Received May 21, 2020, accepted May 29, 2020, date of publication June 2, 2020, date of current version June 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2999544

A Unified Model-Based Framework for the Simplified Execution of Static and Dynamic Assertion-Based Verification

MUHAMMAD WASEEM ANWAR[1], MUHAMMAD RASHID[2], (Member, IEEE), FAROOQUE AZAM[1], AAMIR NAEEM[1], MUHAMMAD KASHIF[3], AND WASI HAIDER BUTT[1]

[1]Department of Computer and Software Engineering, CEME, National University of Sciences and Technology (NUST), Islamabad 44000, Pakistan
[2]Computer Engineering Department, Umm Al-Qura University, Makkah 21955, Saudi Arabia
[3]College of Computer Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar

Corresponding author: Muhammad Waseem Anwar (waseemanwar@ceme.nust.edu.pk)

ABSTRACT The improved productivity and reduced time-to-market are essential requirements for the development of modern embedded systems and, therefore, the comprehensive as well as timely design verification is critical. Assertion Based Verification (ABV) is a renowned paradigm to timely achieve an optimum test coverage, either through static or dynamic techniques. However, the major limitation with ABV is its inherited low-level implementation complexity. In order to simplify its execution, various Model Based System Engineering approaches provide a higher abstraction layer. Nevertheless, the complete verification requirements, targeting the static as well as dynamic ABV at the same time in a unified framework, are not being addressed. Furthermore, the dynamic verification support is provided through some traditional languages (like C, Verilog) where the advanced ABV features cannot be exploited. Consequently, this article introduces the MODEVES (**MO**del-based **DE**sign **V**erification for **E**mbedded **S**ystems) framework to simultaneously support the static and dynamic ABV. Particularly, the UML (Unified Modeling Language) and SysML (Systems Modeling Language) diagrams are used to model the structural and behavioral requirements. Moreover, the NLCTL (Natural Language for Computation Tree Logic) is proposed to include the verification requirements for static ABV while the SVOCL (SystemVerilog in Object Constraint Language) is used to represent the dynamic verification constraints. An open source transformation engine is developed to automatically generate the SystemVerilog Register Transfer Level (RTL) code, Timed Automata model, SystemVerilog assertions and Computation Tree Logic (CTL) assertions with minimum transformation losses. The significance of the MODEVES framework is established through several case studies and the quantitative analysis shows an improvement of almost 100% in design productivity, as compared to the conventional low-level implementations.

INDEX TERMS Assertion based verification, computation tree logic, embedded systems, model based system engineering, systemverilog assertions, timed automata, unified modeling language (UML).

I. INTRODUCTION

The complexity and demand of embedded systems have increased exponentially. In order to manage the reduced time-to-market and improved productivity goals, the comprehensive design verification in an optimal time duration is critical [1]. The design verification of embedded systems is generally classified into static and dynamic verification categories [2]. The static verification techniques deal with the mathematical models to verify the correctness of system design [3]. On the other hand, the dynamic verification techniques are based on the simulation of Register Transfer Level (RTL) code for system validation. Although, the static verification techniques provide some sophisticated features, the issues like state explosion problem [4] and the reliable tool support restrict their applicability on large and complex

The associate editor coordinating the review of this manuscript and approving it for publication was Wen-Sheng Zhao.

designs. Similarly, an exhaustive testing is required for the dynamic verification process to achieve the maximum test coverage [5].

In order to address the limitations of conventional static and dynamic verification techniques, Assertion Based Verification (ABV) is employed which deals with the functional properties (assertions) of the system in a reduced simulation time [6], [7]. The two major types of ABV are static and dynamic [8]. In static ABV, the desired assertions are verified through formal methods. In this regard, Timed Automata [9] is a renowned formalism, particularly designed to validate the correctness of critical temporal aspects. On the other hand, the code of hardware design languages at RTL is simulated to perform dynamic ABV. However, in traditional hardware languages (Verilog, VHDL), ABV is not inherently supported [10], [11] as such languages mainly deal with the system design only. In this context, SystemVerilog [12] is an increasingly popular language that operates at RTL and provides dynamic ABV support through SystemVerilog Assertions (SVAs).

Despite the effectiveness of ABV in improving the design verification process, its low-level implementation complexities result in several verification delays [30]. In this regard, Model Based System Engineering (MBSE) plays an important role by providing a higher abstraction layer [2]. It first deals with the modeling phase to capture the design requirements and constraints at higher abstraction level (source models). Subsequently, a transformation phase is employed to automatically transform the source models into the target models at lower level of abstraction. Finally, the design verification (formal and/or dynamic) can be instantly performed through the automatically generated code. The Object Management Group (OMG) has introduced a standard Unified Modeling Language (UML) profile and its extensions like Systems Modeling Language (SysML) [20] to simplify the modeling phase.

A. LIMITATIONS OF EXISTING MBSE FRAMEWORKS

Several MBSE frameworks have already been proposed to overcome the low-level complexities of ABV (e.g. [8], [14], [30] etc.). However, the existing frameworks either deal with the static or dynamic verification at a time. On the other hand, the complexity of modern embedded systems is significantly increased due to a higher degree of heterogeneity in terms of both software and hardware components [39]. As a result, the complex designs usually require the application of both (static as well as dynamic) types collectively in a unified design environment, to achieve the maximum verification coverage in an optimum time [7].

In addition to the lack of a unified design environment, the existing frameworks do not usually include the verification properties in the actual design models which creates a gap between design and its verification. Consequently, there is a need for a unified model based framework, where the verification properties can be directly included in the design models, to simultaneously support the static and dynamic ABV. To the best of our knowledge (Section **V**), an MBSE framework to support both static and dynamic ABV, by means of a well-known Timed Automata formalism and SystemVerilog language respectively, is hard to find in the literature and industrial projects.

B. THE PROPOSED FRAMEWORK

In order to address the limitations of existing frameworks, this article has proposed the *MODEVES* (***MO****del-based* ***DE****sign* ***V****erification for* ***E****mbedded* ***S****ystems)* framework. The objective is to improve the design productivity (Section **IV-D**) by providing a higher abstraction layer for both static and dynamic ABV, where the design as well as verification aspects are modeled collectively. Particularly, the idea is to enable the accurate transformation of a single/unified design model into SystemVerilog and Timed Automata simultaneously. Furthermore, the verification properties included in the high-level models are also transformed into SVAs and CTL assertions.

The overview of MODEVES framework is shown in **Figure 1**. A complete end-to-end MODEVES Modeling Methodology (MMM) is introduced in Section **II** to model the structural, behavioral and verification (static as well as dynamic) requirements collectively. Particularly, a UML and SysML based approach is introduced to model the system design (Section **II-A**) i.e. Block Definition Diagram (BDD) concepts are used to represent the system structure and State Machine Diagram (SMD) concepts are employed to represent the system behavior. Moreover, the NLCTL (Natural Language for Computation Tree Logic) is proposed (Section **II-B**) to include the verification properties for static verification. Furthermore, the SVOCL [15] is integrated in the framework for dynamic verification constraints.

Once the modeling of design and verification aspects is performed, an open source MODEVES Transformation Engine (MTE) [19] is developed (Section **III)** to automatically generate the target low level codes. It includes the SystemVerilog RTL code, SVAs, Timed Automata model and CTL (Computational Tree Logic) assertions. Particularly, certain transformation rules are developed to perform a conceptual mapping between BDD/SMD constructs and SystemVerilog RTL/Timed Automata constructs in Section **III-B** and Section **III-C** respectively. Similarly, multiple rules are also developed to convert the NLCTL verification properties into CTL assertions while the transformation of SVOCL is integrated to generate SVAs (Section **III-D**). Subsequently, the implementation of transformation rules is performed in JAVA and Acceleo [16].

While the MODEVES framework proposes a unified design model for the automatic generation of SystemVerilog and Timed Automata, it advocates the use of two different formalism for the representation of static and dynamic properties. The reason is that a unified approach is only beneficial where a single requirement can be simultaneously and correctly transformed into the respective SVA and CTL properties. Therefore, the unified representation for

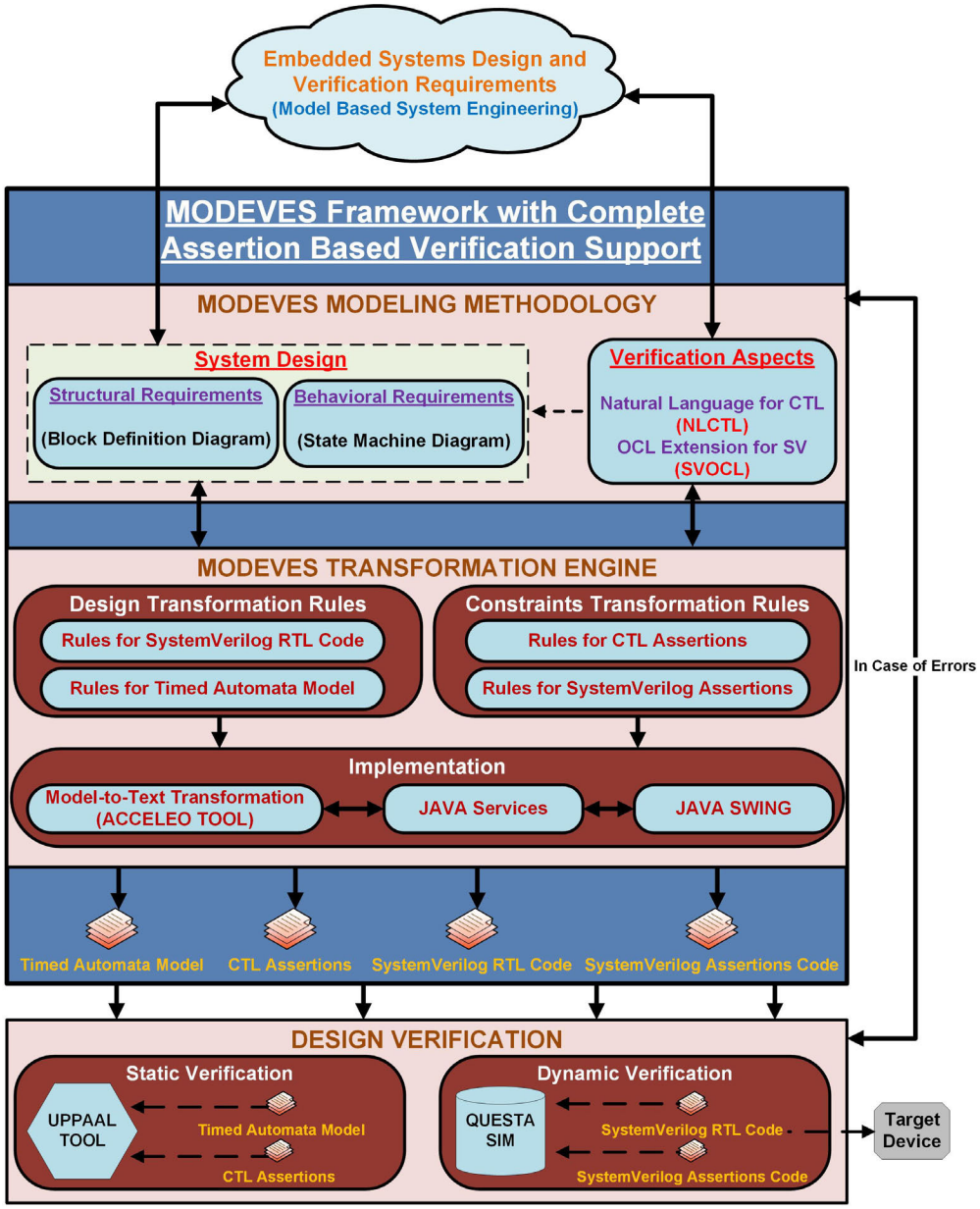


FIGURE 1. Overview of the MODEVES framework.

verification constraints is not feasible without compromising the critical transformation losses due to the significant syntax and semantic differences between CTL and SVAs.

C. VALIDATION AND QUANTITATIVE ANALYSIS

The validation of MODEVES framework is performed (Section **IV**) through eight case studies i.e. Traffic Lights Controller, Car Collison Avoidance System, Arbiter, Elevator, Unmanned Aerial Vehicle, Automated Teller Machine, Train Gate and Bridge Crossing system. The system design and verification aspects are modeled through the proposed modeling methodology and the required target codes are automatically generated through the developed transformation engine. Finally, the QuestaSIM simulator [17] and UPPAAL tool [18] are used to perform dynamic and static ABV respectively. After the successful design verification, the SystemVerilog RTL code can be deployed to the target device as shown in **Figure 1**. The quantitative analysis of MODEVES framework with respect to the native SystemVerilog and Timed Automata technologies shows 100% productivity gain (Section **IV-D**). It is important to note that this article only includes the details of three case studies due to space limitations while the models of all the eight case studies, along with the source code of transformation engine, are available at [19] for further evaluation.

TABLE 1. Notations for the modeling of system structure.

Sr. #	Name	Graphical Representation	Description
1	Flow Ports	«block» flow ports: inout flowport1: Boolean; in flowport2: Boolean; out flowport3: Boolean	The concept of flow ports is provided with three possible directions i.e. *in*, *out* and *inout*. The application of flow ports is proposed for the modeling of structure and linkage between various registers of the system.
2	Data Types	«datatype» DataType1 properties: Attribute1: Boolean; Attribute2: Integer	The primitive data types (Integer Boolean, Real etc.) are used to represent different variables of the system structure.
3	Enumeration	«enumeration» literals: ONE; TWO	The enumerations achieve a particular modeling requirement. A typical example is to represent all the states in a system.
4	Signal Event	«signal» Signal1	The clock and timer are managed through an activity diagram, provided as a built-in function of the framework [21]. The clock and timer are declared in a block with associated properties, and for their desired execution, the signal event is introduced.

D. SUMMARY OF NOVEL CONTRIBUTIONS

To summarize, the novel contributions of the proposed framework are as follows:

1) A unified model-based design approach, by utilizing the standard UML/SysML notations, is proposed. The high-level models contain the necessary information, required to concurrently transform them with minimum transformation losses.
2) A formalism (NLCTL) is proposed to include the properties in design models with simplicity for static ABV. On the other hand, the SVOCL [15] is used to include the verification properties for dynamic ABV.
3) The design and implementation of transformation rules to automatically generate the SystemVerilog RTL code, Timed Automata model, CTL properties and SVAs are proposed.

II. MODEVES MODELING METHODOLOGY

The modeling phase provides a higher abstraction layer for some particular low-level technologies [2]. Subsequently, the transformation is performed to automatically generate the target low-level codes. Therefore, it is essential to systematically include the necessary information of the target technologies in the modeling phase to perform accurate transformations. In the MODEVES framework, we are dealing with SystemVerilog, Timed Automata, SVAs and CTL technologies. Consequently, the objective of MODEVES Modeling Methodology (MMM) is to enable the modeling of general design and verification concepts at higher level while logically preserving the semantics of target technologies. This section briefly describes the details of MMM regarding design (Section **II-A**) and verification (Section **II-B**) requirements.

A. MODELING OF SYSTEM DESIGN

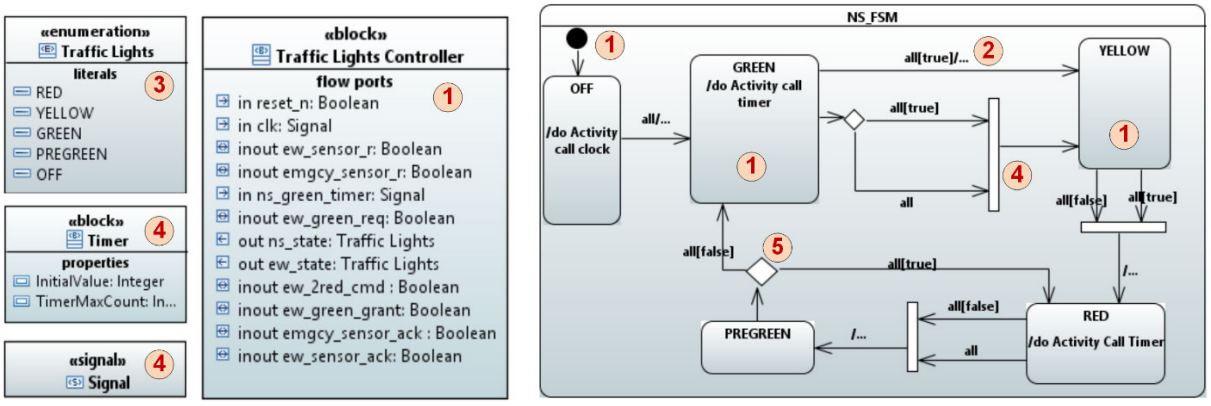
In MMM, we select the standard UML/SysML notations for the modeling of system design (structure and behavior). The structure of a system includes different variables and hardware elements like registers, ports etc. The BDD diagram in SysML, which has been extended from the UML class diagram, provides several modeling concepts like blocks, flow ports etc. for a realistic representation of the system structure. Therefore, the proposed MMM advocates the use of particular BDD concepts for the modeling of structural requirements, as given in **Table 1**. The first column "Sr. #" represents the *serial number* of a given modeling notation. The *names* and *graphical representations* are given in the second and third columns. Finally, the *description* is provided in the last column.

While it is relatively simple to model the structural elements of a system design process, the correct and meaningful modeling of the system behavior is relatively complex. Particularly, the behavior of embedded systems involves complex temporal aspects with several constraints while achieving the correct sequence of an execution flow. In this context, the SMD in SysML/UML provides several notations which are based on the principal of finite-state-transitions. Therefore, the SMD notations allows a systematic modeling of both simple as well as complex behavioral requirements, as given in **Table 2**.

For a better understating of MMM, consider a guiding example of Traffic Lights Controller (TLC). The functionality of TLC is to manage the road traffic at North-South (NS) and East-West (EW) roads intersection. There is a sensor attached at the EW road to detect the presence of a vehicle. The EW light should only be turned green to pass the traffic on the EW road if the EW sensor is activated. An emergency sensor is also deployed at the NS/EW intersection to manage the passage of emergency vehicles. To demonstrate

TABLE 2. Notations for the modeling of system behavior.

Sr. #	Name	Graphical Representation	Description
1	State	State1 InitialState FinalState	The SMD *state* node represents different system states. It specifies a particular action / condition, while entering or leaving the states, and is accomplished through *entry point* and *exit point* pseudostates. For complex requirements, the proposed MMM allows to call other behaviors, modeled in the SysML Activity Diagrams, inside a particular state. The clock and timer behaviors are the typical examples. The start and end points for a particular behavior are represented through SMD *initial* and *final* state nodes.
2	Transition	Transition <Trigger> Guard {?} Constraint1	The SMD transition notation dispatches some particular events through the *trigger* notation. Moreover, the *constraints* can be associated with transitions through the *guard* notation. To summarize, the transition is activated on a particular event and successfully executed on the fulfilment of given constraints.
3	Fork pseudostate		The *fork* pseudostate is used to split a single input transition into two or more output transitions for concurrent execution.
4	Join pseudostate		The *join* pseudostate combines two or more input transitions into a single output transition. Logically, it checks the combination of several input conditions (transitions) to produce a single output transition when all the input conditions become true.
5	Choice pseudostate		The use of *choice* pseudostate is suggested in order to achieve a conditional branching for particular modeling purposes.



(a) System Structure (b) System Behavior

FIGURE 2. Modeling the (a) Structure and (b) Behavior of Traffic Light Controller.

the effectiveness of MMM, the structure and behavior of TCL is modeled in **Figure 2 (a)** and **Figure 2 (b)** respectively where the circled numbers demonstrate the usage of respective UML/SysML notation in **Table 1** and **Table 2** respectively. For simplicity, we only consider the modeling of NS behavioral requirements rather than complete TLC system.

Figure 2 (a) shows that the registers, sensors and other elements are modeled through flow ports (Sr. 1 in **Table 1**), while different states are represented through enumeration (Sr. 3 in **Table 1**). Furthermore, the built-in clock and timer are declared with associated variables through block and signal events (Sr. 4 in **Table 1**). The modeling of TLC behavior (only NS) is shown in **Figure 2 (b)**. The different states like initial state, green, yellow etc. are modeled through state node (Sr. 1 in **Table 2**). Furthermore, several transitions (Sr. 2 in **Table 2**) are utilized to model the desired flow between different states by applying the associated trigger and guard conditions. However, the complete trigger and guard conditions of transitions cannot be represented graphically due to space limitations and only true/ false is displayed on each transition as shown in **Figure 2 (b)**.

In order to achieve certain behavioral requirements between the execution flows of different states, the pseudostate like join (Sr. 4 in **Table 2**) and choice (Sr. 5 in **Table 2**) are used. For example, assume there is a behavioral requirement that the TLC should only move from yellow to red state if NS timer for green state is equal to 3 and reset is

false. The both conditions (i.e. ns-timer = 3 and reset = false) are managed separately through two different transitions from the yellow state. Subsequently, if both conditions become true simultaneously, the join pseudostate combines these two transitions from the yellow state into a single one and take the system in the red state as shown in **Figure 2 (b)**. Similarly, other pseudostates like fork, choice are used as per given behavioral requirements.

To summarize, the MMM is capable of modeling both simple as well as complex system designs. It is important to note that we have only explained some of the modeling concepts here, through the TLC guiding example. However, the modeling and transformation capabilities of the MODEVES framework are comprehensively explained in Section **IV** through some complete case studies.Furthermore, the interested readers can find the detailed modeling guidelines with examples at [21].

B. MODELING OF VERIFICATION REQUIREMENTS

While the system design is based on a unified model (as shown in Section **II-A**), the system verification in the proposed framework requires two different formalisms: (1) the NLCTL is proposed to represent CTL properties in design models for static ABV, (2) the SVOCL is used to include SVAs for dynamic ABV. It is important to note that the SVOCL is already proposed in [15] and only its overview is provided here for the completeness of the framework.

1) NATURAL LANGUAGE FOR COMPUTATION TREE LOGIC (NLCTL)

The motivation behind the NLCTL is to provide a simple and logical modeling approach to include the CTL properties directly in the design models through a natural language alike syntax. The CTL [26] is a well-known temporal logic to specify the branching time constraints and deals with two temporal operators for paths quantifiers i.e. All and Exist. Furthermore, it also provides some temporal operators for the path specific quantifiers i.e. Next, Globally, Finally, Until and Weak Until.

The proposed framework directly loads the Timed Automata model and the CTL properties in UPPAAL tool to perform static ABV instantly [18]. In this context, a subset of standard CTL is utilized in UPPAAL to express the properties for the verification of Timed Automata model. Particularly, five temporal operators, based on the standard CTL concepts, are used. The Possibly operator deals with the reachability properties while the Invariantly and Potentially Always operators deal with the safety properties. Finally, the Eventually and Leads to operators deal with the liveness properties. In NLCTL, we consider all the aforementioned five operators for the modeling and transformation of both simple as well as complex CTL properties.

Proposed Formalism: We systematically develop NLCTL by utilizing the concepts of Extended Backus–Naur Form (EBNF) [27], which is a standard approach for the development of new languages e.g. Accellera Portable Test and Stimulus Standard (PSS) [28] etc. The grammar of NLCTL is defined using EBNF concepts through the following fourteen rules:

1. < Requirement>:: =<Property><Type>

The first rule states that the verification requirement/CTL assertion, represented as a <Requirement>, can be defined through ''Property'' and ''Type'' non-terminal symbols. The definition of Property and Type is given in Rule 2 and 14 respectively.

2. <Property> :: = <Exp>
| if <Exp> then <Exp>
| <Deadlock_Exp>
| <Exp> and <Exp>
| <Exp> or <Exp>

The second rule defines that the property can be stated through expression <Exp> or ''if then expression'' or ''deadlock expression'' <Deadlock_Exp>. Furthermore, the property can be defined through and/or expressions <Exp>. The definitions of expression <Exp> and deadlock expression <Deadlock_Exp> are given in Rule 4 & 3 respectively.

3. <Deadlock_Exp> ::= never deadlock
| system is deadlock free
| deadlock
| system has deadlock

The third rule defines the syntax of a deadlock expression with four terminal symbols i.e. never deadlock, system is deadlock free, deadlock and system has deadlock.

4. <Exp> :: = (<property>)∗
| never <Exp>
| <State_Exp>
| <Time_Exp>
| <Logical_Exp>
| <Deadlock_Exp>

The fourth rule implies that it is possible to use the property non terminal symbol (Rule 2) in the expression repeatedly i.e. zero to n times. Furthermore, the expression can be defined along with ''never'' terminal symbol. In addition, the expression can be defined through <State_Exp>, <Time_Exp>, <Logical_Exp> and <Deadlock_Exp >, as given in the Rule 5, 6, 8 & 3 respectively.

5. <State_Exp>:: = <Process_Name> state is <State_Name>
| <Process_Name> state is equal to <State_Name>

It defines the syntax of state expression <State_Exp>. The terminal symbol ''state is'' should be used between the process name and the state name. Similarly, the terminal symbol ''state is equal to'' can also be used between the process name and the state name non terminal symbols. The definitions of <Process_Name> and <State_Name> are given in Rule 13 & 12 respectively.

6. <Time_Exp>:: = <Variable_Name><OPT><Value>
| <Variable_Name><OPT><Variable_Name>

It defines the syntax of time expression <Time_Exp>. The time expression can be specified by using variable name <Variable_Name>, operator <OPT> and value <Value> non terminal symbols where the operator non terminal symbol should be used between the variable name and the value non terminal symbols. Similarly, time expression can also be defined by using the operator non terminal symbol between the two variable name non terminal symbols. The definitions of <Variable_Name>, <OPT> and <Value> non terminal symbols are given in Rule 11, 7 and 10 respectively.

7. <OPT> :: = less than
| greater than equal to

The syntax of operator <OPT> can be defined either through "less than" or "greater than equal to" terminal symbols.

8. <Logical_Exp> :: = <Variable_Name> <Logical_OPT><Value>

The logical expression <Logical_Exp> can be specified by using the variable name <Variable_Name>, logical operator <Logical_OPT > and value <Value> non terminal symbols where logical operator non terminal symbol should be used between the variable name and the value non terminal symbols. The definitions of <Variable_Name>, <Logical_OPT>, <Value> non terminal symbols are given in Rule 11, 9 and 10 respectively.

9. <Logical_OPT> :: = equal to
| less than
| greater than equal to
| less than equal to

The logical operator <Logical_OPT> can be defined in four different ways, as shown above.

10. <Value> :: = ([a-z][A-Z][0-9])∗
11. <Variable_Name> :: = ([a-z][A-Z][0-9])∗
12. <State_Name> :: = ([a-z][A-Z][0-9])∗
13. <Process_Name> :: = ([a-z][A-Z][0-9])∗

The rules 10, 11, 12 and 13 provide the definitions for declaring a value, variable name, state name and the process name non terminal symbols respectively. The value can be defined through the combination of 0-9 integers e.g. 12, 987 etc. Moreover, both small as well as capital alphabetical letters (i.e. a to z or A to Z) can be used in different combinations to define a particular value. Furthermore, an alpha numeric combination can also be used to define a value. Similarly, the variable, state and process names can be defined as per given aforementioned value rules.

14. <Type> :: = Possibly
| Invariantly
| Potentially always
| Eventually
| Leads to

There are three major types of CTL queries (i.e. Reachability, Safety, Liveness) for the verification of Timed Automata model in UPPAAL tool. To include all the four types of verification queries in NLCTL, type <Type> non terminal symbol is introduced. The type can be defined through five terminal symbols (Possibly, Invariantly, Potentially always, Eventually and Leads to).

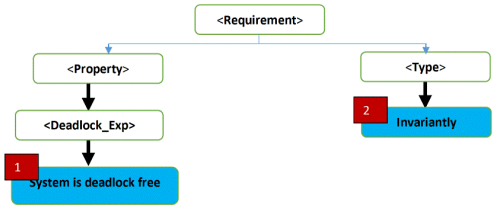


FIGURE 3. NLCTL Syntax Tree for Example 1.

Application of NLCTL: This section demonstrates the application of NLCTL, based on the proposed EBNF grammar, with examples. Particularly, the objective is to demonstrate the systematic working of NLCTL grammar through syntax tree.

Example 1: This example is taken from the TLC system where the requirement is to check a deadlock in the system. This property can be represented in NLCTL as:

Property = System is deadlock free, Type = Invariantly

The syntax tree of NLCTL grammar for a given property is shown in **Figure 3**. Please note that the terminal symbols are highlighted in blue color and the sequence of terminal symbols during parsing is shown in red color with a respective number.

Figure 3 shows that the given verification requirement is expressed through *Property* and *Type* non terminal symbols (rule 1 of the NLCTL grammar). The *Property* non terminal symbol can be directly represented through *deadlock expression* (rule 2). Finally, the deadlock expression can be represented through "system is deadlock free" terminal symbol (rule 3). On the other hand, *Type* non terminal symbol can be represented through "invariantly" symbol (rule 14). Consequently, the given requirement is correctly expressed through NLCTL grammar rules.

Example 2: This example is taken from an elevator system. Particularly, the requirement is to check the reachability of warning, overload and weight reduced states while the weight_sensor is under the allowed limit (i.e. 800 KG) and motion_sensor is activated. It can be expressed as:

Property = If (weight_sensor less than equal to 800 and motion_sensor equal to true) then never (P1 state is WarningState or P1 state is OverLoad or P1 state is WeightReduced), Type = Possibly

The syntax tree for Example 2, as shown in **Figure 4**, utilizes most of the NLCTL grammar rules. The given example is started through rule 1 by utilizing the property and type non terminal symbols. Subsequently, the property can be defined through "if <exp> then <exp>" as per rule 2 of the NLCTL grammar. Therefore, the rule 2 is applied to achieve the desired representation, as shown in **Figure 4**. According to rule 4, expression <exp> can be defined as a property recursively. Similarly, various NLCTL rules are utilized to correctly express the given requirements.

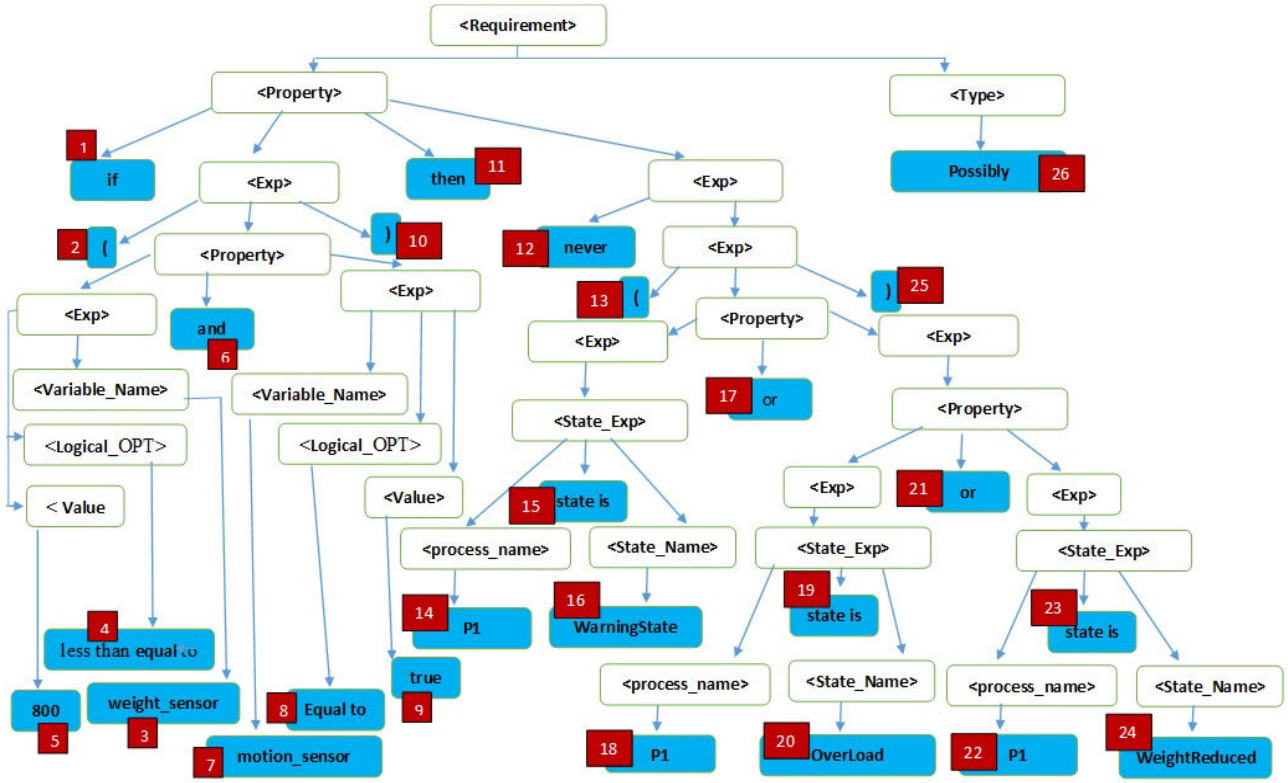


FIGURE 4. NLCTL Syntax Tree for Example 2.

To summarize, the NLCTL is a significant step to express the verification properties at a higher abstraction level. To complete the MBSE flow, its grammar is implemented in the transformation engine, so that, the high-level NLCTL constraints can be automatically transformed to the corresponding CTL assertions (See Section **III-D** for the details of transformation process). The user manual and further details of NLCTL can be found at [31].

2) SVOCL FOR SystemVerilog ASSERTIONS

We have already developed an OCL extension, named as SVOCL [15], to include the dynamic verification requirements in high-level models. In this regard, seven functions are included in SVOCL, as shown in **Table 3** (sr. # 1 to sr. # 7). Furthermore, Disif expression (sr. # 8 of **Table 3**) is incorporated for conditional statements. In the proposed framework, we have integrated the SVOCL for dynamic ABV. In this regard, the practical usage of SVOCL functions (**Table 3**), along with the complete technical details, can be found at [15].

III. MODEVES TRANSFORMATION ENGINE (MTE)

This section describes the MODEVES Transformation Engine (MTE). Firstly, the general architecture is described in Section **III-A**. In order to develop some appropriate transformation rules for system structure, the conceptual mapping between SysML BDD notations and SystemVerilog constructs is defined in Section **III-B**. For system behavior, the conceptual mapping of SystemVerilog and Timed Automata constructs with respect to SMD notations is described (Section **III-C**). Finally, the transformation rules to generate CTL assertions from NLCTL constraints are provided in Section **III-D**.

A. GENERAL ARCHITECTURE

We practice a Model-to-Text (M2T) transformation approach, using the open source Acceleo tool [16], for the implementation of MODEVES Transformation Engine (MTE). Acceleo is a well-known tool for a seamless execution of transformations in MBSE [29]. **Figure 5** shows that the implementation of MTE is accomplished through three main modules: (1) *User Interface*, (2) *Generator* and (3) *Java Services*. In the following, the brief description of each module and respective sub-modules is presented.

User Interface: It launches the application and displays the user interface of MTE on the screen. It takes the MMM compliant models in .UML extension and sends their address to *Generator* module for further processing (e.g. extraction of model elements etc.). Furthermore, it also sends the destination folder address (where the target models are required to be generated) to the *Generator* module.

Generator: It receives the input UML models and generates the target SystemVerilog RTL, Timed Automata model,

TABLE 3. Fundamental constructs of SVOCL.

Sr. #	Required Functionality for Timing Constraints at Higher Abstraction Level	Representation of Constraints with SVOCL
1.	Sequential delay between two expressions	SVSeq (sequential expr, sequential delay, preceding expr)
2.	Consecutive repetitions of an expression	SVRep (expr, numeric value of repetitions)
3.	Implications of behaviors/sequences/expressions	SVImplication (antecedent expr, consequent expr, implication type)
4.	Whether the value of an expression is stable	SVStable (expr)
5.	Whether the value of an expression has changed	SVChanged (expr)
6.	Whether the value of an expression is changed from False to True	SVRose (expr)
7.	Whether the value of an expression is changed from True to False	SVFell (expr)
8.	Customized conditional statements	Disif expression

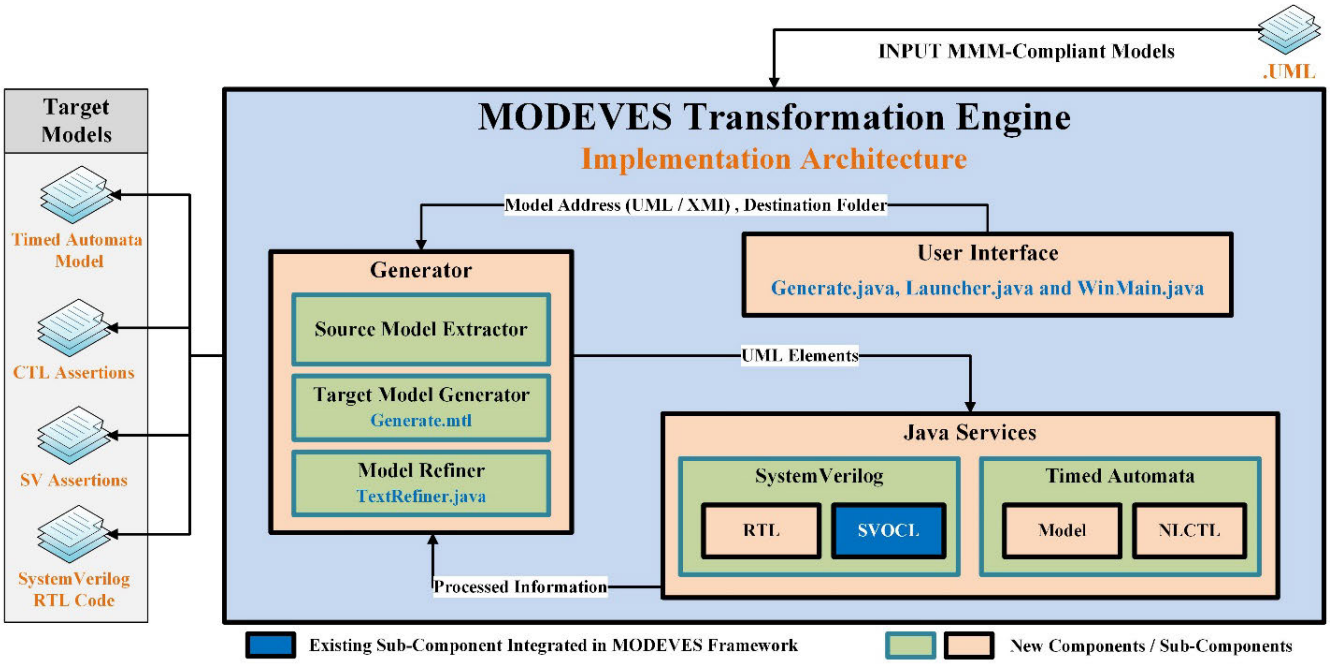


FIGURE 5. Implementation architecture of MODEVES transformation engine.

SVAs and CTL assertions. Particularly, the *source model extractor* extract the required UML elements (e.g. BDD attributes, state and transition elements etc.) from the input models which are then passed to *Java Services* module for further processing. Finally, the *target model generator* generates the target models after receiving the processed information from the *Java Services*. Lastly, the *model refiner* is responsible to manage the formatting issues in the generated target models.

Java Services: It implements the logic of transformation rules (Section **III-B**, Section **III-C** and Section **III-D**). Particularly, the *SystemVerilog* sub-module contains the corresponding transformation rules for the RTL code and SVOCL. It is important to note that the *SVOCL* sub-module is already implemented [15] and only the integration is performed here. On the other hand, the *Timed Automata* sub-module implements the transformation rules for the Timed Automata model and NLCTL assertions. The processed transformation rules from *Java services* component are sent back to the *Generator* module to ultimately generate the target codes, as shown in **Figure 5**.

It can be seen from **Figure 5** that the architecture of MTE is based on a modular approach as the coupling between different modules is as minimum as possible. Consequently, the MTE is highly supportive in terms of scalability and usability. For example, some additional CTL constructs can be added in the current version by only updating the NLCTL sub-component. Similarly, the other components can also be upgraded with simplicity according to some particular requirements.

In addition to the scalability, the architecture of MTE also provides the usability features as different components can be used in other implementations with slight modifications. For example, it is fairly possible to include the

TABLE 4. Conceptual mapping of smd notations w.r.t to systemverilog and timed automata concepts.

Sr. #	SMD Notations	Corresponding SystemVerilog Concepts	Corresponding Timed Automata Concepts
1	Name	Module Name	File Name
2	Region	Sub-section of code for a particular FSM	Timed Automata
3	Initial Node	Starting point for particular code	Initial Location
4	State	State in the context of code	Location
	Name	Name in code	Name
	Entry	Code specification when the system enters in a particular state	Invariant
	Exit	Code specification when the system exits from a particular state	Invariant
	Do Activity	Code logic for the activity which is called in a particular state	Time concepts
5	Transition	Code specification between different states	*Edge*
	Guard	Conditional code to move from one state to another state	Guard attribute of Edge
	Trigger	Code for ports that trigger in a particular transition	Sync attribute of Edge
	Effect	Code specification after the successful execution of transition	Update attribute of Edge
6	Fork	Concurrent execution for blocks of code	Location
7	Join	Conditional Statement by combining different expressions	Location
8	Choice	Conditional branching e.g. If / else, Case	Location

functionality of NLCTL in other frameworks with simplicity. The only requirement is to use the "GenerateCTL.java" file of NLCTL sub-component and a small portion of existing code is required from the Generator and Java Services components. To conclude, the further addition of functions in MTE is easy to implement and the integration of a particular MTE function in other frameworks is quite simple.

We do not include the complete MTE details here due to space limitations and further details like user/installation manual, the low-level architecture containing implementation details, the download link (source code) and the sample case studies can be found at [19]. However, the summary of MTE transformation rules is provided in subsequent sections.

B. TRANSFORMATION FOR SYSTEM STRUCTURE

The SysML BDD notations are proposed in Section **II** (**Table 1**) for the modeling of system structure. Here, the generic transformation rules are defined to generate SystemVerilog structural code from the BDD notations as follows:

1) Flow ports are transformed to SystemVerilog registers where *in* and *out* flow port types are mapped to SystemVerilog *input* and *output* registers/logic respectively. Furthermore, the flow port with *inout* direction type is mapped to SysemVerilog *wire*.

2) The UML/SysML primitive data types are transformed to equivalent SystemVerilog variable types like integer.

3) The UML/SysML enumeration type is transformed to equivalent SystemVerilog enumeration (*typedef enum*).

4) Although the activity diagram is used to implement the clock and timer, the signal type is actually used to represent it in the model. On the other hand, the SystemVerilog has inherited process for clocking. Therefore, only the definition of clock, on the basis of signal type and associated properties, is generated in the target structural code.

It is important to note that although we consider major SystemVerilog data types in the transformation process in order to define the structural aspects of system in RTL code, SystemVerilog supports different variations in standard datatypes e.g. integer datatype can be *signed* or *unsigned* etc. In transformation, we do not consider such variations in standard SystemVerilog datatypes and one can manually include such information in generated RTL code after transformation. Similarly, SystemVerilog also supports advanced datatypes like *chandle* to store pointers during Direct Programming Interface (DPI) [6]. We also do not consider such advanced datatypes in transformation.

C. TRANSFORMATION FOR SYSTEM BEHAVIOR

The behavioral models are simultaneously transformed into SystemVerilog RTL and Timed Automata model to perform both dynamic as well as static ABV. In this regard, it is essential to develop various rules to transform SMD notations into the corresponding SystemVerilog and Timed Automata concepts, as given in **Table 4**.

Table 4 shows that the mappings between SMD notations and Timed Automata are straightforward, as both are based on the principal of finite-state-transition. For example, the SMD transition is logically equivalent to Timed Automata edge. Therefore, the attributes of a transition like guard, trigger and effect can be directly mapped to the edge attributes like gurad, Sync and update respectively. Similarly, the SMD state is logically equivalent to Timed Automata location. Therefore, the state attributes like entry can be directly mapped to the location invariant. Moreover, the SMD Pseudostates like fork,

join and choice are logically equivalent to location in Timed Automata.

It is important to note that we target all the major elements of Timed Automata (i.e. initial location, location, committed location, edge with guard, update and sync attributes and invariant), which are defined in UPPAAL tool. However, few customized features of UPPAAL tool like urgent channel/location are not considered in the proposed transformation. Similarly, few variables are needed to be declared in UPPAAL tool in order to perform simulation. However, the information of global declaration variables cannot be incorporated systematically without compromising the simplicity and generic applicability of the framework. One possibility is to include the information of global declaration variables through UML comments; however, this is a non-standard way and of least use for researchers and practitioners. Therefore, we do not provide the modeling and transformation provision for global declaration variables as this information can easily be included in the automatically generated Timed Automata model manually.

On the other hand, it can be seen from **Table 4** that the mappings between SMD notations and SystemVerilog are indirect as the former is based on finite-state-transition concepts while the latter is a low-level hardware language. For example, the entry attribute of a state refers to the specific condition while the system is entering into a particular state. In SystemVerilog, this refers to a low-level code for a specific condition in the context of a particular state. Therefore, the flow of states and the transitions along with the respective attributes in the model is considered for the transformation of respective SystemVerilog RTL code. For example, the sequence of states belongs to the major (outer) structure of SystemVerilog RTL code while the sequence of transitions belongs to the inner code within the main structure. In this regard, the SMD region attribute is used to provide further division in the main structure i.e. a separate code block in RTL code for each region.

Similarly, the initial node provides a starting point for a particular block of code. Consequently, in order to generate the RTL code for the main structure (based on states) and the inner logic (based on transitions), we consider certain low-level SystemVerilog constructs in the transformation process.

For example, we consider certain procedural statements (i.e. *initial, final, always and function*) during the transformation process to generate the main structure of behavioral code. Moreover, the conditional branching (based on SMD Choice Pseudostate) is achieved in transformation process through SystemVerilog *if / else* and *case* statements. Similarly, we consider SystemVerilog *for* and *foreach* loops in the transformation process. To summarize, all the significant SystemVerilog constructs have been considered in the transformation process which are essential to specify both simple as well as complex design and verification requirements in RTL code and SVAs respectively. Furthermore, in case of any particular requirement, the generated RTL and assertions code can be optimized manually. In addition, it is also important to note that SystemVerilog is a complete design and verification language with several advanced features [33]. Therefore, it is not possible to consider all the aspects of SystemVerilog in the proposed framework. For example, SystemVerilog fully supports the object oriented concepts [33] like inheritance, polymorphism etc. to develop some complex test benches. Similarly, it can interact with other hardware languages through DPI features [6]. We do not consider the modeling and transformation of such advanced SystemVerilog features as these concepts are irrelevant in the given research context.

In **Table 4**, the major details regarding the conceptual mapping of SystemVerilog and Timed Automata with respect to SMD constructs are provided, without going into the low-level implementation details. It is important to mention here that the OCL basic operators like arithmetic, comparison etc. are logically equivalent to the SystemVerilog operators [34]. Consequently, it is straightforward to convert the OCL expressions, given in the SMD attributes like guard, effect etc., into the corresponding SystemVerilog code. On the other hand, the OCL basic operators are also available in Timed Automata. We are not including the low-level implementation details of Timed Automata operators and concepts like locations, edges etc. because such details can be found at [9]. Furthermore, the source code of MTE is publically available [19], where all the low-level details are available with proper comments.

D. TRANSFORMATION FOR NLCTL

The proposed NLCTL grammar is implemented, to accurately generate the corresponding CTL assertions. For example, the NLCTL rule 1 states that the requirement can be expressed through *Property* and *Type* terminal symbols. In MTE, the *Requirement* is transformed to *property* attribute in CTL. Moreover, the NLCTL *Property* terminal symbol is transformed to *Query attribute* in CTL. Furthermore, the *Type* terminal symbol of NLCTL is transformed to *Query Operator* in CTL.

Particularly, the *Type* terminal symbol is ultimately represented through five nonterminal symbols in NLCTL as per rule 14. To implement rule 14 in MTE, the NLCTL Type nonterminal symbols *"Possibly, Invariantly, Potentially always, Eventually* and *Leads to"* are transformed into the CTL operators "$E<>$, $A[]$, $E[]$, $A<>$ and $\rightarrow$" respectively. Similarly, all the proposed NLCTL grammar rules are implemented. Consequently, the MTE is able to accurately generate CLT properties from NLCTL assertions.

For a better understanding of MTE capabilities, consider the transformation of TLC guiding example (introduced in Section **II**) as given in **Figure 6**. Particularly, the MMM-compliant model of TLC is given to MTE which automatically generates four files i.e. SystemVerilog RTL, SVAs, timed automata and CTL properties. As 'SV' is a standard extension for SystemVerilog, the dynamic ABV can be performed instantly in existing Universal Verification

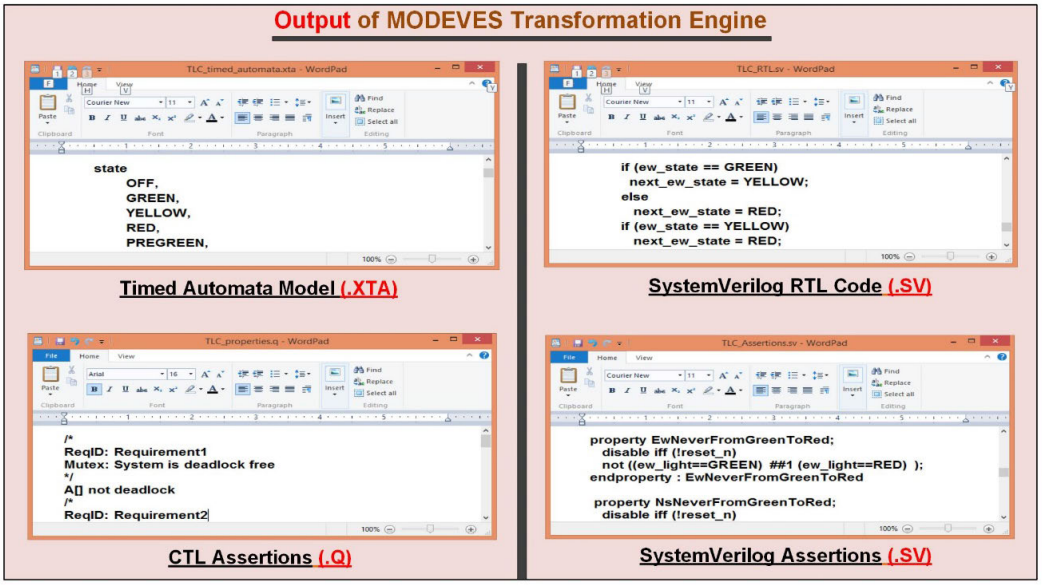


FIGURE 6. Transformation of traffic lights controller guiding example.

Methodology (UVM) [38] based simulators like QuestaSIM. On the other hand, the static ABV can also be performed instantly in UPPAAL tool as it fully recognizes the timed automata model and CTL properties through 'XTA' and 'Q' formats respectively.

IV. VALIDATION

The applicability of MODEVES framework is demonstrated through eight benchmark case studies i.e. Traffic Lights Controller (TLC), Car Collison Avoidance System (CCAS), Arbiter, Elevator, Unmanned Aerial Vehicle (UAV), Automated Teller Machine (ATM), Train Gate and Bridge Crossing system. However, this article includes the details of only three case studies (i.e. Arbiter, Elevator and Unmanned Aerial Vehicle) due to space limitations. In this regard, the models of all the eight case studies, along with the source code of MTE, are available at [19] for further evaluation. The MMM is applied to capture the structural, behavioral and verification requirements (Section **IV-A**, Section **IV-B** and Section **IV-C**), using the Papyrus modeling editor [35]. Subsequently, the MTE is utilized to generate the target code.

A. ARBITER CASE STUDY

This case study represents the design of an arbiter that is implemented as one-hot coding style state machine. It has seven possible states i.e. MASTER1, MASTER2, MASTER3, IDLE, IDLE1, IDLE2 and IDLE3. Any or all of the three master devices can make a request for the grant of the bus and the arbiter will select who gets the bus by using the round robin policy. The master device will be able to perform certain transactions after acquiring the bus. The bus is available for other requests after the completion of transaction.

1) REQUIREMENTS SPECIFICATION

Figure 7 shows the structural requirements of arbiter, captured through the MMM guidelines. The arbiter states have been represented through enumeration. We identify and capture the following behavioral requirements:

- State machine goes to the IDLE state on system reset.
- In the IDLE state, any or all the masters can send a request to acquire the bus. The decision of bus grant is taken in a priority-encoding fashion. For example, the bus will be granted to the MASTER1 device if all the master devices request for the bus in the IDLE state. In the MASTER state, the grant signal will be asserted to the respective master device. The master device will free the bus by sending the done signal and consequently the system moves to the IDLE state of the corresponding master device (e.g. IDLE1).
- When the system is in IDLE1 state, it prioritizes the assignment of a bus to the master devices in the order of MASTER2, MASTER3 and then MASTER1. Similarly, the priority of bus assignment is MASTER3, MASTER1 and then MASTER2 if the system is in IDLE2 state. Finally for IDLE3 state, the priority of bus assignment is MASTER1, MASTER2 and then MASTER3.

The behavior of arbiter is modeled and verification aspects are included in the models through both SVOCL as well as NLCTL, as shown in **Figure 8**. The following verification aspects have been identified and included in the arbiter model:

1. A grant should be asserted for every request received by the arbiter. The "gnt" signal should be asserted within 1 clock cycle for any one of the requesting masters.

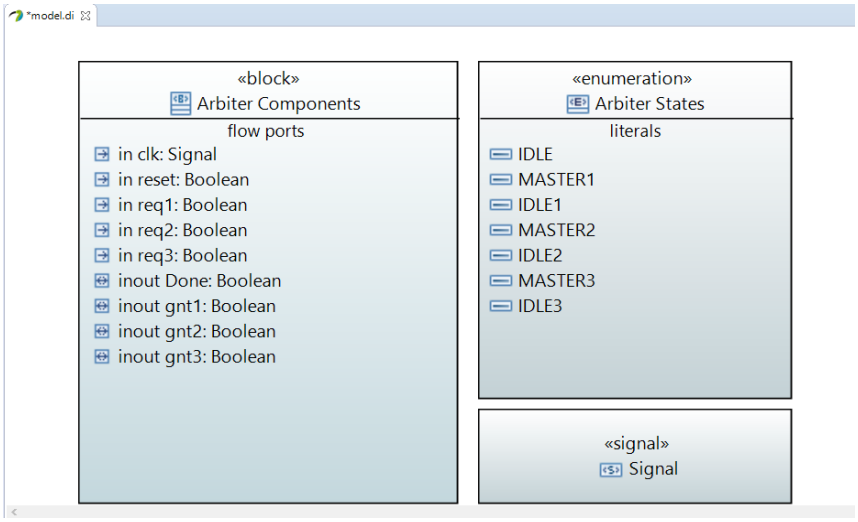


FIGURE 7. Structural requirements of arbiter through MMM.

2. The arbiter IDLE state should be working correctly i.e. it cannot go to IDLE 1, IDLE 2 and IDLE 3 from the IDLE state.
3. The arbiter master states should work as: 1) arbiter cannot go to IDLE2, IDLE3, MASTER2 and MASTER3 states from MASTER1 state 2) arbiter cannot go to IDLE1, IDLE3, MASTER1 and MASTER3 states from MASTER2 state 3) arbiter cannot go to IDLE1, IDLE2, MASTER1 and MASTER2 states from MASTER3 state.
4. Arbiter idle states should work as: (1) arbiter cannot go to IDLE2 and IDLE3 states from IDLE1 state. (2) arbiter cannot go to IDLE1 and IDLE3 states from IDLE2 state. (3) arbiter cannot go to IDLE1 and IDLE2 states from IDLE3 state.
5. Ensure the fairness of arbiter i.e. the grants should be assigned to masters on equal basis.

As design and verification requirements are modeled altogether (**Figure 8**), this provides the basis to generate both SVAs and CTL assertions along with SystemVerilog RTL code and Timed Automata model through MTE. It is important to note that we differentiate SVOCL and NLCTL assertions through Requirement key word of NLCTL, as shown in **Figure 8**. The next step is to utilize the MTE to transform the source models into the target models, as shown in **Figure 9**.

The MTE generates model_asserts.sv, model_behavior.sv, model_timed_automata.xta and model_properties.q files for SVAs, SystemVerilog RTL code, Timed Automata model and CTL assertions respectively. It is not possible to describe the complete MTE details due to space limitations. However, we upload MTE and arbiter case study here [19], so that, the interested readers can perform further evaluation.

2) DESIGN VERIFICATION

After the automatic generation of target models through MTE, both static as well as dynamic ABV can be performed in the respective tools. Here, we use QuestaSIM to perform dynamic ABV by utilizing the SystemVerilog RTL and assertions files which are generated through MTE. During the design varication (simulation), we encounter failure of an assertion. We analyze the cause of failure by utilizing the advance features of QuestaSIM, as shown in **Figure 10**. We investigated the cause of this failure and correct the design accordingly. Finally, after rigorous simulation, it has been analyzed that the arbiter design is free of errors. Here, we include the summary of design verification, however, the complete details can be found at [36].

In addition to dynamic ABV, we also perform static ABV by using open source UPPAAL tool. Particularly, we use Timed Automata model and CTL assertions files that are generated through MTE. We verify different properties like deadlock etc. as included in the model through NLCTL. The verification of one property in UPPAAL tool is shown in **Figure 11**. Further details regarding the static ABV of arbiter can be found at [36].

After successful verification, we employ Xilinx Vivado [37] to perform the code synthesis of SystemVerilog RTL code. However, we are not including the details of dynamic ABV (QuestaSim) and code synthesis (Xilinx Vivado) here due to space limitations. The interested readers can find the complete design verification and code synthesis details at [36].

B. ELEVATOR CASE STUDY

The case study represents the design of an elevator which is used to move the people and goods between different floors of a building. A weight sensor is installed to calculate the overall load. The maximum allowed wait is 800 kg. The panel is attached in the elevator to select the desired floors in a sequence as per requirements. The emergency sensor is also installed to instantly stop and open exit passage of the moving elevator in the case of emergency situations.

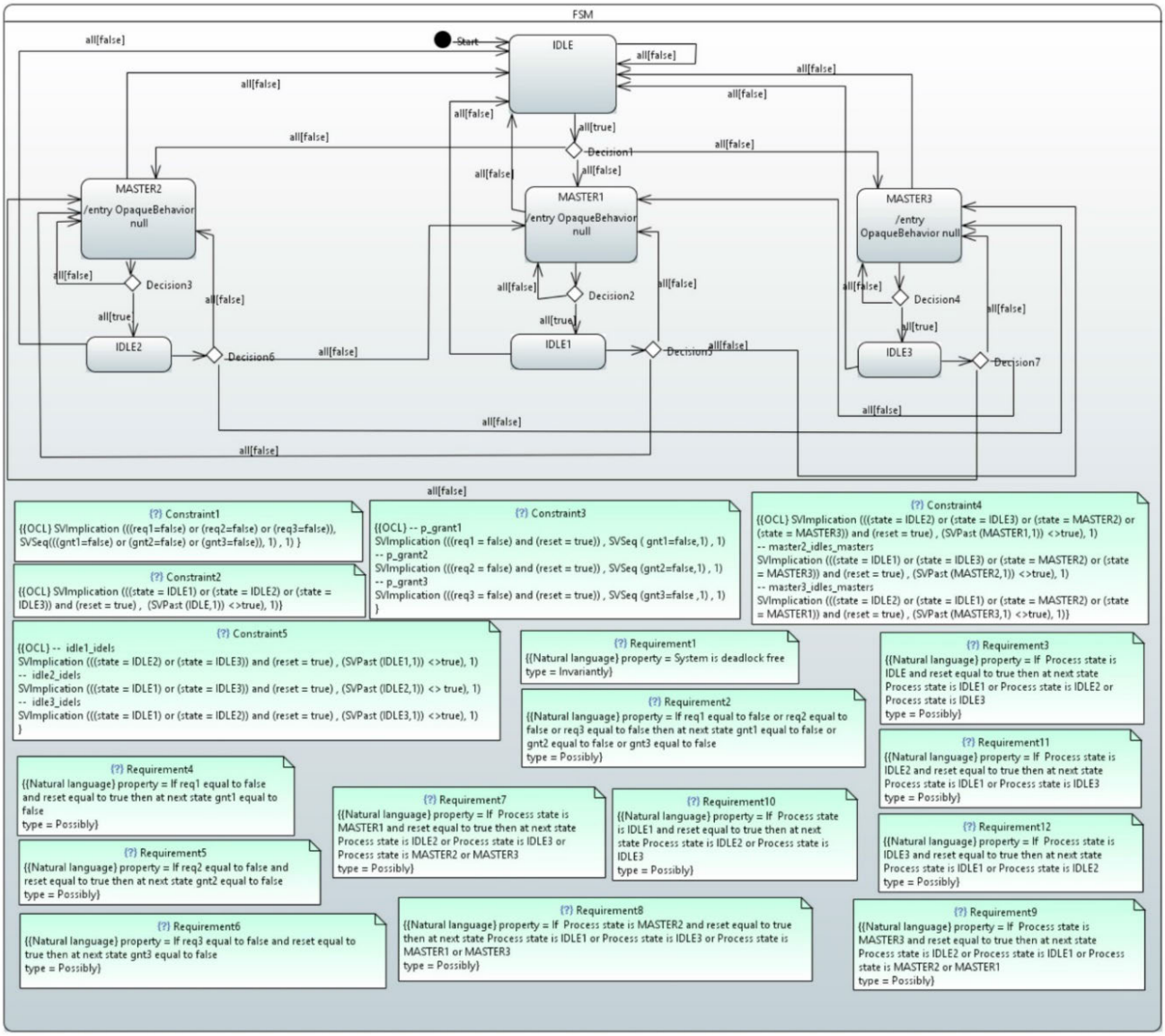


FIGURE 8. Modeling behavioral and verification requirements of arbiter through MMM.

1) REQUIREMENTS SPECIFICATION

The structure of elevator is modeled through the MMM guidelines, as shown in **Figure 12**. The following behavioral requirements of elevator are required:

- Initially, elevator should be in the IDLE state. Once persons are entered into an elevator, the overall weight is calculated. Subsequently, if the weight of elevator is greater than 800 than it is moved to the IDLE state by producing weight overload alarm. Otherwise, elevator should move further to check the desired floor number.
- The elevator should move upward or downward depending upon desired floor number and current floor of elevator.
- Finally, the elevator should reach to desired floor and subsequently move to initial state (IDLE).

The following verifications aspects of elevator are identified:

1. Confirm that whenever an elevator is moving, it should not be in the IDLE state.
2. Confirm the correct operating mechanism of elevator i.e. It shouldn't move in the upward and downward direction at the same time.
3. Confirm that when the elevator is moving either upward or downward, the door of elevator should not be opened.
4. Confirm that the overall weight of the elevator should be within the given limits (less than 800 KG).
5. Confirm that whenever the emergency sensor gets activated, the emergency exit should also be activated at the same time.

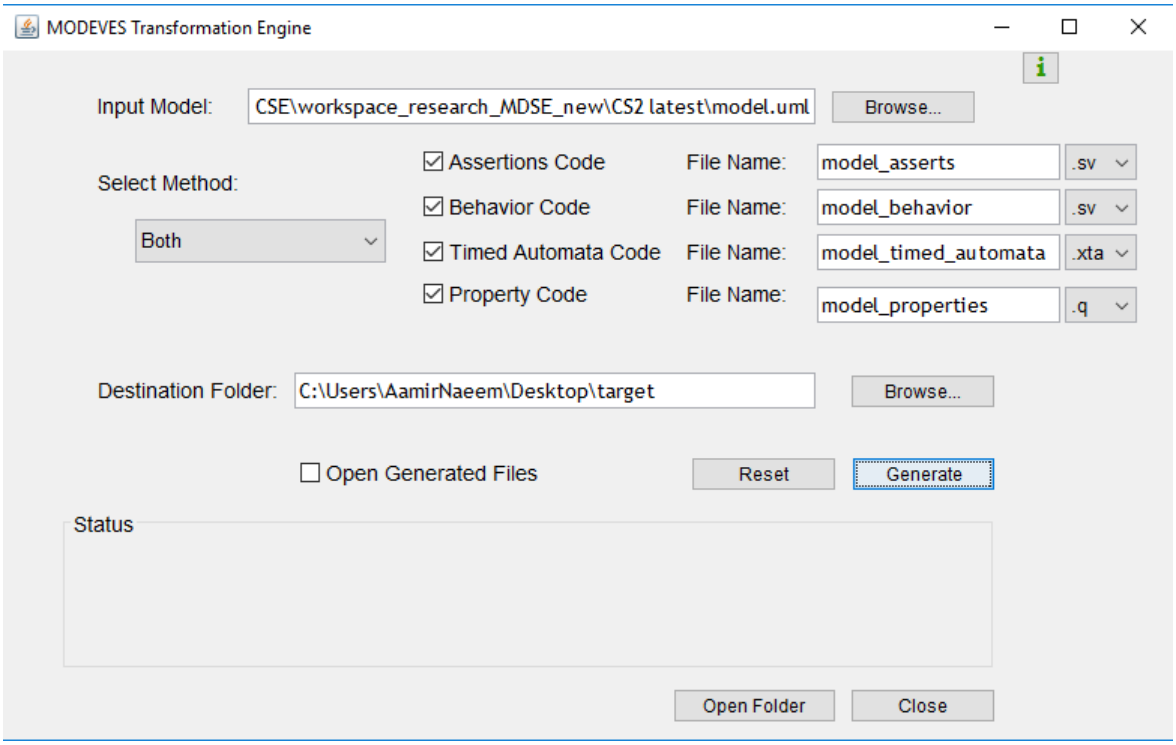


FIGURE 9. Generating target files for the arbiter model through MTE.

The aforementioned verification requirements are included in the behavioral model of elevator by utilizing both SVOCL and NLCTL approaches as shown in **Figure 13**. Once the design and verification requirements have been successfully modeled (**Figure 12** and **Figure 13**), we generate SystemVerilog RTL, Timed Automata model, SystemVerilog Assertions and CTL assertions through MTE. However, we are not including the details of code generation here because such details are already given in Section **IV-A-2**.

2) DESIGN VERIFICATION

We have performed both static as well as dynamic ABV for elevator system by utilizing the generated target files i.e. SystemVerilog RTL, Timed Automata model, SystemVerilog Assertions and CTL assertions. Firstly, we perform static ABV by utilizing UPPAAL tool. Particularly, the Timed Automata model and CTL assertions file, generated through MTE, are given to UPPAAL tool. Subsequently, design verification is performed through UPPAAL tool as shown in **Figure 14**. Once, all the given properties are successfully verified through UPPAAL, we perform dynamic ABV. Particularly, SystemVerilog RTL and assertions files are given as an input to QuestaSIM simulator. Subsequently, the simulation is performed to verify the given properties.

C. UNMANNED AERIAL VEHICLE (UAV) SYSTEM

The UAV is typically an aircraft that can be controlled remotely or fly autonomously on the basis of pre-programmed flight plans. It has quite complex dynamic automation systems. This case study demonstrates the design of UAVs with various safety constraints.

1) REQUIREMENTS SPECIFICATION AND DESIGN VERIFICATION

The structure of UAV system comprises one major block that contains required ports/registers. Furthermore, various UAV states are represented through enumeration, as shown in **Figure 15**. The following behavioral requirements are identified:

- Initially, the system is in FLYING state and continuously monitoring the engine failure and GPS failure states. The system moves to either ENGINE FAILURE state or GPS FAILURE state depending on the sensor values (engine_failure_sensor and gps_failure_sensor). Similarly, the system is also monitoring TERMINATION COMMAND RECEIVED, 2.4GHz LINK FAILURE, SOFT GEOFENCE BREACH and DATALINK FAILURE states through the termination_command_recieved, 2.5GHz_link_failue_senor, geo_fencing_sensor and datalink_failure_sensor respectively.
- Once the system is in ENGINE FAILURE state, it has to be landed on emergency basis within 3 clock cycles. In the TERMINATION COMMAND RECEIVED state, the system should move to the FLIGHT TERMINATION INITIATED state after one clock cycle and

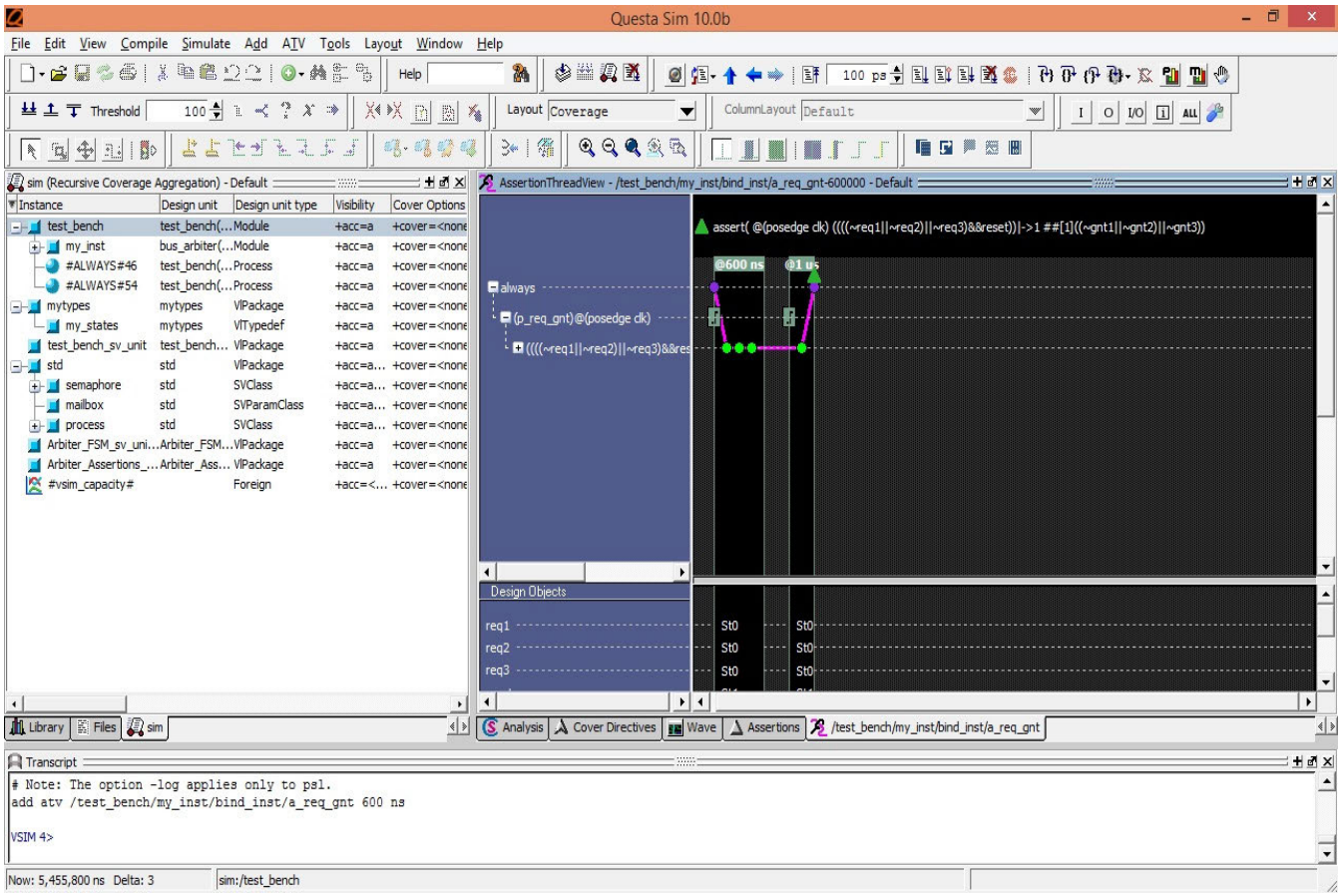


FIGURE 10. **Performing design verification of arbiter in QuestaSIM.**

subsequently move to the MANUALLY LAND AIRCRAFT state within three clock cycles.

- Once the system is in SOFT GEOFENCE BREACH state, it should move to RESTORING FROM NO FLY ZONE state on the next clock cycle and perform certain checks to move into respective state. Similarly, in the DATA LINK FAILURE state, the system evaluates certain conditions (e.g. 900MHz_link_failure_sensor, 5.8GHz_link_failure_sensor etc.) to move into a particular state accordingly.
- In case of GPS failure, the system continues normal flight on a successful GPS auto restore. Otherwise, the system should move to the FLIGHT BACK TO STATION state and subsequently moves to either REACHED BACK TO STATION state or AIRCRAFT LOST state, depending on the underlying conditions.

Following safety constraints are identified (**Figure 16**):

1. Property1: In case of termination command during the flight, the system should move to FlightTerminationInitiated within two clock cycles and subsequently enters into ManuallyLandAirCraft state within two to four clock cycles.
2. Property2: From AutoRestoring state, the system should move to Flying state in 4 clock cycles on the activation of auto_restore and ground_station_restore.
3. Property3: From RestoringFromNoFlyZone, the system should move to CheckFlyZone within 1 to 3 clock cycles. However, on the deactivation of safe_zone, the system should move to FlightTerminationInitiated state within 3 clock cycles and subsequently to ManuallyLandAirCraft state within 2 to 4 clock cycles.
4. Property4: The system should move from RestoringFromNoFlyZone to CheckFlyZone within 1 to 3 clock cycles and upon the activation of safe_zone, the system should move to the Flying state after 3 clock cycles.
5. Property5: In case of a GPS failure with an unsuccessful automatic restore, the system should go back to station within 4 clock cycles.

In arbiter and elevator case studies, we represent the verification aspects through both SVOCL as well as NLCTL to demonstrate the simultaneous application of both approaches. Now, we prove the application of SVOCL and NLCTL individually. Particularly, the idea is to show that SVOCL and NLCTL can be used separately through the MODEVES framework as per requirements. Here, we include the

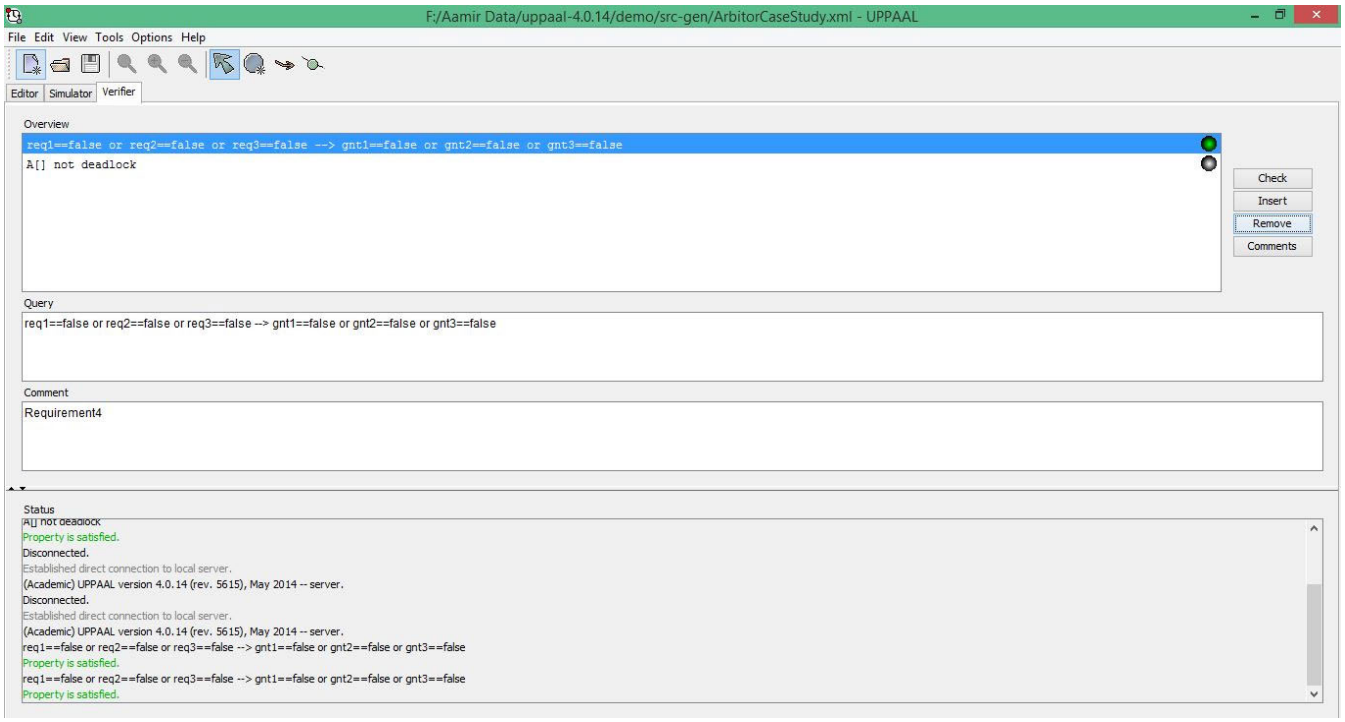


FIGURE 11. Performing design verification of Arbiter in UPPAAL.

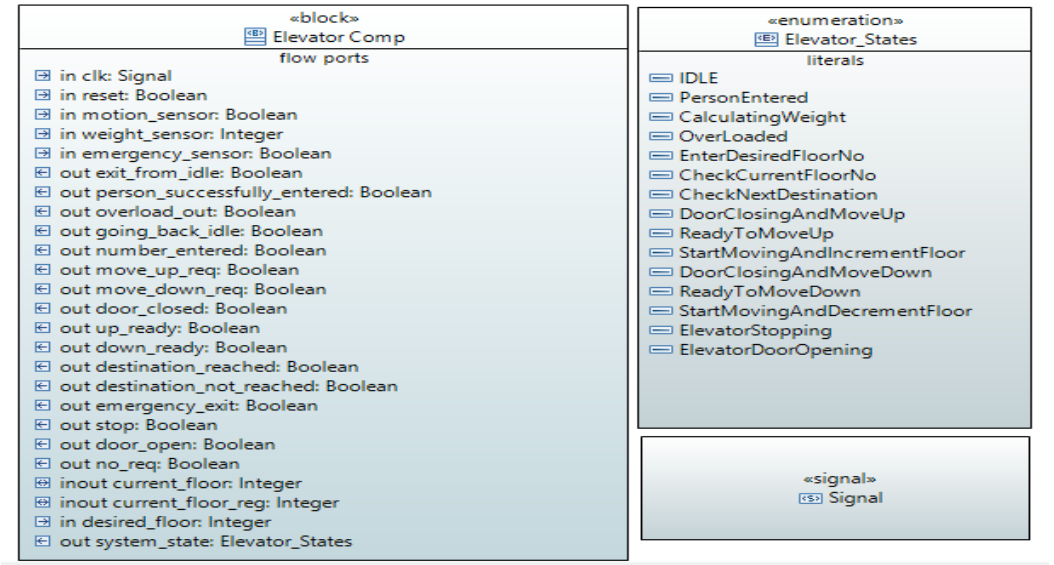


FIGURE 12. Structure of the elevator in BDD.

verification aspects of UAV system in the model through SVOCL only, as shown in **Figure 16**. Subsequently, only the SystemVerilog RTL and assertions files are generated. Once the design and verification requirements have been successfully modeled (**Figure 15** &**Figure 16**), we utilize MTE to generate the target code for design verification. Due to space limitations, we are not including the MTE and design verification details and the interested readers can found a complete model of UAV case study along with the MTE at [19].

In this section, we applied MMM on three industrial case studies to successfully model the structural, behavioral and verification requirements. Furthermore, we demonstrate the applicability of MTE by correctly transforming the models of two case studies into SystemVerilog RTL, Timed Automata model, SystemVerilog Assertions and CTL assertions.

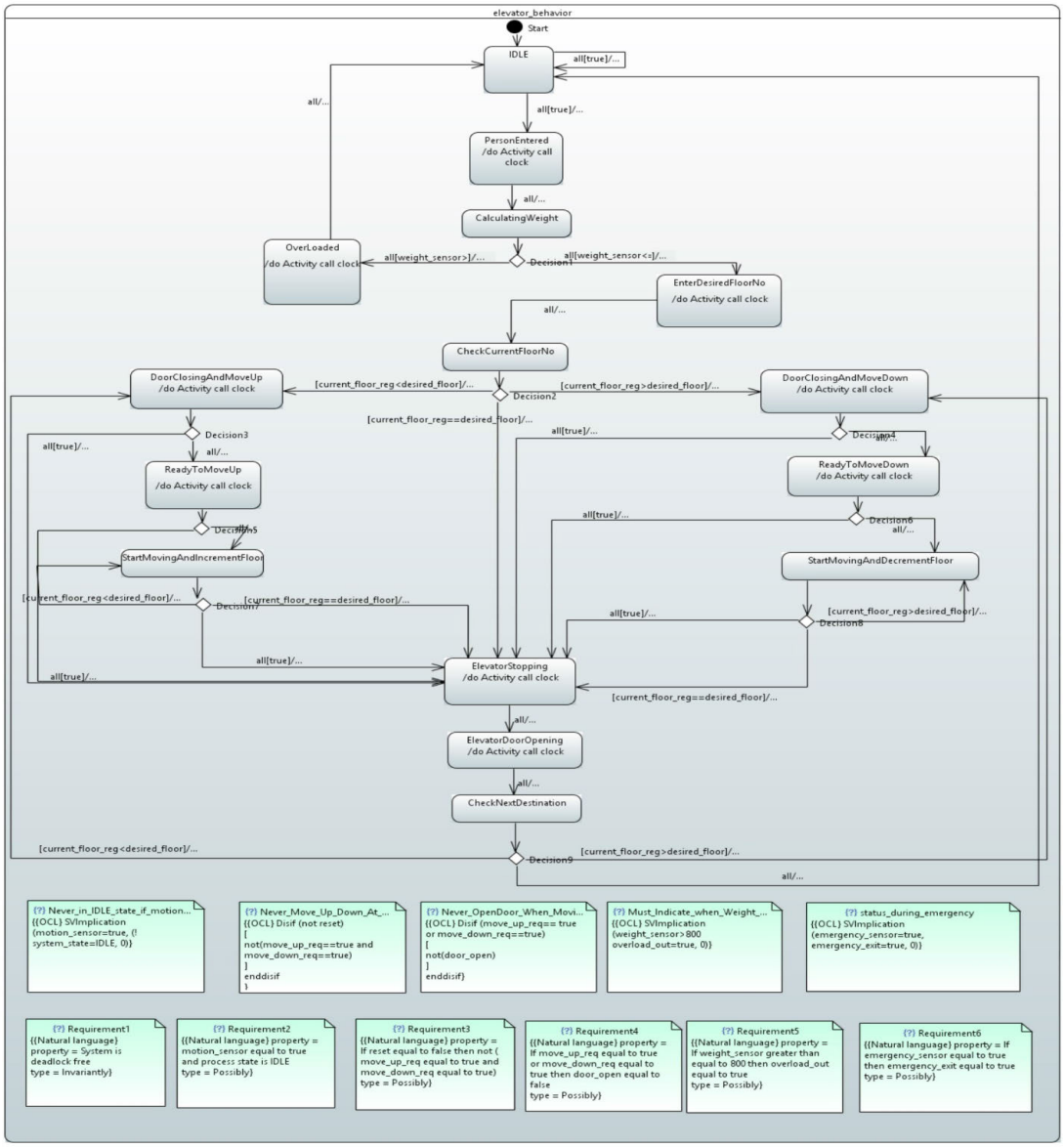


FIGURE 13. Modeling behavioral and verification requirements of elevator through MMM.

Consequently, it can be concluded that the MODEVES framework is widely applicable on a variety of embedded systems. It is important to note that MODEVES framework is validated through eight benchmark case studies. The models of all eight case studies, MTE source code and user manual can be found at [19] for further evaluation. Furthermore, the complete modeling guidelines of MODEVES framework can be found at [21], so that, the interested readers can model and transform any case study of their choice.

D. QUANTITATIVE ANALYSIS OF THE MODEVES FRAMEWORK

To this point, we have demonstrated the modeling and transformation capabilities of the MODEVES framework in Section **IV-A**, Section **IV-B** and Section **IV-C**. However, the following important questions are difficult to answer without performing a systematic quantitative analysis:

Q1: Why there is a need to propose UML/SysML based modeling approach for the system design that can support formal and dynamic ABV at the same time?

Q2: What is the improvement in design productivity through the proposed framework?

Q3: Does the proposed methods lead to fewer design mistakes and/or easier to correct the corresponding design errors?

Q4: Do the benefits of the proposed methodology outweigh by any transformation losses?

Q5: How easy is to learn the proposed framework, as compared to the low-level technologies?

Without answering the aforementioned questions, the novelty, contributions and benefits of the MODEVES framework

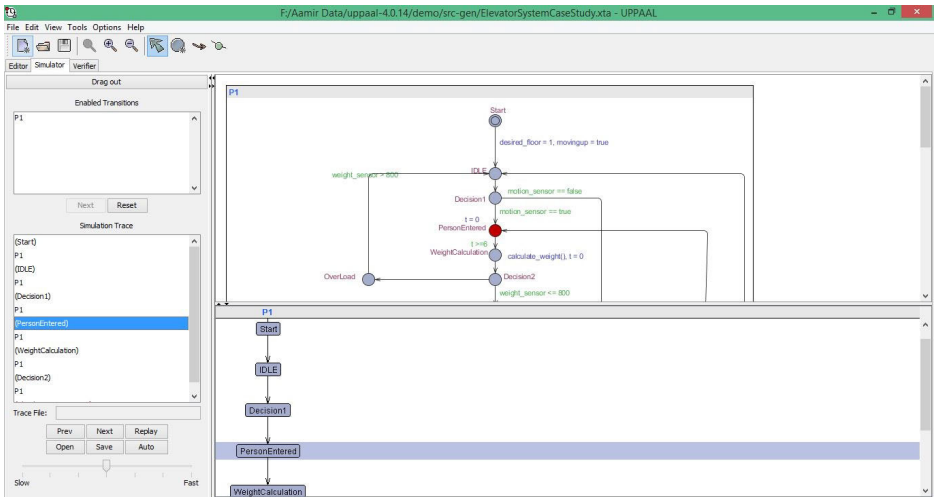


FIGURE 14. Design verification of elevator in UPPAAL.

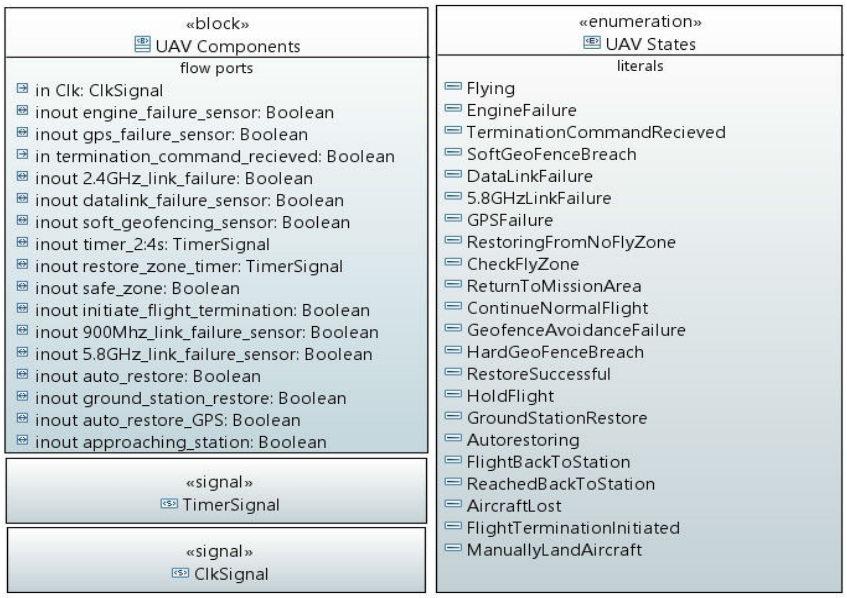


FIGURE 15. Structure of the UAV System.

cannot be established. Therefore, we perform a quantitative analysis of the proposed framework to get the genuine answers of aforementioned questions.

In order to perform a quantitative analysis of the proposed framework, we chose three professionals (i.e. P1, P2 and P3) from industry. Particularly, professional 1 (P1) has more than five years of experience in digital design using Verilog/SystemVerilog while professional 2 (P2) has three years of formal verification experience in Timed Automata/Petri nets using UPPAAL/TAPAAL tools. Similarly, professional 3 (P3) has more than five years of experience in UML/SysML based system development and is well familiar with the model transformation process. To perform a competitive quantitative analysis of MODEVES framework, the three benchmark case studies (UAV, Arbiter and Elevator) given in the article have been selected. The professional P1 is required to implement the system design and verification constraints for each case study through SystemVerilog while P2 is required to develop each case study in Timed Automata and CTL using UPPAAL tool. On the other hand, P3 is required to develop the system design and verification constraints for each case study through the proposed framework (MODEVES). Subsequently, P3 is supposed to generate SystemVerilog RTL, SVAs, Timed Automata and CTL properties from models by using the proposed transformation engine (MTE).

The evaluation results of all the three case studies are summarized in **Table 5**. It can be observed from **Table 5**

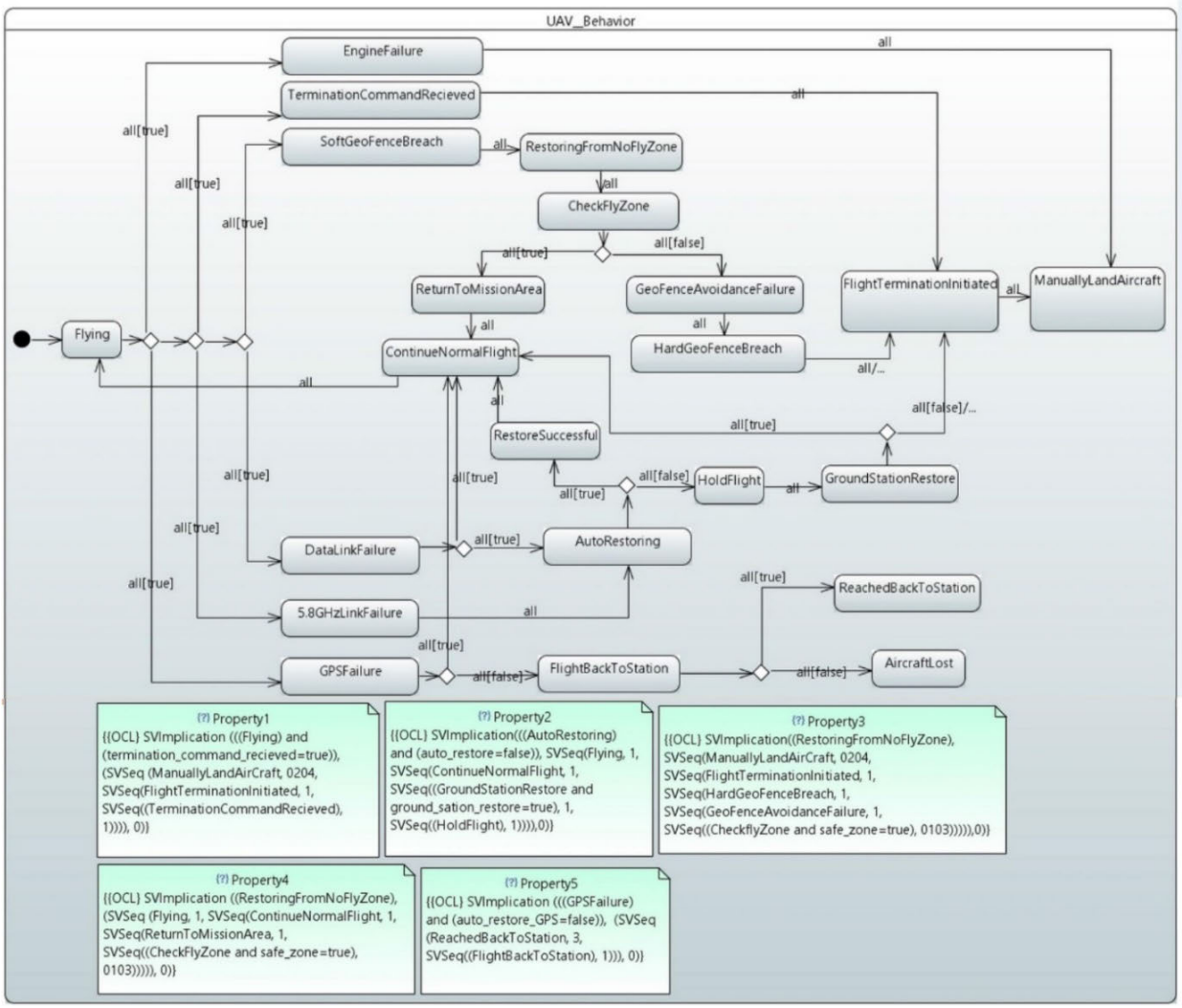


FIGURE 16. Modeling behavioral and verification requirements of UAV through MMM.

that SVOCL has saved only two hours for each case study as compared to native SVAs. In this regard, Professional 3 also reported that the modeling of assertions through SVOCL is a bit complex. In other words, it is required to carefully call different SVOCL functions recursively in order to specify complex assertions. Furthermore, it is also essential to exactly follow the syntax of SVOCL. Actually, SVOCL has been developed to represent low-level SVAs concepts in high-level models. Therefore, the modeling of assertions in SVOCL is relatively complex [15]. However, SVOCL is still marginally better than native SVAs as far as productivity gain is concerned. Moreover, the major benefit of SVOCL is the simultaneous modeling of assertions along with the system design. Furthermore, the complexity of SVOCL is comparable with existing state-of-the-art property specification approaches [15], [22]–[25].

It can be analyzed from **Table 5** that NLCTL fairly improves the productivity as compared to native CTL while dealing with the modeling of verification properties. Particularly, the productivity gains regarding NLCTL are 150%, 100%, and 100% for UAV, Arbiter and Elevator case studies respectively, as shown in **Table 5**. In this regard, Professional 3 also highlighted that it is very simple to specify verification properties through NLCTL. Furthermore, the correctness of NLCTL assertions with respect to the given grammar can be verified before transformation which is very useful in an industry environment.

So far, we have evaluated the productivity of MODEVES framework with respect to the native SystemVerilog and Timed Automata technologies, as given in **Table 5**. However, the analysis of transformation losses is equally important. In this regard, a test engineer with three years industry experience was selected. Particularly, the SystemVerilog RTL and assertions codes for all the case studies, implemented by P1 in the native SystemVerilog, were given to the test engineer in order to develop the corresponding test benches. Subsequently, the test engineer reported that there are no syntax errors in the SystemVerilog RTL and assertions codes which

TABLE 5. Quantitative analysis of the MODEVES framework.

Case Study	Implementation of Cases Studies in Native Low-Level Technologies					Design and Transformation of Case Studies through Proposed Framework (MODEVES)				Productivity Gain through MODEVES Framework (%Age)
	Working hours for SystemVerilog		Working hours for Timed Automata		Total Working hours	Working hours for Modeling Design	Working hours for Modeling SVOCL Assertions	Working hours for Modeling NLCTL Properties	Total Working hours	
	Design	Verification through SVAs	Design	Verification through CTL Prop.						
UAV	21	8	13	5	**47**	15	6	2	**23**	104 %
Arbiter	23	9	12	6	**50**	14	7	3	**24**	108 %
Elevator	19	7	11	4	**41**	12	5	2	**19**	115 %
Overall Productivity Gain										109 %

were implemented by P1. Furthermore, the test engineer also confirmed that the design verification was successfully performed for all the case studies.

Once the native implementations were verified, the SystemVerilog RTL and assertions codes for all the case studies, design and transformed through the proposed framework by P3, were given to the test engineer. Subsequently, he reported that the structural aspects in RTL and SVAs for all the case studies were successfully transformed through the proposed framework with 100% accuracy. Even the binding statements, required to bind the assertions file with RTL, have been successfully generated. He also confirmed that the proposed framework also successfully generated all the behavioral logic in RTL code. However, few syntax errors like missing semicolon, incorrect sequence of start and ending brackets and repetition of few statements are found in the behavioral code. In this regard, the test engineer took 2, 3 and 2 working hours for the correction of syntax errors in UAV, Arbiter and Elevator case studies respectively. We believe that such slight additional time periods for the corrections of syntax errors in behavioral code are acceptable considering the broader applicability of the proposed framework. Finally, the test engineer successfully performed design verification of all the case studies in QuestaSIM.

The test engineer was also asked to analyze the quality of automatically generated code. Subsequently, the test engineer confirmed that the proposed framework structural as well as SVAs code were almost similar, as implemented through SystemVerilog natively by P1. However, more lines of behavioral code were generated through the proposed framework, as compared to the native SystemVerilog code. This is because we have developed some generic transformation rules in the proposed framework by utilizing the concepts of system states. Particularly, all the control statements and required variables in the behavioral code are generated based on modeled states. This is necessary for the broader applicability of the proposed framework. Despite the more lines of codes as compared to native SystemVerilog, the automatically generated RTL code is synthesizable, as confirmed by the test engineer, for all the three case studies.

To analyze the quality and transformation losses regarding the static ABV, the automatically generated codes were given to P2. Subsequently, he confirmed that the CTL properties are transformed without any syntax or logical errors through the proposed framework for all the case studies. He also confirmed that the essential Timed Automata concepts (like locations, guards etc.) are successfully transformed through the proposed framework. In fact, P2 was able to load the generated Timed Automata model (.xta) and properties (.q) in UPPAAL tool without any changes. However, few statements/variables in the global declaration section are missing in the automatically generated code. This is because we do not include such information in MMM and subsequently no support is available in the transformation engine. In this regard, P2 took approximately one hour for each case study to include the global declaration information in the automatically generated Timed Automata models. Subsequently, P2 successfully verified the automatically generated CTL properties of all the case studies in UPPAAL.

In addition to the productivity gain in terms of working hours, the proposed framework also provides several benefits for the achievement of certain business objectives like time-to-market and cost effectiveness. For example, it can be seen from **Table 5** that it is required to employ two resources (i.e. P1 and P2) for implementation at lower level. On the other hand, the same goals can be achieved with a single resource (i.e. P3) through the proposed framework. Another benefit is the simplified identification and correction of design errors. Particularly, it is hard to track the errors during verification, where a single design requirement is represented in two different technologies. In the MODEVES framework, any error reported during the verification process is required to be corrected in a single design model only. Subsequently, the corrections are automatically transferred to both low-

level SystemVerilog and Timed Automata models during the transformation process. This leads to significantly improve the time-to-market objective.

To summarize, the aforementioned quantitative analysis reveals that the MODEVES framework certainly improves the design productivity and supports the automatic transformation of high-level models into SystemVerilog RTL, SVAs, Timed Automata model and CTL properties with minimal transformation losses.

V. COMPARATIVE ANALYSIS OF MODEVES FRAMEWORK

This section compares the MODEVES framework with those state-of-the-art approaches where the UML/SysML based solutions are provided in the context of ABV. For example, Di Guglielmo *et al.* [8] propose a complete model-driven framework for embedded systems to support the dynamic ABV. The standard UML diagrams are used to represent the system design while the verification properties are expressed in Properties Specification Language (PSL) through a separate editor. The limitation of [8] lies in the fact that it deals only with the dynamic ABV, taking C language as the target model. The proposed work in this article supports both static as well as dynamic ABV through Timed Automata and SystemVerilog. Even the target model for dynamic verification in our case (SystemVerilog) provides more advanced capabilities as compared to the native C language which is targeted in [8]. In addition, the proposed framework advocates the representation of verification properties in actual models, and not through a separate editor, to significantly reduce the design and verification gap.

While the work in [8] employs standard UML diagrams, Luciano Baresi el al. [14] propose the extension of standard UML diagrams through some temporal logic for the representation of system design. Furthermore, Corretto Property Language (CPL) is introduced to include the verification properties. In contrast to the MODEVES framework, the work in [14] only deals with the static ABV. Furthermore, the modeling approach of [14] is relatively complex as standard UML diagrams are annotated with some temporal logic concepts.

In addition to [8] and [14], where either dynamic or static verification has been targeted, there exist few state-of-the-art MBSE frameworks like MADES [13] and Gaspard [32] which support both static as well as dynamic ABV. However, both Gaspard and MADES frameworks do not support SystemVerilog along with the Timed Automata platforms, and therefore, the advanced ABV features cannot be exploited. In this regard, the MODEVES is the first framework that simultaneously support static as well as dynamic verification with advanced ABV features through Timed Automata and SystemVerilog respectively. This combination significantly improves the design productivity (Section **IV-D**) while achieving the maximum test coverage during the verification process.

While the work in [13] and [32] lack the exploitation of advanced ABV features due to the absence of SystemVerilog support, there exist an MBSE approach [30] where the support for only dynamic ABV is provided through SystemVerilog. Particularly, a UML profile for SystemVerilog (UMLSV) is proposed to model the design and verification requirements. However, the UMLSV is not complete in a sense that the support for static verification has not been provided at all. On the other hand, the MODEVES is a complete end-to-end framework providing a simultaneous support for both static as well as dynamic ABV with a significantly improved productivity as compared to UMLSV (Section **IV-D**).

In addition to the aforementioned ABV-based frameworks [8], [13], [14], [30], [32], there exist several state-of-the-art studies [41]–[43] where the UML/SysML profiles have been exploited to provide a particular solution other than ABV. For example, Nicola Bombieri *et al.* [41] introduce an integrated approach to represent the existing heterogeneous components through some equivalent SysML behavioral models. In another study [42], a transformation approach is proposed to generate the synthesizable hardware descriptions from UML state machine diagrams. Additionally, Wolfgang Mueller *et al.* [43] introduce the SysML based SATURN methodology for co-design. Although these approaches [41]–[43] provide significant UML/SysML based solutions, we are unable to perform a meaningful comparison with the MODEVES framework as their scope is different in the given research context.

In addition to the existing approaches in academia, it is interesting to consider few industrial projects for comparative analysis. For example, PolarSys[1] is a free platform, where various model-based techniques can be employed. However, in the context of the proposed framework, the CHESS and Papyrus-RT projects are largely related. A CHESS-ML profile is presented by employing the notations of UML, SysML and MARTE profiles. In Papyrus-RT project, an end-to-end environment is provided for UML-RT [40] profile which is based on the concepts of capsules, ports, protocol and connectors to represent the system structure and behavior. The major limitation of these projects is the lack of a genuine property specification approach to include the verification constraints directly in the design models as proposed in this article through SVOCL and NLCTL. In addition to this, none of the PolarSys project simultaneously support both static and dynamic ABV through Timed Automata and SystemVerilog respectively.

It can be argued that the expressiveness of SVOCL and NLCTL can be unified in a single language for a more realistic representation of verification aspects. However, the systematic merging of SVOCL and NLCTL in a single language is not possible without compromising the substantial transformation losses as there are significant syntax and semantic differences between SVAs and CTL. Therefore, unifying SVOCL and NLCTL in a single language is not feasible. For example, the extension of SVOCL for CTL properties

[1] https://www.polarsys.org/list-of-projects

eventually make SVOCL impracticable due to severe complexity while the transformation losses is another issue. On the other hand, few separate rules may be added in NLCTL for SVAs. However, this will actually lead to two separate languages (i.e. few rules for CTL and other rules for SVAs) under the umbrella of NLCTL which is unsystematic. To summarize, both SVOCL and NLCTL have their own importance in certain situations, therefore, it is more suitable to manage them separately in the MODEVES framework.

Despite the introduction of two separate modeling languages, the combination of SVOCL and NLCTL provides several benefits to designers over the traditional ABV methods: 1) SVOCL enables the exact inclusion of SVAs in design models while NLCTL allows the modeling of CTL properties in natural language alike syntax with simplicity. This leads to an effective and accurate transformation of SVOCL and NLCTL properties into corresponding SVAs and CTL assertions. (2) SVOCL and NLCTL enable the concurrent modeling of assertions along with the system design, while in traditional methods, it is essential to first develop system design (RTL or formal model) before the specifications of assertions. (3) Their combination has improved the design productivity, as compared to the traditional low level SVAs and CTL properties, as established in Section **IV-D**.

It can be argued that the objectives of MODEVES framework can be achieved through existing state-of-the-art approaches. For example, the approach in [30] can be used to perform dynamic ABV and the work in [14] can be employed to perform static ABV. As a result, both static as well as dynamic ABV can be achieved. However, the utilization of existing methods (such as [30] and [14]) in an ad hoc manner is not practically feasible without compromising the design productivity. Particularly, it is required to model the system design twice i.e. modeling in [30] for dynamic ABV and modeling in [14] for static ABV. This is time consuming and may lead to several verification errors as it is always difficult to trace and correct the errors in the presence of two separate design models. Such issues significantly reduce the design productivity and time-to-market.

To summarize, the key benefits of the proposed work are:

1. **Simplicity**: It allows to model the system design in a unified way through standard UML/SysML notations. The verification is supported through two different formalisms to make the verification process easy and accurate.
2. **Early Design Verification**: The seamless and automatic translation of ''complete models'', in terms of design as well as verification, leads to early verification by simply employing state-of-the-art tools.
3. **Productivity Gain**: It is almost 100% more efficient (Section **IV-D**) as compared to the corresponding native technologies (SystemVerilog, Timed Automata and CTL).
4. **Open Source**: It is publically available [19] for practical usage and, therefore, enables the researchers and practitioners to customize its various components, individually as well as collectively, for a particular research objective.

VI. CONCLUSIONS AND FUTURE WORK

A model driven framework (MODEVES), with a simultaneous support for static and dynamic assertions based verification, has been proposed which is capable to capture the structural, behavioral and verification requirements collectively. The modeling methodology for system design has exploited the well-known notations in UML/SysML diagrams while a simple formalism (NLCTL) has been proposed to model the static verification properties. Moreover, the SVOCL has been used to support the dynamic verification process. Subsequently, an open source transformation engine has been developed to automatically transform the high-level source models into SystemVerilog RTL, Timed Automata model, SVAs and CTL assertions. This enables the execution of both static and dynamic ABV in early development phases. The applicability of the proposed framework is demonstrated through various case studies and the experimental results have shown that the MODEVES framework is almost 100% more efficient as compared to the conventional low-level implementations.

Being a fully open source solution, which is based on some renowned MBSE standards, several extensions of the MODEVES framework are possible. For example, it can be extended for System of Systems (SoS) by incorporating various integration aspects. In addition to this, the modeling and transformation support for test benches can be included to completely automate the dynamic ABV process. Similarly, the support for modern verification standards like Portable Test and Stimulus Standard (PSS) can be encompassed to meet the growing verification demands of embedded systems.

REFERENCES

[1] R. Xu, L. Zhang, and N. Ge, ''Modeling and timing analysis for microkernel-based real-time embedded system,'' *IEEE Access*, vol. 7, pp. 39547–39563, 2019.

[2] M. Rashid, M. W. Anwar, and A. khan, ''Towards the tools selection in model based system engineering for embedded systems—A systematic literature review,'' *J. Syst. Softw.*, vol. 106, pp. 150–163, Oct. 2015.

[3] J. Hooman, H. Kugler, I. Ober, A. Votintseva, and Y. Yushtein, ''Supporting UML-based development of embedded systems by formal techniques,'' *Softw. Syst. Model.*, vol. 7, no. 2, pp. 131–155, May 2008.

[4] M. Edmund Clarke, W. Klieber, M. Nováek, and P. Zuliani, ''Model checking and the state explosion problem,'' in *Tools for Practical Software Verification* (Lecture Notes in Computer Science), vol. 7682. Berlin, Germany: Springer, 2011, pp. 1–30.

[5] H. Foster, ''Applied assertion-based verification: An industry perspective,'' *J. Found. Trends Electron. Design Autom.*, vol. 2009, Volume 3, no. 1, pp. 1–95.

[6] H. Sohofi and Z. Navabi, ''System-level assertions approach for electronic system-level verification,'' *IET Comput. Digit. Techn.*, vol. 9, no. 3, pp. 142–152, 2015.

[7] M. Loghi, T. Margaria, G. Pravadelli, and B. Steffen, ''Dynamic and formal verification of embedded systems: A comparative survey,'' *Int. J. Parallel Program.*, vol. 33, no. 6, pp. 585–611, Dec. 2005, doi: 10.1007/s10766-005-8911-2.

[8] G. Di Guglielmo, L. Di Guglielmo, A. Foltinek, M. Fujita, F. Fummi, C. Marconcini, and G. Pravadelli, ''On the integration of model-driven design and dynamic assertion-based verification for embedded software,'' *J. Syst. Softw.*, vol. 86, no. 8, pp. 2013–2033, Aug. 2013.

[9] R. David and L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.
[10] E. Cindy and D. Fismana, *A Practical Introduction to PSL*. New York, NY, USA: Springer-Verlag 2006.
[11] (Aug. 2019). *IEEE Standard for Property Specification Language*. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5446004
[12] (Aug. 2019). *IEEE SystemVerilog Standard IEEE STD 1800-2009*. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5354133
[13] R. I. Quadri, E. Brosse, I. Gray, N. Matragkas, L. S. Indrusiak, M. Rossi, A. Bagnato, and A. Sadovykh, "MADES FP7 EU project: Effective high level SysML/MARTE methodology for real-time and embedded avionics systems," in *Proc. 7th Int. Workshop Reconfigurable Commun.-Centric Syst. Chip (ReCoSoC)*, 2012, pp. 1–8.
[14] L. Baresi, A. Morzenti, A. Motta, M. M. P. K., and M. Rossi, "A logic-based approach for the verification of UML timed models," *ACM Trans. Softw. Eng. Methodol.*, vol. 26, no. 2, pp. 1–47, Oct. 2017.
[15] M. W. Anwar, M. Rashid, F. Azam, and M. Kashif, "Model-based design verification for embedded systems through SVOCL: An OCL extension for systemVerilog," *An Int. J. Design Autom. Embedded Syst.*, vol. 21, no. 1, pp. 1–36, 2017.
[16] (Dec. 2019). *Eclipse Acceleo*. [Online]. Available: https://eclipse.org/acceleo/
[17] Mentor Graphics. (Dec. 2019). *QuestaSim*. [Online]. Available: https://www.mentor.com/products/fv/questa/
[18] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou, "Testing real-time systems using UPPAAL," in *Formal Methods and Testing* (Lecture Notes in Computer Science), vol. 4949, R. M. Hierons, J. P. Bowen, and M. Harman, Eds. Berlin, Germany: Springer, 2018.
[19] MODEVES Project. (Mar. 2020). *Transformation Engine*. [Online]. Available: http://modeves.com/mte.html
[20] (Mar. 2019). *Object Management Group (OMG) Systems Modeling Language Specification*. [Online]. Available: http://www.omg.org/spec/SysML/1.3/
[21] MODEVES Framework. (Feb. 2020). *MODEVES Modeling Methodology*. [Online]. Available: http://modeves.com/docs/MMM.pdf
[22] M. Rashid, M. W. Anwar, F. Azam, and M. Kashif, "Model-based requirements and properties specifications trends for early design verification of embedded systems," in *Proc. 11th Syst. Syst. Eng. Conf. (SoSE)*, Jun. 2016, pp. 1–15.
[23] R. Bill, S. Gabmeyer, P. Kaufmann, and M. Seid, "Model checking of CTL-extended OCL specifications," *Softw. Lang. Eng.*, vol. 8706, pp. 221–240, Dec. 2014, doi: 10.1007/978-3-319-11245-9_13.
[24] S. Flake and W. Mueller, "Real-time systems, specification properties UML," in *Proc. HICSS*, 2002, pp. 3977–3986.
[25] A. Louati, K. Barkaoui, and C. Jerad, "Properties verification of real-time systems using UML/MARTE/OCL-RT," *Formalisms Reuse Syst. Integr. Adv. Intell. Syst. Comput.*, vol. 346, pp. 133–147, Oct. 2015, doi: 10.1007/978-3-319-16577-6_6.
[26] A. C. Patthak, I. Bhattacharya, A. Dasgupta, P. Dasgupta, and P. P. Chakrabarti, "Quantified computation tree logic," *Inf. Process. Lett.*, vol. 82, no. 3, pp. 123–129, May 2002.
[27] F. Essalmi and L. J. B. Ayed, "Graphical UML view from extended backus-naur form grammars," in *Proc. 6th IEEE Int. Conf. Adv. Learn. Technol.*, 2006, pp. 455–456.
[28] (Sep. 2019). *Accellera Portable Test and Stimulus Standard (PSS)*. [Online]. Available: https://www.accellera.org/downloads/standards/portable-stimulus
[29] I. Qasim, M. W. Anwar, F. Azam, H. Tufail, W. H. Butt, and M. N. Zafar, "A model-driven mobile HMI framework (MMHF) for industrial control systems," *IEEE Access*, vol. 8, pp. 10827–10846, 2020.
[30] M. W. Anwar, M. Rashid, F. Azam, M. Kashif, and W. H. Butt, "A model-driven framework for design and verification of embedded systems through SystemVerilog," *Design Autom. Embedded Syst.*, vol. 23, nos. 3–4, pp. 179–223, Dec. 2019.
[31] MODEVES Project. (Dec. 2019). *NLCTL Grammar and Validation*. [Online]. Available: http://modeves.com/nlctl.html
[32] A. Gamatié, S. Le Beux, É. Piel, R. Ben Atitallah, A. Etien, P. Marquet, and J.-L. Dekeyser, "A model-driven design framework for massively parallel embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 10, no. 4, pp. 1–36, Nov. 2011.
[33] T. Fitzpatrick, "SystemVerilog for VHDL users," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, 2004, pp. 1334–1349.
[34] M. Rashid, M. W. Anwar, F. Azam, and M. Kashif, "Exploring the platform for expressing systemVerilog assertions in model based system engineering," in *Information Science and Applications* (Lecture Notes in Electrical Engineering), vol. 376. Singapore: Springer, 2016, pp. 533–544.
[35] (Dec. 2019). *Papyrus MDT*. [Online]. Available: http://www.eclipse.org/modeling/mdt/papyrus/
[36] MODEVES Project. (Mar. 2020). *Design Verification Details*. [Online]. Available: http://www.modeves.com/dvquesta.html
[37] (Feb. 2019). *Xilinx Vivado Design Suite*. [Online]. Available: http://www.xilinx.com/products/design-tools/vivado.html
[38] (Nov. 2018). *Accellera Universal Verification Methodology Standard*. [Online]. Available: http://www.accellera.org/downloads/standards/uvm
[39] M. Lora, S. Vinco, and F. Fummi, "Translation, abstraction and integration for effective smart system design," *IEEE Trans. Comput.*, vol. 68, no. 10, pp. 1525–1538, Oct. 2019.
[40] B. Selic, "Using UML for modeling complex real-time systems," in *Proc. Lang., Compliers Tools Embedded Syst. (LCTES)*, 1998, pp. 250–260.
[41] N. Bombieri, "On the reuse of heterogeneous IPs into SysML models for integration validation," *J. Electron. Test*, vol. 29, p. 647, May 2013, doi: 10.1007/s10836-013-5409-5.
[42] M. Lora, F. Martinelli, and F. Fummi, "Hardware synthesis from software-oriented UML descriptions," in *Proc. 15th Int. Microprocessor Test Verification Workshop*, New York, NY, USA, Dec. 2014, pp. 33–38.
[43] W. Mueller, "The SATURN approach to SysML-based HW/SW," in *Proc. IEEE Comput. Soc. Annu. Symp.*, 2010, pp. 151–164.

MUHAMMAD WASEEM ANWAR is currently pursuing the Ph.D. degree with the Department of Computer and Software Engineering, CEME, National University of Sciences and Technology, Pakistan. He is also a Senior Researcher and an Industry Practitioner in the field of model-based system engineering (MBSE) for embedded and control systems. His major research interest includes MBSE for complex and large systems.

MUHAMMAD RASHID (Member, IEEE) received the bachelor's degree in electrical engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2000, the master's degree in embedded systems design from the University of Nice, Sophia-Antipolis, France, in 2006, and the Ph.D. degree in embedded systems design from the University of Bretagne Occidentale, Brest, France, in 2009. He is currently an Assistant Professor with the Computer Engineering Department, Umm Al-Qura University, Mecca, Saudi Arabia.

FAROOQUE AZAM is currently a Key Faculty Member with the Department of Computer and Software Engineering, EME College, National University of Sciences and Technology (NUST), Pakistan. He has been involving in post graduate teaching and research, since 2007. His research interests include model driven software engineering, model driven testing, model driven embedded applications, model driven web engineering, software design and architectures, and so on. He is a Regular Member of Evaluation Committees of Pakistan Engineering Council (PEC) and Higher Education Commission's Technological Development Funding (HEC-TDF). Since April 2020, he has 138 international journal and conference publications on his credit.

AAMIR NAEEM received the master's degree from the Department of Computer and Software Engineering, CEME, National University of Sciences and Technology (NUST), Pakistan. He is currently working as a Research Associate with the Model Driven Software Engineering Group, CEME-NUST. His research interests include formal verification and model-based development for embedded systems.

MUHAMMAD KASHIF received the master's degree from Istanbul şehir University, Turkey, in 2019. He is currently pursuing the Ph.D. degree in computer science and engineering with Hamad Bin Khalifa University, Qatar. His research interests include system level modeling, early design verification of embedded systems, hardware security, quantum computing, and machine learning.

WASI HAIDER BUTT is currently an Assistant Professor with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. His research interests include model driven software engineering, and web development and requirement engineering.