePhasePermission());
7      Append(buffer, steps 2 − 4 in algorithm 1);
       // The second step is for the
          automaton computation.
8      Append(buffer, GetOnePhasePermission());
9      Append(buffer, CASE (state));
10     for all state l_i ∈ L contained in a_i^j do
11         Append(buffer, WHEN l_i);
           // Select all transitions starting
              from the state l_i
12         Trans_temp[] ← Get_Trans(E, l_i);
           // Sort the selected transitions
              according to the priority.
13         Trans_temp[] ← Sort(Trans_temp[], P);
14         for each e_i ∈ Trans_temp[] do
15             Append(buffer, IF g(e_i));
16             Append(buffer, Update(a(e_i)));
17             Append(buffer, Destination(e_i));
       // The third step is for the output
          of data ports
18     Append(buffer, GetOnePhasePermission());
19     Append(buffer, steps 20 − 21 in algorithm 1);
20     Append(buffer, GetOnePhasePermission());
       // Optional: the system function sleep
          is used to keep consistency with
          the frequency of the hardware
          execution timer
21     Append(buffer, DWORD End = timeGetTime());
22     Append(buffer, sleep(H.f − Δt));
23 return buffer;
```

3. **Process Construction.** Generate the process construction for behavior in the style of two-stage. The first main process is used for update of output data port, state and variable, with corresponding accompanied signals. Then, the second process for value computation and state transition is generated for each automaton $\langle L, l_0, V, A, E, P, T\rangle$. Each automaton can be captured

**Algorithm 3:** Implementation for Compound Block

**Input:** The compound block $cb_i$

**Output:** The function for the compound block

```
StringBuilder buffer = new StringBuilder();
// Function for the compound block,
   which is used to call each function
   of the contained blocks.
buffer ← Append(buffer, void cb_i());
foreach sub-atom block ab_i ∈ cb_i do
    buffer ← Append(buffer, ab_i());
foreach sub-compound block cb_j ∈ cb_i do
    buffer ← Append(buffer, cb_j());
return buffer;
```

by a CASE statement of VHDL on the state signal $\{l_i|i \in [0,n]\}$. Transitions $\{e_i|i \in [0,n]\}$ and corresponding priorities $\{p_i \subseteq e_i < e_j|i,j \in [0,n], e_i \text{ and } e_j \text{ start from the same source state}\}$ can be captured by the IF ELSE statement inside a branch of WHEN. If action $\{a_i|i \in [0,n]\}$ has some assignment operations, output data port and local variable to be updated should be replaced with their accompanied signals.

For the compound block $cb_i$ contained in the selected block, it is defined as a tuple $< I, O, B, W, T >$, where $B$ is a set of sub-atom blocks $\{ab_i|i \in [0,n]\}$ and sub-compound blocks $\{cb_j|j \in [0,n]\}$, and $W$ is a set of point-to-point channel connections $\{w_i \subseteq port \times port|port \in \{cb_i.I \cup cb_i.O \cup B.I \cup B.O\}$. The synthesis engine parses those elements to get the hardware description as below:

1. Interface Definition. Generate the port declaration in the ENTITY definition of VHDL module from the input data ports $\{I_i|i \in [0,n]\}$ and output data ports $\{O_i|i \in [0,n]\}$ of block $cb_i$, which is obviously the same as the interface definition of atom block.
2. Instance Generation. Generate the COMPONENT instance for each sub-atom block $\{ab_i|i \in [0,n]\}$ and sub-compound block $\{cb_j|j \in [0,n]\}$ of $cb_i$. The COMPONENT instance declaration is the same as the ENTITY definition, except that the keyword COMPONENT is used to replace ENTITY.
3. Component Map. Generate the signal connection for each COMPONENT instance defined in the previous step according to the connection $\{w_i \subseteq port_1 \times port_2|port_{1(2)} \in \{cb_i.I \cup cb_i.O \cup B.I \cup B.O\}\}$. There are two types of connection. The first is that the port of the instance is connected to the port $\{cb_i.I \cup cb_i.O\}$ of the compound block $cb_i$. In this case, the port is mapped to the connected port directly. The second is that the port of the instance is connected to the port $\{B.I \cup B.O\}$ of other sub-atom and sub-compound blocks. In this case, we need to generate a temporary signal for the connection. are mapped to the temporary signal.

Based on the synthesis engines mentioned above, we can generate C code and VHDL code of MVBC from **MVBChart** model automatically, and the generated VHDL files can be synthesized into FPGA processor with VHDL ports mapped to FPGA pin, and the generated C files can be compiled into ARM processor with C variables mapped to ARM GPIO. Point-to-point channel communication is used to generate the configuration file for the connection between the pin of FPGA and GPIO of ARM. Furthermore, to keep execution cycle consistency between the reaction of hardware processor and the reaction of software processor, the frequency of FPGA processor is used to initiate parameter of sleep() function contained in the generated software, as presented the algorithm 2. Currently, the correctness of the code generation is mainly ensured through simulation and testing.

## IV. Experiment Results

We build the model with message data communication of class 5, and analyze the whole MVBC model through graphical model simulation and verification. For formal verification, the model is translated through verifier translator contained in TooMVBC, and safety-critical properties are formulated as logic assertions for message transfer, where four examples are presented as below. The first two properties are about data retransmission procedure and the last two are about data acknowledgment procedure. The translated file with suffix.elts and those properties are fed to the verifier Beagle for formal verification.

[ ]*(RECEIVER.SEND_AK derive (AK==true))*

[ ]*((AK==true) derive (AK_number==next_send))*

[ ]*(RECEIVER.SEND_NK derive (NK==true))*

[ ]*((NK==true) derive (NK_number==next_send))*

Based on the verification outputs, unsatisfiable properties are detected in the MVBChart model for message communication, where the information data transmission is unreliable. In specific, the last two assertions are violated, which means that two defects about the message retransmission are exposed. Note that the violation is not easy to be detected in graphical simulation because of the limitation of input patterns for simulation. For example, the third assertion implies the sender would not set the resend flag and the fourth means that the resend data packet could be incorrect.

Through the counterexample presented in Beagle and manual code review, we located violation for the third property in atom block $frame\_retransmission$ contained in compound block $message\_service$. An incomplete guard on a transition leads to the bug. The bug can be backtracked to C statement *(expected < NK_number ≤ send_not_yet)*, which is located in table 33 of the standard IEC 61375. The statement should be changed to *(expected ≤ NK_number ≤ send_not_yet)*. Furthermore, corresponding physical problem of the bug is that when source MVBC sends data packets for the first time and the first packet is lost in the link layer, receiver on dedicated MVBC will reply with a signal asking for retransmission of the first packet. However, the source MVBC could be trapped into a deadlock without issuing the retransmission, because the original wrong guard is not satisfied.

Moreover, the second bug corresponding to the violation of the fourth property is tracked to 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. 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 dedicated MVB controller will also reply with a number asking for retransmission of the lost packet 2. However, source MVB controller will mistakenly retransmit packet 3 when receiving the retransmission signal. Besides these two bugs, another four bugs about data transmission of table 33 and some ambiguousness about master controller transfer of workflow Figure 105 contained in the original IEC standard are detected, and have been confirmed by the IEC organization, and will be revised in the coming updated version. By fixing the bugs, we make modified MVBChart model pass verification.

## V. Conclusion

In this paper, we propose a design framework named TooMVBC using MVBChart computation model to capture specification of MVBC systems at high level. The specification model MVBChart offers modeling capabilities of major features of MVBC systems with both hardware and software components, data flow and control flow behaviors, memory control etc. Formal semantics of MVBChart for co-simulation and co-synthesis are proposed to overcome the gap between model and implementation. Graphical model validation through simulation and verification can help uncover design defects much easier at early stage of the development cycle. As mentioned in experiments, several critical bugs and some ambiguousness are located in the IEC standard by using TooMVBC. After all properties are satisfied in design, we can leverage TooMVBC to generate executable C and VHDL implementations from the validated model automatically according to the partitioning result. With those predefined libraries consisting of 36 general modules of MVBC domain features, different types of MVBC can be modeled rapidly and can be dynamically implemented according to different partitioning strategy. Generated code can be synthesized into corresponding processors on the proposed MVBC platform for execution with little manual work. Besides, based on those basic primitives, the framework can also be applied to the design of some data processing and control systems.

### A. Discussion and Future Work

(1) Currently, its reuse model libraries such as encoder module, decoder module, and memory control etc are mainly for MVBC, we will use the basic primitives to build more libraries to support the design of more complex controllers such as WTBC. Also, we will integrate more features to extend the ability of the basic primitives such as continues differential equation for function modeling. (2) Verification ability of MVBChart relies on the capability of Beagle, which supports computation tree logic assentations, and automatical trace back tool for counterexample of Beagle to trace of MVBChart would be researched. (3) The correctness of the code generation algorithm needs to be proved. Semantics preserving code generation, which has not been formally proved in existing tools yet, is a very hard but important research topic.

## References


[1] R. Alur, "Timed automata," in *Computing Aided Verification*. Berlin, Germany: Springer, 1999, p. 688.

[2] 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.

[3] Berry, "Scade-synchoronous design and validation of embedded control software," in *Proc. Workshop Next Gener. Design Verification Methodol. Distrib. Embedded Control Syst.* New Delhi, India: Springer, 2007, pp. 19–33.

[4] *IEC 61375-1*, Train Commun. Netw., Geneva, Switzerland, 2011.

[5] F. He, L. Yin, and B.-Y. Wang, "VCS: A verifier for component-based system," Tsinghua Univ., Beijing, China, Tech. Rep. 2013-01-0092, Oct. 2013.

[6] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre, "Lazy abstraction," *ACM SIGPLAN Notices*, vol. 37, pp. 58–70, Oct. 2002.

[7] X. Iturbe, A. Zuloaga, J. Jiménez, J. Lázaro, and J. L. Martín, "A novel soc architecture for a mvb slave node," in *Proc. Annu. Conf. Ind. Electron.*, Nov. 2008, pp. 1455–1460.

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

[9] Y. Jiang *et al.*, "From stateflow simulation to verified implementation: A verification approach and a real-time train controller design," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Dec. 2016, pp. 1–11.

[10] Y. Jiang *et al.*, "Uncertain model and algorithm for hardware/software partitioning," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Oct. 2012, pp. 243–248.

[11] Y. Jiang, H. Zhang, X. Song, W. Hung, M. Gu, and J. Sun, "Verification and implementation of the protocol standard in train control system," in *Proc. 37th Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2014, pp. 549–558.

[12] J. Jimenez, J. L. Martin, U. Bidarte, A. Astarloa, and A. Zuloaga, "Design of a master device for the multifunction vehicle bus," *IEEE Trans. Veh. Technol.*, vol. 56, no. 6, pp. 3695–3708, Nov. 2007.

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

[14] J.-H. Lee, J.-G. Hwang, D. Shin, K.-M. Lee, and S.-U. Kim, "Development of verification and conformance testing tools for a railway signaling communication protocol," *Comput. Standards Interface*, vol. 31, no. 2, pp. 362–371, Feb. 2009.

[15] Z. Li *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.

[16] Z. Li, F. Yang, and Q. Xing, "Design of multifunction vehicle bus controller," in *Computer and Computing Technologies in Agriculture*. New York, NY, USA: Springer, 2010, pp. 177–183.

[17] H. Ma, H. Zhang, and M. Gu, "Heterogeneous model merging based on model transformation," *Int. J. Model. Optim.*, vol. 6, no. 1, p. 39, 2016.

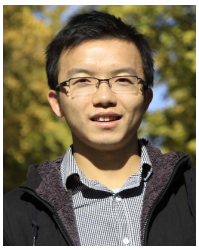
[18] R. Aarthipriya and S. Chitrapreyanka, "Fpga implementation of multifunction vehicle bus controller with class 2 interface and verification using beaglebone black," *Int. J. Sci. Eng. Res.*, vol. 3, no. 5, pp. 3221–3225, 2015.

[19] I. Sander and A. Jantsch, "System modeling and transformational design refinement in ForSyDe," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 1, pp. 17–32, Jan. 2004.

[20] 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.

[21] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "PAT: Towards flexible verification under fairness," in *Proc. Int. Conf. Comput. Aided Verification*. Paris, France: Springer, 2009, pp. 709–714.

[22] R. Whitfield *et al.*, "System and method for automatic train operation," U.S. Patent 6 135 396, Oct. 24, 2000.
[23] H. Zhang, Y. Jiang, H. Liu, H. Zhang, M. Gu, and J. Sun, "Model driven design of heterogeneous synchronous embedded systems," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Oct. 2016, pp. 774–779.
[24] H. Zhang, Y. Jiang, X. Song, W. N. Hung, M. Gu, and J. Sun, "Tsmart-galsblock: A toolkit for modeling, validation, and synthesis of multi-clocked embedded systems," in *Proc. Found. Softw. Eng.*, 2014, pp. 711–714.



**Yu Jiang** received the B.S. degree in software engineering from the 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 was a Post-Doctoral Researcher with the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 2016. He is currently an Associate Professor with Tsinghua University. His current research interests include domain specific modeling, formal computation model, formal verification, and their applications in embedded systems.

**Mingzhe Wang** received the B.S. degree in software engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2018. He is currently pursuing the Ph.D. degree with the School of Software Engineering, Tsinghua University, Beijing. His research interests include software testing, mainly focusing on fuzzing of distributed systems.

**Zhuo Su** received the B.S. degree in software engineering from Northeast University, Shenyang, China, in 2018. He is currently pursuing the Ph.D. degree with the School of Software Engineering, Tsinghua University, Beijing, China. His research interests include model driven design and code synthesis.

**Yixiao Yang** received the B.S. degree in software engineering from Nanjing University, Nanjing, China, in 2015, and the Ph.D. degree from the School of Software Engineering, Tsinghua University, Beijing, China, in 2020. He is currently a Post-Doctoral Researcher with the School of Software Engineering, Tsinghua University. His research interest includes software testing, mainly focusing on model based embedded software testing.

**Huihui Wang** (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Virginia, Charlottesville, VA, USA, in August 2013. In August 2013, she joined the Department of Engineering, Jacksonville University, Jacksonville, FL, USA, where she is currently an Associate Professor and the Founding Chair of the Department of Engineering. In 2011, she was an Engineering Intern with Qualcomm, Inc. She is the author of more than 50 articles and holds one U.S. patent. Her research interests include cyber-physical systems, the Internet of Things, healthcare and medical engineering based on smart materials, robotics, haptics based on smart materials/structures, ionic polymer metallic composites, and microelectromechanical systems.