This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

Design of Mixed Synchronous/Asynchronous Systems with Multiple Clocks

Yu Jiang, Hehua Zhang, Huafeng Zhang, Han Liu, Xiaoyu Song, Ming Gu, and Jiaguang Sun

Abstract—Today's distributed systems are commonly equipped with both synchronous and asynchronous components controlled with multiple clocks. The key challenges in designing such systems are (1) how to model multi-clocked local synchronous component, local asynchronous component, and asynchronous communication among components in a single framework. (2) how to ensure the correctness of model, and keep consistency between the model and the implementation of real system. In this paper, we propose a novel computation model named GalsBlock for the design of multi-clocked embedded system with both synchronous and asynchronous components. The computation model consists of several hierarchical compound and atom blocks communicating with data port connections. Each atom block can be refined as parallel mealy automata. The synchronous component can be captured in an atom block with the corresponding local control clock while the asynchronous component in an atom block without clock, and the asynchronous communications can be captured in the data port connections among blocks. The unified operational semantics and formal semantics are defined, which can be used for simulation and verification, respectively. Then, we can generate efficient VHDL code from the validated model, which can be synthesized into the FPGA processor for execution directly. We have developed the graphical modeling, simulation, verification, and code generation toolkit to support the computation model, and applied it in the design of a sub-system used in the real train communication control.

Index Terms—Synchronous and Asynchronous fusion System, Computation Model, Model Validation and Implementation

1 Introduction

Embedded systems are being widely used and are vital to many critical complex applications. They usually involve concurrent behaviors with different local control clocks, which leads to several challenges for the traditional computation model. The first challenge that the computation model faces is the modeling capability that how to evaluate the behavior of the local synchronous component, asynchronous component and their communications in a single model. For synchronous components, operations are coordinated under the centralized control of several local clocks. For asynchronous components, instead, they operate under distributed control, producing outputs in response to input changes. The asynchronous communication among these components should be synchronized with all corresponding clocks. The second challenge focuses on the analytical capability of the computation model, including how to ensure the correctness of the model to satisfy the function descriptions, and how to keep the consistency between the computation model and the executable implementation.

- *Y. Jiang and H.F. Zhang are with the Department of Computer Science and Technology, TNLIST, KLiss, Tsinghua University, China. E-mail: jiangyu10@mails.tsinghua.edu.cn*
- *H.H. Zhang, H Liu, J Sun, and M. Gu are with the School of Software, TNLIST, KLiss, Tsinghua University, China.*
- *X. Song, is with the Dept. ECE, Portland State University, USA.*

This research is supported in part by NSFC Programs (No.61202010, No.91218302), National Key Technologies R&D Program (No.SQ2012BAJY4052) and 973 Program (No.2010CB328003), and Tsinghua University Initiative Scientific Research Program (20131089331). Correspondence authors : Hehua Zhang and Yu Jiang. {jiangyu10,zhang-hh04}@mails.tsinghua.edu.cn

A lot of researchers have made many efforts to the computation models [4], [21], but there is no practical solution for synchronous and asynchronous fusion systems with multiple local control clocks. In this paper, we present an automaton [32] and block diagram [35] based computation model, named GalsBlock, to address the challenges in modeling, validation and implementation. In the proposed computation model, a system is modeled as a combination network of compound and atom blocks communicating through data port connections. The compound block can be refined by some sub-compound and sub-atom blocks hierarchically, and the atom block is refined by some parallel mealy automata with optional clocks. Despite the regular state transitions, we allow users to append complex actions and priorities on transitions to enhance the modeling ability. The atom block with clock is used to capture the behavior of synchronous component, and the atom block without clock is used to capture the behavior of asynchronous component. The connections among data ports attached on different blocks are used to capture the asynchronous communications, and the expressions appended on the connection are used to facilitate the modeling of data-oriented behaviors. In order to solve the indeterminacy caused by parallel execution, we use the local clocks and signal dependencies to give a topological sort for all atom blocks' computation. Then, the design and implementation of complex embedded systems can be simplified to three steps. First, we can build the graphic model of GalsBlock based on the system requirement and function descriptions. Then, the model can be simulated and verified for all kinds of proper-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

Modeling Capability	Esterel	Lustre	State-Charts	Signal	SSM	CRP	MC-Esterel	SHIM	GalsBlock
Parallel process	✓	✓	✓	✓	✓	✓	✓	✓	✓
Hierarchical	✓	✓	✓	✓	✓	✓	✓	✓	✓
Data-flow	X	✓	X	✓	✓	X	X	X	☑
Control-flow	✓	X	✓	X	✓	✓	✓	✓	✓
Multi-clock	X	X	X	X	X	✓	✓	✓	✓
Analytical Capability	CEC	SCADE1	Simulink	Poly chrony	SCADE6	X	Esterel studio7	X	Tsmart
Formal verification	☑	☑	X	☑	☑	X	☑	X	✓
VHDL Code generation	☑	X	☑	X	X	X	☑	X	✓
C Code generation	X	✓	☑	✓	✓	X	☑	X	☑
Graphical Simulation	X	✓	✓	✓	✓	X	✓	X	✓
Available version	Academic version	Industrial version	Industrial version	Academic version	Industrial version	X	X	X	Academic Prototype

"✓" means strong support, "☑" means limited support, "X" means no support

Fig. 1. The modeling capabilities of the system level modeling languages based on formal computation model and the corresponding available tools supporting the analytical capabilities.

ties derived from the requirements. If properties are not satisfied, we should return back to the modeling stage to find bugs in the model. This helps us find problems in the early stage of system design. Finally, after all properties are satisfied, we can generate the executable implementation from the validated model automatically. We apply GalsBlock and the developed toolkit to the design of a multifunction vehicle bus control system used in the real train communication network according to the standard IEC 61375.

The main contributions are: (1)First, a novel computation model named GalsBlock is proposed to support hierarchical decomposition, fusion of synchronous and asynchronous concurrent process execution, and data-oriented asynchronous signal communication. (2) Second, the operational semantics of GalsBlock is described in detail, based on which, we can do some graphical simulations. (3) Third, the formal semantics of GalsBlock is determined by its equivalent labeled transition system, based on which, we can do some verifications. The labeled transition system is constructed bottom-up, by starting from the lowest level automaton style chart to the top level block. (4) Fourth, code generation algorithms for automatic implementation of GalsBlock model are presented. This helps to keep the consistency between the validated model and the possible implementations. We implement the supporting toolkit and apply it to the real design of a multifunction vehicle bus control system of real train communication network.

The paper is organized as follows: related work is presented in Section 2. The proposed system level design language GalsBlock is presented in Section 3, including the graphical elements for modeling, operational semantics for simulation, and formal model of computation for verification. Section 4 presents the implementation of GalsBlock, including the automatic code generation algorithms. Experiment results on a real system design and implementation is given in Section 4, and we conclude in Section 6.

2 Related Work

A large body of work has been dedicated to facilitate the design of embedded systems. In the literature, the formal computation model based approach is appealing, because it provides a unified basis for formal analysis to achieve expected correctness. Traditionally, embedded systems are supposed to be controlled by a single synchronous clock, and the formal computation modeling languages constructed on this hypothesis mainly contain Esterel [9], Lustre [17], Statecharts [19] and Signal [25]. These languages are the basis for traditional synchronous system design, but are not fit for the multi-clocked systems with both synchronous and asynchronous components.

Embedded systems on the current market are more and more complex that the synchronous hypothesis is not valid. Most systems consist of hundreds of components, which can be controlled by many local clocks. The formal computation modeling languages for multi-clocked systems with both synchronous and asynchronous components mainly contain CRP [6], MC-Esterel [7], and SHIM [16]. CRP combines the synchronous reactive model of Esterel [24] with the asynchronous coupling of CSP [22] to offer a mathematically elegant framework. Locally synchronous Esterel modules communicate through rendezvous

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

channels. The problem is that it is hard to support data-driven operations and rendezvous protocol through asynchronous coordinators. Its variants such as CRSM, ECRSM [28] have similar limitations. MC-Esterel is specifically developed for the design of multi-clocked digital systems. The designer is responsible for creating communication mechanisms among different clock domains. Esterel modules need explicit clocks and a designer has to construct underlying synchronizers to guarantee the synchronization among these modules. While MC-Esterel provides a powerful mechanism for modeling asynchronous and multi-rate systems, the problem is that a designer has to work at a relatively underlying level and the productivity is limited. In addition, its support for data-driven operations is quite limited due to its reliance on Esterel. The main idea of SHIM is that both hardware and software functions are written as C-like functions and the communication between these asynchronous objects is mapped to a restricted Kahn Process Network. The major limitation of SHIM is the lack of support for modeling reactivity and synchrony behaviors of system components.

Based on those formal computation modeling languages, there are many corresponding toolkits. For multi-clocked systems with both synchronous and asynchronous components, Esterel studio v7 [5] supports the design of digital systems based on MC-esterel. But the code generation capacity is limited, many basic modeling constructs are not supported, and the simulation is not visual. The verification tool Xeve only supports pure signal, and cannot deal with valued signals and variables. Furthermore, we cannot find an alive version for use anymore. Ptolemy [10] supports modeling, and simulation of mixed synchronous and asynchronous systems. A major problem area being addressed is the use of heterogeneous mixtures of computation models (e.g., asynchronous discrete event and synchronous dataflow) in hierarchical way. The inner model will lose some original properties when adjusting to the outer model of computation. Furthermore, it is primarily used as a simulation environment and can not be verified and synthesized. Simulink [12] also has the same disadvantage of no formal semantics. Furthermore, the operational semantics of the parallel execution of stateflow [18] is too complex and dependent on the relative position of the module. They are simulated in a sequential manner, from left to right, up to down, and can be interrupted by each other. The VHDL code generation algorithm is not efficient, neither. Besides these environment, some translation based frameworks are also proposed to solve the analysis of multi-clocked systems. For example, F. Doucet et al. [15] use a mixture of synchronous descriptions in Signal and asynchronous descriptions in Promela, and provide a translation from Signal modules to Promela processes. Each clock domain is described by a Signal module, and communication between two clock domains is described by Promela channel. The underlying deficiencies of these modeling languages are actually prevalent in those traditionally modeling and validation frameworks as presented in Fig 1, which motivates our work. The proposed GalsBlock has clear operational and formal semantics, as well as complete toolkit for hierarchical graphical modeling, validation and implementation.

3 GALSBLOCK

In this section, we introduce the formal computation model GalsBlock, including the graphical elements, operational semantics for simulation, and the formal semantics for verification.

3.1 GalsBlock Computation Model

The example in Fig 2 illustrates the features of GalsBlock. At the top level, the compound block **CompoundTop** consists of two sub-blocks (compound block **Compound1** and atom block **Atom1**). The compound block does not do computation, just presents the hierarchical decomposition of system structure, and the data flow path among the structured blocks. The clock attached on the compound block does not play a part, and just provides a virtual interface for the control clocks of its inner atom blocks. For example, the frequency of real clock **CLK2** is derived from the virtual clock **CLK1**, where the derived rules such as double and triple frequency can be configured according to different requirements. The dot attached on the right side of each block is used to denote the output data port, while the dot on the left side is used for the input data port. The input data ports of **CompoundTop** can be connected to the input ports of the two sub-blocks (b $\rightarrow$ g, a $\rightarrow$ c), and the output ports of the sub-blocks can be connected to the both input and output data ports of other blocks (e $\rightarrow$ f, h $\rightarrow$ i). The expression on the connection from port **b** to port **g** facilitates the data-oriented behavior modeling.

The asynchronous communication between two data ports is realized through asynchronous channel. Once the synchronous atom block finishes the computation at the beginning of the local control clock, the data from the output data port will flow through connections to the sink port, until the arrival of the input port of another atom block or the output port of the top block. The atom block on which the sink port is attached will read the value for computation at the beginning of its local clock. In case of conflicts of read-write operations during the parallel execution, the old value will be read according to the propagation delay. For example, block **Atom2** will update the output port **m** when the computation finishes at the periodical beginning of clock **CLK3**, and the updated value will flow through connection (m $\rightarrow$ o) until the connected input ports of another atom block. Because block

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

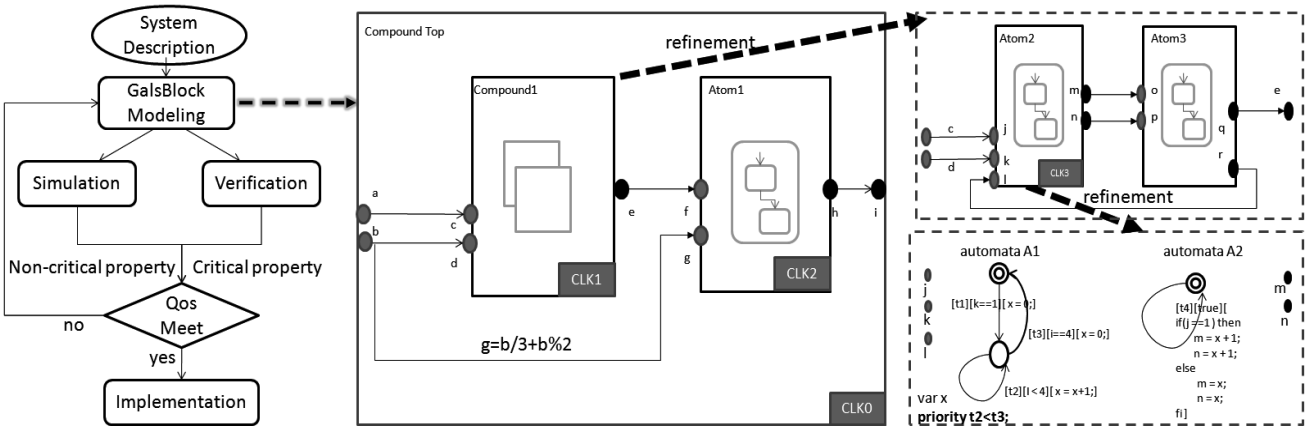


Fig. 2. The left side is the system design flow based on the computation model GalsBlock. The right side is an example of GalsBlock, including the graphical elements of compound and atom blocks. The top model consists of compound block Compound1 and atom block Aotm1. The compound1 is refined as atom blocks Atom2(synchronous) and Atom3(asynchronous). The Atom2 is refined as two automata running in parallel.

Atom3 is asynchronous, it will do computation and update the output data port **q** immediately. Then, the updated values will flow through connections (q $\rightarrow$ e, e $\rightarrow$ f) until the connected input ports of another atom block **Atom1**. At the next periodical beginning of clock **CLK2**, block **Atom1** will read the updated value for computation. If the periodical beginning of clock **CLK3** happens to be the same as clock **CLK2**, the parallel executions will lead to conflicts of read-write operations. In real world, those conflicts can be solved by the delay of computation and signal flow. Based on this fact, we embed the delay that read operation is prior to write operation when the conflicts caused by the same frequency of local clocks happen.

Besides the solution for the read-write operation conflict, we release certain types of cycle restriction in the model. The cycle, in which there is a synchronous atom block, is broken by the inner data communication principle. For example, the compound block **Compound1** which is refined as two atom blocks (**Atom2** and **Atom3**) contains a direct cycle. The connection from port **r** to port **l** results a cycle between the two atom blocks. If the output port **m** of block **Atom2** is updated when the computation finishes at a periodical beginning of the clock **CLK3**, the value will flow through the connection (m $\rightarrow$ o). Because block **Atom3** is asynchronous, it will perform computation and update the output data port **q** and **r** immediately on the changes of its input ports. The updated value of port **r** will be passed to port **l** of block **Atom2** instantaneously. But this will not lead to infinite computation cycle, because block **Atom3** will not read the updated value for computation until the next periodical beginning of clock **CLK2**. Remind that the cycle, in which all elements are the asynchronous atom blocks, will lead to the zero timing paradox.

With the hierarchical structure and dataflow presented in the compound block, system behaviors are described by parallel automata in the atom block. For example, the atom block **Atom2** is refined as two automata controlled by clock **CLK3**, with a local shared variable and a transition priority expression. For synchronous atom block, the inner automata are controlled by the same local clock. At each periodical beginning of the clock, each automaton is allowed to take a step of transition or stay in the current state. Each transition consists of three parts: name, guard, and action. The priority expression is defined on the name of two transitions in a single automaton in case that both transitions meet their guards. The action supports some basic control structures such as **IF ELSE** statement. All statements and guards are defined on the ports and local variables. For parallel execution of multiple automata, there will also be conflicts on read-write operations. Each local variable and output data port can be written by one automaton for only one time, and the write operation is prior to the read operation. When all automata finish the computation, values of output ports and local variables will be updated, and passed through connections to the endpoints, which are the input data ports of an atom block or the output data ports of the top block. For asynchronous block, the inner automata are triggered by the changes of the input data ports, and may take several transitions from the current state until no guard is satisfied.

3.2 Operational Semantic of GalsBlock

The operational semantics of GalsBlock model is defined on the configuration. A configuration is a maximal set of states that the system could be simultaneously in. Any subset of states is not a legal

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

configuration. Let **B** be the top block associated with a GalsBlock model, consisting of several synchronous and asynchronous atom blocks b_i. A legal configuration **C** satisfies: for each atom block b_i contained in **B** and each automaton c_i directly contained in b_i, **C** must contain exactly one state directly contained in c_i. Then, the operational semantics can be transferred to the configuration computation of GalsBlock.

Because the compound block just presents the decomposition of the system structure and does not do computation, hence, the computation of GalsBlock relies on the computation of atom blocks. The computation can be divided into three macro steps: (1) Import values from the environment to input data ports of the top model continuously, and pass those updated values to the input data ports of atom blocks through connections; (2) Atom blocks read updated values of the input data ports for configuration computation on the periodical beginning of each local control clock with the priority rules; (3) When an atom block finishes the computation, export all output data ports to the connected compound and atom blocks. The algorithm implementation of the three macro steps is described below. Notice that the second step for configuration computation is decomposed hierarchically, and the computation mechanism is different for synchronous and asynchronous atom blocks.

```
for each input port p_i of the top GalsBlock B do
    for each input port p_i^j connected to p_i through
    connection w_j (p_i → p_i^j) do
        if p_i^j is not the input port of an atom block then
            value of p_i^j is set as p_i;
            add p_i^j to the end list of p_i;
        else
            value of p_i^j is set as p_i;
        end
    end
end
```

Fig. 3. Import values from the environment to the atom blocks for computation.

```
for each atom block b_i of the top GalsBlock B do
    if b_i is a synchronous atom block then
        wait for clock CLK_i to be satisfied;
    else
        wait for the changes of input ports;
    end
    /*if write p_i happens at another block at the same
    time, read the old value*/
    read values of input ports p_i and local variables v_i;
    for each automata a_i^j contained in block b_i do
        compute the configuration c_i ;
    end
    if all parallel executions of automata are done then
        return the configuration of atom block C_i;
    end
end
```

Fig. 4. Computation of the atom block. All atom blocks and automata are computed in parallel.

```
for all transitions start from the current state do
    start with the transition with the highest priority;
    /*if write v_i happens at another automaton at the
    same time, read the new value*/
    read values of local variables v_i;
    if guard is true then
        execute the actions on the transition;
        set the current state to the sink state;
        return current state;
    end
end
```

Fig. 5. Computation of the automata contained in synchronous atom block are in parallel.

```
for all transitions start from the current state do
    start with the transition with the highest priority;
    /*if write v_i happens at another automaton at the
    same time, read the new value*/
    read values of local variables v_i;
    if guard is true then
        execute the actions on the transition;
        set the current state to the sink state;
        jump to the outmost for statement again;
    end
    return current state;
end
```

Fig. 6. Computation of the automata contained in asynchronous atom block are in parallel.

For the first step presented in the Fig 3, the top model of Galsblock imports the inputs from environment continuously. It should get the value and pass the value through connections until the endpoints arrive, which are the input data ports of atom blocks. Note that the value may flow through some compound blocks before arriving at the endpoints.

For the second step, the computation of GalsBlock is divided into the computation of atom block and automata hierarchically. The automata computation contained in synchronous atom block is triggered by the local clock, and all automata are allowed to take at most one transition only. While for the asynchronous atom block, the automata computation is triggered by the changes of input data ports, and all automata can take different numbers of transitions until no guard is satisfied. The asynchronous atom block reads the updated values immediately, while the synchronous atom block reads the updated values according to the local clock. Then, the parallel automata in each block will compute the next state and the values of the output ports and shared variables with priority in consideration. If there are more than one transitions starting from the current active state that can be triggered, the transition with the highest priority is chosen to be taken. After the recursive executions, the state configuration as well as values of the output data ports and shared variables will be returned. The details of the computation steps are described in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

Fig 4, 5 and 6. In the Fig 4, all atom blocks are said to compute in parallel, but there is a topological sort of all local clock signals. Based on the topological sort, computation is in sequential. If several local clocks happen to be triggered in the same time, it will lead to conflicts of read operations and write operations. The atom block that reads the signal executes before the atom block that writes the signal to solve the conflict, according to the introduction in the previous subsection. The parallel executions of automata contained in synchronous atome block and asynchronous atom block are presented in the Fig 5 and 6, respectively. The parallel executions of automata contained in a single atom block may also lead to conflicts of read and writ operations. No shared variables and output data ports can be written by two automata at the same time, and the write operation to a shared variable is prior to read operation to the shared variable. In this way, the conflicts can be avoided. The jump statement in the Fig 6 shows the asynchronous executions of multiple transitions until no guard is true.

For the third step presented in the Fig 7, each atom block updates the values of the shared variables and exports the output data ports. The value of the updated output data ports should be passed through connections until the endpoints arrive, which are the input data ports of an atom block, or the output data ports of the top block **B**.

```
for each atom block b_i ∈ the top GalsBlock B do
    for each output port p_i of the atom block b_i do
        read value of p_i;
        for each port p_i^j that is connected p_i through
        connection w_j do
            if p_i^j is not the input port of an atom block
            or the output port of B then
                value of p_i^j is set as p_i;
                all p_i^j to the end list of p_i;
            else
                value of p_i^j is set as p_i;
            end
        end
    end
end
```

Fig. 7. Export values of the output data ports of atom blocks to other blocks.

Based on the operational semantics, we have developed the simulation tool for GalsBlock. As described in the experiment section, the model can be simulated step by step. The user can capture the state of each atom block, and the value of each output data port and local variable for each configuration. This facilitates the designer to ensure the correctness that the behaviors of the model map the system requirements to be implemented. Through the simulation tool, most of the functional requirements can be checked on the model. While coding with the underlying programming language such as VHDL and C according to the requirements directly is more difficult than building the graphical GalsBlock model.

3.3 Formal Semantics of GalsBlock

For the formal semantics definition of the GalsBlock computation model, there are two methods for choice. The first method is to translate the GalsBlock model to an existing model that has formal semantics. Then, the translation of GalsBlock model can be interpreted and verified directly based on the supporting tools of the target model. The second method is to define the basic formal semantics by its equivalent labeled transition system [11], which is the basis for verification. The labeled transition system of the Galslbock model is constructed bottom-up, starting from the lowest level parallel automaton c_i of each atom block b_i, upwards to the top level block **B** through combination. The construction is defined as follows.

Definition 1S: An automaton c_i contained in a synchronous atom block is defined as a tuple $(L, l_0, V, A, E, P, clk)$, where L is a set of locations $\{l_i | i \in [0, n]\}$, $l_0 \in L$ is the initial location, V is a set of parameters $\{v_i | i \in [0, n]\}$ inherited from the local variables, input data ports, and output data ports of the atom block, A is a set of actions $\{a_i | i \in [0, n]\}$, E is a set of edges $\{e_i \subseteq L \times G(V) \times A \times L | i \in [0, n]\}$ between locations with the action and guard, $G(V)$ is the set of guard $\{g_i(V) | i \in [0, n]\}$ on parameters, P is the set of priority valuation function $\{p_i \subseteq e_i \rightarrow N | i \in [0, n]\}$ defined on the transitions that may take simultaneously, and clk is the synchronous local clock signal inherited from the atom block.

Let $(L, l_0, V, A, E, P, clk)$ be an automaton contained in the synchronous atom block. $U(V)$ is a set of parameter valuation functions $\{u_i \subseteq V \rightarrow B \cup N | i \in [0, n]\}$ from the parameters to the boolean or integer values. Then, the semantics of the automaton can be defined in the property 1. For the clk signal, there are two kinds of methods for embedding the signal into the labeled transition system. The first method is to use all local clocks to give a topological sort of all atom blocks when doing combination. The second method is to map the local clock to a basic physical clock and transfer the clk into the guard of each transition through the local clock re-mapping mechanism proposed in [23], and the transition with the highest priority will be chosen to be triggered. We choose the second method for easier implementation.

Property 1S: The semantics of the automaton c_i $(L, l_0, V, A, E, P, clk)$ contained in a synchronous atom block b_i can be defined as a labeled transition system $\langle S, s_0, \rightarrow \rangle$, where S is the set of configuration $\{s_i \subseteq L \times U(V) | i \in [0, n]\}$, $s_0 = (l_0, u_0)$ is the initial configuration, and $\rightarrow \subseteq S \times A \times S$ is the transition relation such that:

- $(l, u) \xrightarrow{a} (l', u')$, if there is a transition $e = (l, g, a, l')$ s.t. $p(e) = max\{p(e_i) | g(u(v)) \& g(u(clk)) == true\}$.
- $(l, u) \xrightarrow{\emptyset} (l, u)$, if there is no transition $e = (l, g, a, l')$ s.t. $g(u(v)) \& g(u(clk)) == true$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

Definition 2S: A synchronous atom block b_i is defined as a tuple (I, O, V, C, clk), 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 $\{c_i | i \in [1,n]\}$, and clk is the local clock used to trigger the computation.

Let (I, O, V, C, clk) be a synchronous atom block, consisting of n automata $\bigcup_{i=1}^{n} c_i$. The automaton c_i equals $(L^i, l_0^i, V^i, A^i, E^i, P^i, clk)$, where $V^i \subseteq I \cup O \cup V$. Then, the location set $\overline{L}$ for the atom block is defined as $\{\overline{l}_i \subseteq (l^1, l^2, \cdots, l^j, \cdots, l^n) | \ i \in [0,n], \forall j, l^j \in L^j\}$, where $\overline{l}_0$ equals $(l_0^1, l_0^2, \cdots, l_0^n)$. The action set $\overline{A}$ is defined as $\{\overline{a}_i \subseteq (a^1, a^2, \cdots, a^j, \cdots, a^n) | i \in [0,n], \forall j, a^j \in A^j\}$. For the automaton c_i, U^i is a set of parameter valuation function $\{u^i \subseteq V^j \rightarrow B \cup N | j \in [0,n]\}$ from the parameters to the boolean or integer value. Then, for the atom block, $\overline{U}$ is defined as $\{\overline{u}_i \subseteq (u^1, u^2, \cdots, u^j, \cdots, u^n) | i \in [0,n], \forall j, u^j \in U^j\}$, where $\overline{u}_0$ equals $(u_0^1, u_0^2, \cdots, u_0^n)$. Then, the semantics of the synchronous atom block can be defined as below.

Property 2S: The semantics of the synchronous atom block b_i (I, O, V, C, clk) can be defined as a labeled transition system $\langle S, s_0, \rightarrow \rangle$ based on the semantics of automata, where S is the set of configuration $\{s_i \subseteq \overline{L} \times \overline{U}(I \cup O \cup V) | i \in [0,n]\}$, $s_0 = (\overline{l_0}, \overline{u_0})$ is the initial configuration, where $\overline{l_0}$ equals $(l_0^1, l_0^2, \cdots, l_0^n)$ and $\overline{u}_0$ equals $(u_0^1, u_0^2, \cdots, u_0^n)$. Then, $\rightarrow \subseteq S \times A \times S$ is the transition relation such that:

- $(\overline{l}, \overline{u}) \xrightarrow{\overline{a}} (\overline{l'}, \overline{u'})$, when $g(u(clk)) == true$:

$$\overline{l'}.l^j = \begin{cases} l^{j'} & if \ \{(l^j, g^j, a^j, l^{j'}) | max\{p(e_i^j)| \\ & g^j(u^j(v)) == true\}\} \neq \emptyset \\ l^j & if \ \{(l^j, g^j, a^j, l^{j'}) | max\{p(e_i^j)| \\ & g^j(u^j(v)) == true\}\} = \emptyset \end{cases}$$

$$\overline{u'}.u^j = \begin{cases} u^{j'} & if \ \{(l^j, g^j, a^j, l^{j'}) | max\{p(e_i^j)| \\ & g^j(u^j(v)) == true\}\} \neq \emptyset \\ u^j & if \ \{(l^j, g^j, a^j, l^{j'}) | max\{p(e_i^j)| \\ & g^j(u^j(v)) == true\}\} = \emptyset \end{cases}$$

where $\forall j \in [1,n]$, each automaton can take one transition at most on each clock appearance, staying in the current state or jumping into the sink state with the highest priority. This is the key difference between parallel automata computation of synchronous atom block and that of asynchronous atom block.

In asynchronous atom block, the parallel automata do not need to be synchronized, and each automaton can take any step of transitions until on guard is true. Another difference is that the computation is not triggered by the clock but by changes of input ports. The label transition system for asynchronous atom block can be defined in the same way as below.

Property 1As: The semantics of the automaton c_i (L, l_0, V, A, E, P) contained in an asynchronous atom block b_i can be defined as a labeled transition system $\langle S, s_0, \rightarrow \rangle$, where S is the set of configuration $\{s_i \subseteq L \times U(V) | i \in [0,n]\}$, $s_0 = (l_0, u_0)$ is the initial configuration, and $\rightarrow \subseteq S \times A \times S$ is the transition relation such that:

- $(l, u) \xrightarrow{a} (l', u')$, if there is a transition $e = (l, g, a, l')$ s.t. $p(e) = max\{p(e_i) | \ g(u(v)) == true\}$.
- $(l, u) \xrightarrow{\emptyset} (l, u)$, if there is no transition $e = (l, g, a, l')$ s.t. $g(u(v)) == true$.

Property 2As: The semantics of the asynchronous atom block b_i (I, O, V, C) can be defined as a labeled transition system $\langle S, s_0, \rightarrow \rangle$ based on the semantics of automata, where S is the set of configuration $\{s_i \subseteq \overline{L} \times \overline{U}(I \cup O \cup V) | i \in [0,n]\}$, $s_0 = (\overline{l_0}, \overline{u_0})$ is the initial configuration, where $\overline{l_0}$ equals $(l_0^1, l_0^2, \cdots, l_0^n)$ and $\overline{u}_0$ equals $(u_0^1, u_0^2, \cdots, u_0^n)$. Then, $\rightarrow \subseteq S \times A \times S$ is the transition relation such that:

- $(\overline{l}, \overline{u}) \xrightarrow{\overline{a}} (\overline{l'}, \overline{u'})$:

$$\overline{l'}.l^j = \begin{cases} l^{j'} & if \ \{(l^j, g^j, a^j, l^{j'}) | max\{p(e_i^j)| \\ & g^j(u^j(v)) == true\}\} \neq \emptyset \\ l^j & if \ \{(l^j, g^j, a^j, l^{j'}) | max\{p(e_i^j)| \\ & g^j(u^j(v)) == true\}\} = \emptyset \end{cases}$$

$$\overline{u'}.u^j = \begin{cases} u^{j'} & if \ \{(l^j, g^j, a^j, l^{j'}) | max\{p(e_i^j)| \\ & g^j(u^j(v)) == true\}\} \neq \emptyset \\ u^j & if \ \{(l^j, g^j, a^j, l^{j'}) | max\{p(e_i^j)| \\ & g^j(u^j(v)) == true\}\} = \emptyset \end{cases}$$

The compound block may consist of several synchronous atom blocks and asynchronous atom blocks. It does not do any computation, just presents the hierarchical decomposition of the system structure and some connections of data ports. Hence, it has nothing to do with the computation semantics. Then, the semantics of GalsBlock can be reduced to the combine the parallel execution of all atom blocks.

Definition 3: The top block of a Galsblock model M is defined as a tuple (I, O, B, W, CLK), where I is a set of input ports $\{I_i | i \in [0,n]\}$, O is a set of output ports $\{O_i | i \in [0,n]\}$, B is a set of atom blocks $\{b_i | i \in [0,n]\}$, W is a set of connections $\{w_i \subseteq port \times port | port \in \{M.I \cup M.O \cup B.I \cup B.O\}\}$, and CLK is the set of distributed clocks used to control those synchronous atom blocks $\{clk^i | i \in [0,n]\}$.

Let (I, O, B, W, CLK) be the top block M, consisting of n atom blocks $\bigcup_{i=1}^{n} b_i$. b_i is defined as $(I^i, O^i, V^i, C^i, clk^i)$ for synchronous block and (I^i, O^i, V^i, C^i) for asynchronous block. Then, the location set $\overline{ML}$ is defined as $\{\overline{ml}_i \subseteq (\overline{l}^1, \cdots, \overline{l}^j, \cdots, \overline{l}^n) | \ i \in [0,n], \forall j, \overline{l}^j \in \overline{L}^j\}$, and $\overline{ml}_0$ equals $(\overline{l}_0^1, \cdots, \overline{l}_0^n)$. The action set $\overline{MA}$ is defined as $\{\overline{ma}_i \subseteq (\overline{a}^1, \cdots, \overline{a}^j, \cdots, \overline{a}^n) | i \in [0,n], \forall j, \overline{a}^j \in \overline{A}^j\}$. For the atom block b_i, $\overline{U}^i$ is a set of parameter valuation function vectors $\{\overline{u}^i \subseteq \overline{V}^j \rightarrow B \cup N | j \in [0,n]\}$ from the parameters to the bool or integer values. Then, for the top block, $\overline{MU}$ is defined as $\{\overline{mu}_i \subseteq (\overline{u}^1, \cdots, \overline{u}^j, \cdots, \overline{u}^n) | \ i \in [0,n], \forall j, \overline{u}^j \in \overline{U}^j\}$, where $\overline{mu}_0$ equals $(\overline{u}_0^1, \overline{u}_0^2, \cdots, \overline{u}_0^n)$. Based on the above definitions and properties, the labeled transition system for the model is defined as:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

Theorem 1: The semantics of a GalsBlock computation model $M\ (I, O, B, W, CLK)$ is defined as a labeled transition system $\langle S, s_0, \rightarrow \rangle$, where S is the configuration $\{s_i \subseteq \overline{ML} \times \overline{MU}(V) | i \in [0, n], V \subseteq B.I \cup B.O \cup B.V\}$, $s_0 = (\overline{ml_0}, \overline{mu_0})$ is the initial configuration. Transition $\rightarrow \subseteq S \times MA \times S$ is the transition relation such that:

- $(\overline{ml}, \overline{mu}) \xrightarrow{\overline{ma}} (\overline{ml'}, \overline{mu'})$, where $\forall j \in [1, n]$, $\overline{ml'}.\overline{l}^j = \overline{l}^{j'}$, $\overline{mu'}.\overline{u}^j = \overline{u}^{j'}$. $\forall (port1 \times port2) \in W$, $u(port2) \rightarrow u'(port1)$. $\overline{l}^{j'}$ and $\overline{u}^{j'}$ are the combinational status of atom block b_j.

Then, we can formalize decision problems in GalsBlock in the same way as timed automata [1], [2]. Based on the semantics of the labeled transition system, the verification team in our lab has developed a verification tool [20]. Then, several safety critical properties can be verified directly, and the incompleteness of simulation can be overcome through this way.

4 Implementation of GalsBlock

In this section, we concentrate on the automatic implementation, where another strength of GalsBlock is demonstrated compared to some system level modeling languages and tools. Generally speaking, GalsBlock does not need special programming language for implementation. However, it is desirable to map the hierarchical structure, parallel processing, synchronous and asynchronous behavior descriptions to a corresponding programming language. Then, the consistency between GalsBlock and the implementation of real system can be acquired better.

Following the above principles, we choose VHDL code [30], [3] for the implementation of GalsBlock computation model. Because VHDL provides good ways to describe both synchronous behavior and asynchronous behaviors. System implementation by programming VHDL code directly is complex and intuitionistic. We will overcome this gap by defining some mapping mechanism to derive the underlying VHDL code from the abstract GalsBlock model. The code is generated in the manner of three stages, which is cumbersome to write but is the best organization of VHDL code description. Compared to the one-stage and two-stage manner, it is cumbersome to write but can eliminate the instability and glitches, and can be better synthesized. It is conductive to the timing path grouping into the programmable logic devices such as FPGA. Each atom block can be mapped to an entity description of VHDL code. The parallel automata can be mapped to the process description contained in VHDL entity. The local clock used to control the synchronous block can be mapped to the process trigger signal of VHDL code. The compound block and connections can be mapped to component map of VHDL code. When the user chooses the top block, the kernel should generate a set of VHDL codes for the computations of all atom blocks, and the hierarchical relations contained in the compound block.

```
ENTITY PORT GENERATION;
for each port p_i of the atom block b_i do
    if p_i.port_type equals in then
        add "p_i.port_name : IN p_i.data_type" ;
    else
        add "p_i.port_name : OUT p_i.data_type" ;
    end
end
add "RST, CLK: IN std_logic" ;

TYPE AND SIGNAL GENERATION;
for each chart c_i ∈ the atom block b_i do
    add "TYPE T_state_c_i.name IS (all states in c_i)" ;
    add "SIGNAL S_c_i.name : T_state_c_i.name" ;
    add "SIGNAL Next_S_c_i.name : T_state_c_i.name" ;
end
for each shared variable v_i of b_i do
    if v_i.data_type equals boolean then
        changes the data_type to std_logic;
    end
    add "SIGNAL v_i.name: v_i.data_type" ;
end

PROCESS DESCRIPTION GENERATION;
for each chart c_i ∈ the atom block b_i do

    /*the first segment process of the behavior*/;
    /*the sensitive signal of the process is CLK*/;
    if rst then
        reset SIGNAL S_c_i;
    else
        update SIGNAL S_c_i with Next_S_c_i;
    end

    /*the second segment process of the behavior*/;
    /*the sensitive signal of the process is S_c_i*/;
    add CASE statements on the state S_c_i;
    for each state value in the chart do
        add a WHEN branch;
        while transitions from this state is not NULL do
            select the one with the highest priority;
            add a IF branch in this WHEN branch;
            update NEXT_S_c_i with the target;
        end
    end

    /*the third stage process of the behavior*/;
    /*the sensitive signal of the process is CLK*/;
    add CASE statements on the state S_c_i;
    for each state value in the chart do
        add a WHEN branch;
        while transitions from this state is not NULL do
            select the one with the highest priority;
            add a IF branch in this WHEN branch;
            add the action embedded on the transition;
        end
    end
end
```

Fig. 8. VHDL code generation algorithm for the synchronous atom block contained in the Galsblock model.

First, let us see the code generation of the synchronous atom block presented in the Fig 8. The atom block contains priorities, input ports, output

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

ports, shared variables, and some parallel automata to describe the behaviors. We should process these elements into an equivalent VHDL entity. The first step is to generate the ENTITY definition of a VHDL module for the input ports and output data ports. Because we support boolean and integer values in GalsBlock computation model, we need to change the boolean to std_logic in VHDL for more efficient synthesis. Besides, we also need to add the clock signal CLK and system reset signal RST into the interface definition of the ENTITY. The second step is to generate a signal for each shared variable, which is used for behavior descriptions. For each automaton, we need to add a type definition for the state and generate the three-stage code. The first stage is the update of state, the second is the state transition, and the third is the output computation. The second and third stage can be captured by the CASE statement of VHDL, and the priority of different transitions can be captured by the IF ELSE statements inside a branch of WHEN. If the action on the transition has some assignment action, the port or the variable to be updated should be replaced with the temporary signal Next_name. The code generation algorithm for the asynchronous atom block is the same, except that the sensitive signals of process are changed to the input data ports without the clock signal.

Then, let us see the code generation algorithm of the compound block presented in the Fig 9. The compound block contains sub-atom block instance declarations, sub-compound block instance declarations, connections among these blocks, input data ports, and output data ports. We should process these elements to an equivalent VHDL module. The first step is to generate the ENTITY definition, in the same way as the atom block described above. The second step is to generate the COMPONENT instance for each contained block, and some temporary signals for connection. We need to declare an instance for each block contained in the selected compound block. There are two kinds of connections. The first is the connection between two inner blocks, and the second is the connection between the compound block and the inner block. For the first, we need to generate some temporary signals for component port map. We extend the name of the right port of the connection to the temporary signal. The third step is to initiate the signal connection for each COMPONENT instance according to the connection. If the port of the component is connected to a inner port, it is mapped to the generated temporary signal. Otherwise, it is mapped to the original port signal. The data-oriented behaviors appended on the port connections are transferred to the signal update of VHDL statements.

```
ENTITY DECLARATION GENERATION;
/*same as the entity generation of atom block*/;
/*the interface ports of the compound block*/;
for each port p_i of the compound block B do
    if p_i.data_type equals boolean then
        changes the data_type to std_logic;
    end
    if p_i.port_type equals in then
        add "p_i.port_name : IN p_i.data_type" ;
    else
        add "p_i.port_name : OUT p_i.data_type" ;
    end
end
add "RST, CLK: IN std_logic" ;

COMPONENT INSTANCE GENERATION;
/*the instance of each contained atom blocks*/;
/*include the entity declaration of atom blocks*/;
for each block b_i contained in this compound block B do
    add the COMPONENT definition of the block b_i;
    same as the entity generation of atom block;
end
/*the temporary signal for inner connection*/;
for each connection w_i ∈ this compound block do
    if w_i do not contain the port of B then
        add a temporary signal for mapping;
        add "SIGNAL w_i.right_port.name_temp ;
        : w_i.right_port.data_type";
    end
end

COMPONENT SIGNAL MAP GENERATION;
/* Initiate the port connection of each atom block*/;
for each each block b_i ∈ this compound block B do
    for each port p_i of this block b_i do
        for each connection w_i in this compound block do
            if one port of w_i equals p_i then
                if the other port of w_i ∉ B then
                    use the other port of w_i to connect p_i;
                else
                    connect p_i with the generated ;
                    w_i.right_port.name_temp ;
                end
            end
        end
    end
end

DATA-ORIENTED BEHAVIOR GENERATION;
/*Add the action embedded on the connection*/;
for each connection w_i in this compound block do
    if there is action on the connection then
        replace the operation and parameter;
        append the action statement to the module;
    end
end
```

Fig. 9. VHDL code generation algorithm for the compound block contained in the Galsblock model.

Based on the code generation algorithms, the constructed Galsblock computation model can be chosen to be implemented, and get the VHDL code automatically. We have implemented the code generation tool. With the help of Xilinx ISE toolkit [33], we can also simulate the generated VHDL code, and compare the result of the code simulation with the result of the GalsBlock model simulation. We have done large amounts of simulation comparisons to prove the con-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

sistency between the model and the generated code. Furthermore, the generated code can be synthesized and loaded into the FPGA processor to run directly. This facilitates the designer to ensure the consistency of the model and the implemented system.

5 Experiment Results

We have implemented the graphical modeling, simulation, verification and code generation toolkit to support the GalsBlock computation model. Then, we apply it to the design of a real sub-system contained in the train communication network control [29], [14], [13], which is a safety critical embedded system.

The system consists of many multifunction vehicle bus (MVB) controllers interconnecting devices within a vehicle. The MVB controller mainly provides time-critical process data communication and delay-tolerant information data communication. The communication is controlled by one master, which controls the sending of all data frames on the link layer bus. The master broadcasts a master frame, which carries the identifier for a process data frame. The device which sources this process data responds by broadcasting a slave frame. In order to accomplish this function, MVB controller needs a master frame sender control module and a process data response control module. In each basic period, the sender control module reads 8 master frames from the memory and send them onto the link layer, and the corresponding controller module will read process data from the memory and broadcast them onto the link layer for each master frame. The main modules are designed, validated, and generated automatically in GalsBlock, according to the descriptions of the standard [13].

We use the compound block $mf\_generator$ contained in the master frame sender control module to demonstrate the tool support of GalsBlock computation model for system design. It consists of three synchronous atom blocks ($mf\_generator\_ctrl$, mf_pool_ram, mf_pool_ram), and is used to generate the master frame to be sent. It will ask for the use of memory, and read the master frame data according to the output data port mf_addr. The data read procedure is controlled by the block $mf\_pool\_ram\_ctrl$. The master frame data read from the memory is stored in the block mf_pool_ram, and can be sent through the block $mf\_generator\_ctrl$. We build these atom blocks first, and drag them into the interface and connect those corresponding data ports. As presented in the Fig 10, the atom block $mf\_generator\_ctrl$ contains three parallel automata. The first automaton is used for the main control logic that reads the master from the pool ram and sends it to the link layer. The other two automata are used for the aided counters. For each computation, we can input the values of input ports. The resulted values of the output ports and shared variables, and the current state of the automata are highlighted in the graphical interface. For verification, the constructed can be interpreted by labeled transition system automatically, and can be input to Beagle [20] for counter-example analysis. If the model simulation and verification work as expected, we can generate VHDL code for this compound block directly. Four documents end with the suffix '.vhd' are generated, one document for the architectural description of the compound block, and three for the behavioral descriptions of atom blocks. For example, the size of VHDL code for the atom block $mf\_generator\_ctrl$ is 359 lines about 7 KB. The generated code size for the other blocks are presented in the Table 1. As presented in the Fig 11, the generated VHDL codes can be simulated and synthesized in the ISE development suit of xilinx, with some handwrite test bench case. The waveform of simulation in the Fig 10 shows that the generated codes work well and can generate 8 master frames in correct timing sequence. We compare the output result of the generated code simulation in ISE with the output result of the GalsBlock model simulation in the developed toolkit, and the results are consistent. Then, the generated codes can be synthesized to register transfer level circuit description, and the generated bit file can be loaded into the FPGA processor directly.

TABLE 1
The generated VHDL code for each block contained in the GalsBlock model.

The block name	Lines of the generated VHDL code
$macro\_timer$	115
sf_sender_timer	62
$cmf\_ssc\_combine$	312
$mf\_send\_control$	323
mux_in	49
$receiver\_controller\_out$	270
tm_access_ctrl	362
$mf\_generator$	146
$mf\_generator\_ctrl$	359
mf_pool_ram	191
$mf\_pool\_ram\_ctrl$	223
$main\_controller$	409

For the whole process data communication service of MVB controller, we need six blocks to send the master frame to the data link layer: the kernel compound block $mf\_generator$, the synchronous atom block $macro\_timer$, tm_access_ctrl, $mf\_send\_control$ and mux_in. The $macro\_timer$ is used to start the periodical phase with the output data port out_bp_start per 1ms. The output data port bp_num is used to determine the address of the master frame in this period. The memory data access is controlled by the tm_access_ctrl block. The $mf\_send\_control$ block is used to control the sending of the generated master frame from the atom block $mf\_generator\_ctrl$. The atom block mux_in transmits the master frame data to the sender of the system. Also, we need another four blocks to response the feedback pro-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

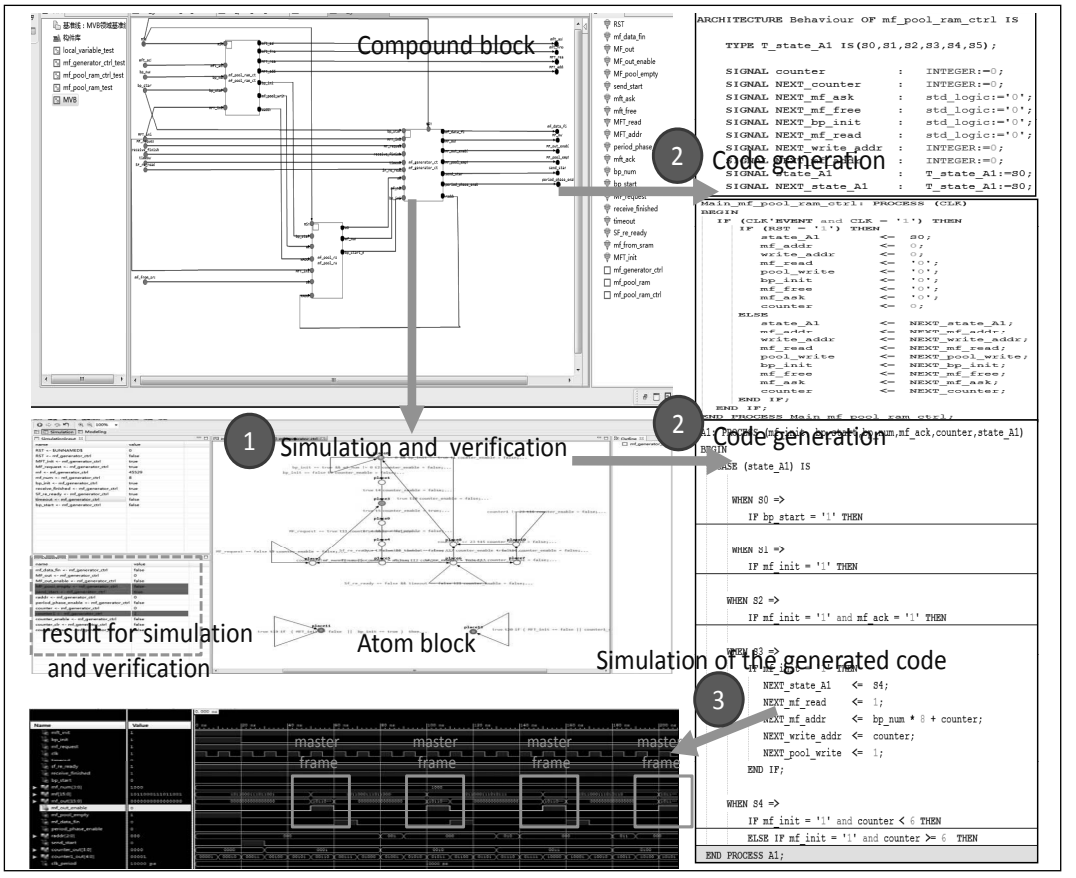


Fig. 10. The GalsBlock model for the master frame generator module of the process data service: modeling, simulation, verification, code generation in the developed toolkit and the generated code simulation in ISE. The $mf\_generator$ block can be refined by three synchronous atom blocks. The parallel automata and the generated VHDL code of $mf\_generator\_ctrl$ is also presented and simulated.

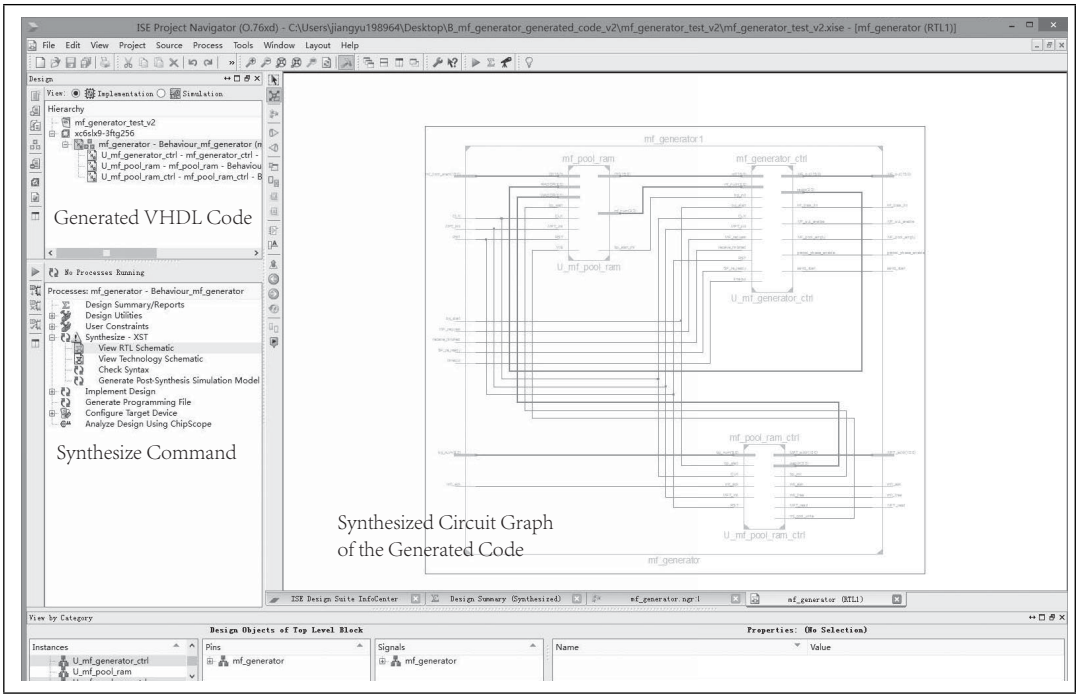


Fig. 11. All generated files are composed and synthesized in ISE development environment of xilinx company. Then, the generated bit file can be loaded into the FPGA processor.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

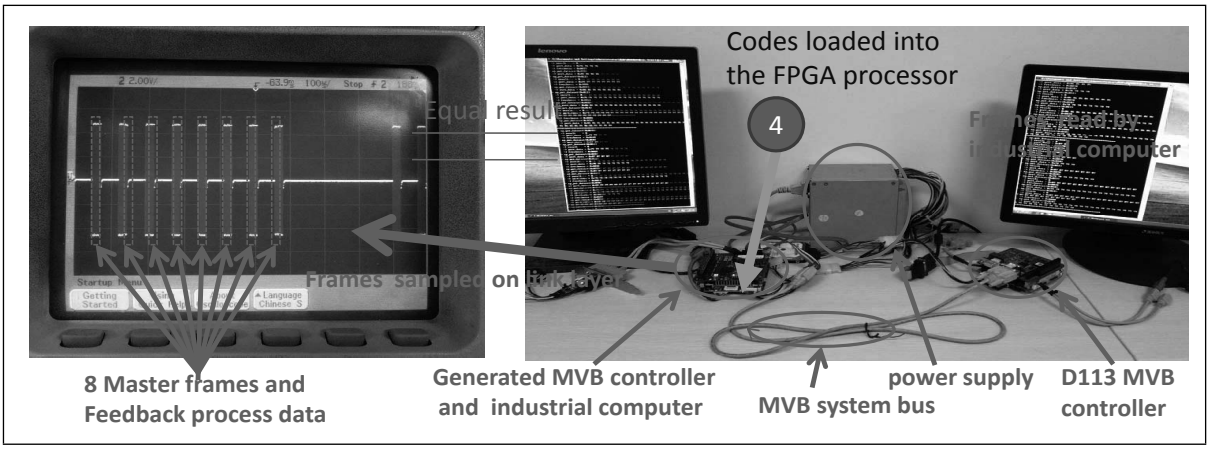


Fig. 12. We implement the MVB controller through loading the generated VHDL code with the original assistant code in to the FPGA processor, and connect it with the world most widely used MVB controller D113 from Duagon company throung MVB system bus. The waveform from the link layer sampled by the oscilloscope shows that the system implemented based on the GalsBlock computation model works well with the D113.

cess data to data link layer: the synchronous atom block $receiver\_controller\_out$ used to distinguish whether the received data frame is a master data frame or a process data frame. If the received data frame is a master data frame, the synchronous block $cmf\_ssc\_combine$ block is used to distinguish whether the process data frame should be responded by this device or not. If the device should response, the block sf_send_timer is used to decide when to read the process data from tm_access_ctrl. After that, the process data frame will be sent onto the link layer. Firstly, we build all blocks strictly according to the algorithm and pseudo code descriptions in the standard IEC 61375. Due to limited space, all Galsblock computation models can be found on the website [34]. Then, we do some simulations and verifications through the developed toolkit. Unfortunately, the first version of the constructed model can not accomplish the process data communication service. Through artificial analysis, we locate the problem in the atom block $mf\_pool\_ram\_ctrl$. The bugs have been certified through theoretical analysis and our engineering practice, and proposed to the standard organization. The revised GalsBlock model passes the simulation and verification.

Then, we generate the VHDL code from the revised GalsBlock model automatically, and load the generated codes presented in the table 1 into the FPGA processor of MVB controller. As presented in the Fig 12, the world most widely used MVB controller D113 from Duagon Company are connected to test the reliability of the generated system through the system bus. The implemented system is embedded into the industrial computer to get some instructions from the keyboard, such as communication start. We use the application running on the industrial computer to monitor the communication. Furthermore, we also use oscilloscope to sample the data from the serial port that connected to the MVB system bus. Both methods show that the master data frame and the feedback process data frame are sent correctly.

From the experiment results, it is reasonable to draw the conclusion that the Galsblock computation model supports the analysis and design of complex train control system better, compared to existing techniques. Right now, most companies such as Dugon and CNR develop the system by writing codes directly according to the descriptions of the IEC 61375 directly. Then, computer-aided simulation methods are used to validate the developed train control system [27], [8], [26], [31]. Simulation methods give accurate results when system failures occur. However, simulations are inefficient when applications are complex and the number of vehicles is large. Another drawback is that simulations are based on simulation patterns. The effectiveness of simulation depends on the number and quality of patterns. Hence, the exhaustiveness cannot be guaranteed. The traditional development process is hard working and the system reliability cannot be guaranteed. For example, we have found some deadlocks in the VHDL codes of CNR. But the GalsBlock computation model based development process overcomes those problems, with the simulation, verification, and generation toolkit.

6 Conclusion

In this paper, we present a synchronous and asynchronous cooperation computation model for embedded system design. The hierarchical blocks and the data port connections among these blocks present the decomposition structure and the signal flow of the system clearly. The parallel automata in the atom block controlled by different clocks describes the control logic and computation of each function. The implementation of GalsBlock is accomplished auto-

matically, the generated VHDL code can be synthesized and loaded into FPGA processor directly. We have developed the modeling, simulation, verification and code generation toolkit to support the GalsBlock computation model. The operational semantics based on the execution logic of real systems and the formal semantics based on the labelled transition system facilitate the designer to do some simulation and verification in the early stage of system engineering. This is efficient because coding with underlying programming languages such as VHDL and C according to the requirements directly is more difficult than building the abstract level GalsBlock model. Furthermore, simulation of VHDL and C programs needs more efforts to write test benches, and the verification is even harder. To the best of our knowledge, none of the existing methods supports the formal verification and implementation of embedded systems with both synchronous components and asynchronous components. The GalsBlock and the supporting toolkit will give a good guidance to reduce the complexity of the design of the complex embedded systems.

REFERENCES

[1] R. Alur. Timed automata. In *Computer Aided Verification*, pages 8–22. Springer, 1999.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[3] P. J. Ashenden. *The designer's guide to VHDL*, volume 3. Morgan Kaufmann, 2010.

[4] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *IEEE transaction on Computer*, 36(4):45–52, 2003.

[5] G. Berry. Circuit design and verication with esterel v7 and esterel studio. In *High Level Design Validation and Test Workshop, 2007. HLVDT 2007. IEEE International*, pages 133–136. IEEE, 2007.

[6] G. Berry, S. Ramesh, R. Shyamasundar, et al. Communicating reactive processes. In *Proceedings of the ACM SIGPLAN-SIGACT symposium on POPL*, volume 20, pages 85–98. ACM, 1993.

[7] G. Berry and E. Sentovich. Multiclock esterel. *Proceedings of the Correct Hardware Design and Verification Methods*, pages 110–125, 2001.

[8] J. Bolot. End-to-end packet delay and loss behavior in the internet. *ACM SIGCOMM Computer Communication Review*, 23(4):289–298, 1993.

[9] F. Boussinot and R. De Simone. The esterel language. *Proceedings of the IEEE*, 79(9):1293–1304, 1991.

[10] C. Brooks, E. A. Lee, and S. Tripakis. Exploring models of computation with ptolemy ii. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 331–332. IEEE, 2010.

[11] M. Bundgaard. Labelled transition system.

[12] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert. From simulink to scade/lustre to tta: a layered approach for distributed embedded applications. In *ACM Sigplan Notices*, volume 38, pages 153–162. ACM, 2003.

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

[14] H. Dong, B. Ning, B. Cai, and Z. Hou. Automatic train control system development and simulation for high-speed railways. *IEEE circuits and systems magazine*, 10(2):6–18, 2010.

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

[16] S. Edwards and O. Tardieu. Shim: A deterministic model for heterogeneous embedded systems. *IEEE Transactions on Very Large Scale Integration Systems*, 14(8):854–867, 2006.

[17] N. Halbwachs, F. Lagnier, and C. Ratel. Programming and verifying real-time systems by means of the synchronous data-flow language lustre. *Software Engineering, IEEE Transactions on*, 18(9):785–793, 1992.

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

[19] D. Harel. Statecharts: A visual formalism for complex systems. *IEEE Transactions on Software Engineering*, 8(3):231–274, 1987.

[20] F. He, L. Yin, and B.-Y. Wang. *VCS: A Verifier for Component-Based Systems*. Tsinghua University, 11th automated technology for verification and analysis edition, 10 2013.

[21] T. A. Henzinger and J. Sifakis. The embedded systems design challenge. In *FM 2006: Formal Methods*, pages 1–15. Springer, 2006.

[22] C. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[23] Y. Jiang, Z. Li, H. Zhang, Y. Deng, X. Song, M. Gu, and J. Sun. Design and optimization of multi-clocked embedded systems using formal technique. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 703–706. ACM, 2013.

[24] L. Ju, B. Huynh, S. Chakraborty, and A. Roychoudhury. Context-sensitive timing analysis of esterel programs. In *Proceeding of the 46th ACM/IEEE Design Automation Conference, 2009.*, pages 870–873, 2009.

[25] P. LeGuernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real-time applications with signal. *Proceedings of the IEEE*, 79(9):1321–1336, 1991.

[26] G. Palshikar. Safety checking in an automatic train operation system. *Information and Software Technology*, 43(5):325–338, 2001.

[27] K. Radecka and Z. Zilic. Design verification by test vectors and arithmetic transform universal test set. *Computers, IEEE Transactions on*, 53(5):628–640, 2012.

[28] S. Ramesh, S. Sonalkar, V. Dsilva, N. Chandra R, and B. Vijayalakshmi. A toolset for modelling and verification of gals systems. In *Proceedings of the International Conference on Computer Aided Verification*, pages 385–387. Springer, 2004.

[29] C. Schifers and G. Hans. Iec 61375-1 and uic 556-international standards for train communication. In *Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st*, volume 2, pages 1581–1585. IEEE, 2000.

[30] M. Shahdad. An overview of vhdl language and technology. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 320–326. IEEE Press, 1986.

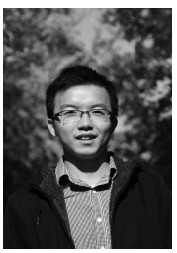
[31] R. Whitfield, W. Matheson, F. Ford, W. Basta, E. Peek, A. Guarino, B. Furtney, and C. Gipson. System and method for automatic train operation, Oct. 24 2000. US Patent 6,135,396.

[32] S. Wolfram. Theory and applications of cellular automata. 1986.

[33] I. Xilinx. Design suite, 2012.

[34] J. Yu, Z. Hehua, and J. Gu. Galsblock model for the mvb controller. Technical Report 19, Tsinghua University, 2 2014.

[35] B. P. Zeigler, H. Praehofer, T. G. Kim, et al. *Theory of modeling and simulation*, volume 19. John Wiley New York, 1976.



Yu Jiang received the BS degree in software engineering from Beijing University of post and telecommunication, beijing, China, in 2010. He is currently studying for the PhD degree in computer science from Tsinghua University, Beijing, China. His current research interests include domain specific modeling, formal verification and their applications in embedded systems.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2346171, IEEE Transactions on Parallel and Distributed Systems

Hehua Zhang received the BS and MS degree in computer science from Jilin University, Changchun, China, in 2001 and 2004, respectively. She received the PhD degrees in computer science from Tsinghua University, Beijing, China, in 2010. She is currently a lecturer in the School of Software at Tsinghua University. Her current research interests include domain specific modeling, formal verification and their applications in embedded systems.

Huafeng Zhang received the BS degree in software engineering from Xi'an Jiaotong University, Xi'an, China, in 2011. He is currently studying for the PhD degree in computer science from Tsinghua University, Beijing, China. His current research interests include domain specific modeling, formal verification and their applications in embedded systems.

Han Liu is a received the BS degree in computer science from Beijing University of post and telecommunication, beijing, China, in 2012. He is currently studying for the PhD degree in software engineering from Tsinghua University, Beijing, China. His current research interests include domain specific modeling, formal verification and their applications in embedded systems.

Xiaoyu Song received the PhD degree from the University of Pisa, Italy, 1991. In 1999, he joined the faculty at Portland State University. He is currently a professor in the Department of Electrical & Computer Engineering at Portland State University, Oregon. His current research interests include formal methods, design automation, embedded system design, and emerging technologies.

Ming Gu received the BS degree in computer science from the National University of Defence Technology, Changsha, China, in 1984, and the MS degree in computer science from the Chinese Academy of Science at Shengyang in 1986. Since 1993, she has been working as a professor in Tsinghua University. Her research interests include formal methods, middleware technology, and distributed applications.

Jiaguang Sun received the BS degree in automation science from Tsinghua University in 1970. He is currently a professor in Tsinghua University. He is dedicated in teaching and R&D activities in computer graphics, computer-aided design, formal verification of software, and system architecture. He is currently the director of the School of Information Science & Technology and the School of Software in Tsinghua University.