A novel approach to Real-time contract based reasoning for Hybrid Systems

1st Surinder Sood
Dept. of Electrical and Computer eng.
University of Auckland
Auckland, NewZealand
surinder.sood@gmail.com

2nd Avinash Malik
Dept. of Electrical and Computer eng.
University of Auckland
Auckland, Newzealand
avinash.malik@auckland.ac.nz

3rd Partha Roop
Dept. of Electrical and Computer eng.
University of Auckland
Auckland, Newzealand
p.roop@auckland.ac.nz

Worst Case Execution Time (WCET) analysis of large and complex hybrid systems can be time consuming. Contract based design allows for compositional reasoning of complex systems. Contracts justify the behavior of a system by way of assumptions (which are to be satisfied by the system environment) and guarantees, which are to be met by the system. Contracts also play a major role in compositional reasoning, refinement and re-usability of the system components. In this paper, we present a formal framework to enforce real-time contracts using Hoare triples, for synchronous system design and verification. In that regard, we propose real-time Hoare rules which are based on the WCET of the system and its components. We verify the real-time behavior of the system by applying these rules. These rules not only justify the system behavior and the behavior of its components but their timing as well. We also show that these Hoare rules are sound. Then we show that the synchronous composition of component level Hoare rules based contracts justify a system level contract. This real-time contract composition and reasoning technique which is based on real-time Hoare logic rules is the first ever attempt in synchronous system design and verification.

***Index Terms*—Cyber-physical Systems (CPS), Hoare Logic, Worst Case Execution Time (WCET), Timed properties**

I. Introduction

Real time functional correctness of safety critical hybrid systems can only be ensured if they are met in strict time. With technological advancements, the scope and complexity of such systems is increasing at a very fast pace. This calls for effective and rigorous methods to find defects in the early design phase so as to guarantee their correctness, safety and strict in time behavior. Simulink® is the de-facto standard for model based design and validation of such systems. However, it does not provide an adequate platform for compositional real-time verification of such safety-critical systems, although few attempts are made in this field in the recent past [1], [2]. Compositional verification becomes very important as the designs scale up in complexity and composition.

Contract-based design, a concept first coined for defining software specification [3] is now applied to design of embedded systems [4]–[9]. A contract is usually specified by a pair of *assumptions*, or the properties, which the environment of a component must satisfy, and *guarantees*, or a set of properties which must be satisfied by the component. Contract based design comes with many advantages: ① contracts help in compositional reasoning, ② contracts help in re-use of components to design different systems, ③ contract refinement [10] helps guarantee the safety behavior of a system. This means that if the refined contracts are satisfied then a system contract will also be satisfied, thus guaranteeing system correctness and safety. Static reasoning is the most popular technique to do contract verification, which is a very important step in the design of embedded systems. There are many works focusing on *assume-guarantee* based static reasoning of contracts [10]–[14]. But none of the works focus on real time contract based reasoning for hybrid systems and the said techniques (which are mostly Linear Temporal Logic (LTL) based) are not suited for effective reasoning of a program stack trace.

978-1-5386-6195-6/18/$31.00 ©2018 IEEE

In this paper we define the notion of real time contracts which use Hoare logic rules [15]. Main **contributions** of our work are briefly described as follows:

① We define real-time contract composition and reasoning technique which is based on Hoare Logic rules [15] and is well-equipped to reason the program states. In this regard, we first define a new language syntax which is a subset of Pret-C [16]. A given Simulink® model is translated into this language. We call this new language as Synch-C, ② then we define real-time Hoare logic rules which are based on WCET of the program, ③ we derive Satisfiability Modulo Theory (SMT) encodings of the program written in this language using WCET based Hoare logic rules and micro architectural level instruction timings. We feed these SMT encodings to Z3®SMT solver and guarantee that they satisfy the contracts defined for every program component. This is how we prove that our rules are sound and also guarantee program correctness, ④ finally we show that a synchronous composition of these real time component level contracts guarantee a system level contract. The overall methodology is shown in Figure 1 and is described as follows:

- *Step-1*: At the very first step, we create a Simulink® model of the system. Every component of the model is modeled as per it's contract definition. We use Hoare logic contracts refinements for every component. For example, in Figure 1, $Comp-1$ is an implementation based on it's contract refinement C_{h1}. We elaborate this in Section III.
- *Step-2*: We define a new programming language whose syntax is a subset of a Pret-C [16] and call it Synch-C. The Simulink® model is then translated into this language program.

- *Step-3* In this step we do the following:
  - The micro-architectural analysis of the program obtained in previous step is done. This involves translation of the program into PT-Arm [17] assembly for assembler instruction timing analysis. The WCET computation of the program is based on the number of clock cycles taken by each assembly instruction. It is assumed that the control flow bound analysis is already done using the techniques specified in [18].
  - Then we obtain the SMT encoding of the program by applying WCET based real-time Hoare rules defined in this paper. The SMT encoding of a component is based on the component level Hoare logic contracts defined for it. For example, property $P1$ is based on the contract definition C_{h1} for component, $Comp-1$. These component level contracts are the refinements of a system level contract C.
- *Step-4*: These SMT encodings are input to a SMT solver so that all the component level properties are satisfied. Then a synchronous parallel composition P of all component level properties automatically satisfy a system level property. This we describe in Section III. Consequently, the correctness of a system level contract C_h is guaranteed.

The rest of the paper is structured as follows: In Section II, we describe the running example and envision the contract based framework that requires methods proposed in the paper. In Section III we give background of Hoare Logic and Hoare Logic contracts. We then describe how a contract compositional framework is used to justify system level contracts in Section IV. In Section V we introduce a new imperative language called Synch-C and show how a Simulink®model is translated in to Synch-C. Then we describe WCET based derivation rules using Hoare Logic in Section VI. These rules are then used to formulate SMT encoding which we describe in Section VI-B. Later on we present four benchmarks on which we tested our methodology, this is described in Section VII. Finally, in Section VIII we present the related work and we conclude in Section IX.

II. Motivating Example and Problem Description

We use a train-gate control system adapted from [19] as the running example of the paper. Figure 2 shows a circular track on which a train moves and a gate crossing for the incoming traffic. The total train trajectory is 25m, and is divided into two regions, one is *Un-safe region* which is of length 15m while other is the *safe region* of length 10m. Whenever train enters the *safe region* gates are opened, else, the gates are closed. The Simulink® model of this train gate system is shown in Figure 3c. Train's movement is depicted by `y_out` which is input to the controller, while train's angular motion is tracked by signals `xc_out` and `yc_out`. The train's linear motion follows the Ordinary Differential Equation (ODE) described in Equation 1, (here v_f is velocity of the train). The gate movement is based on the ODE described in Equation 2 (here x is the position of gate).

$$f(y) \stackrel{def}{=} [\dot{y}] = [v_f] \tag{1}$$

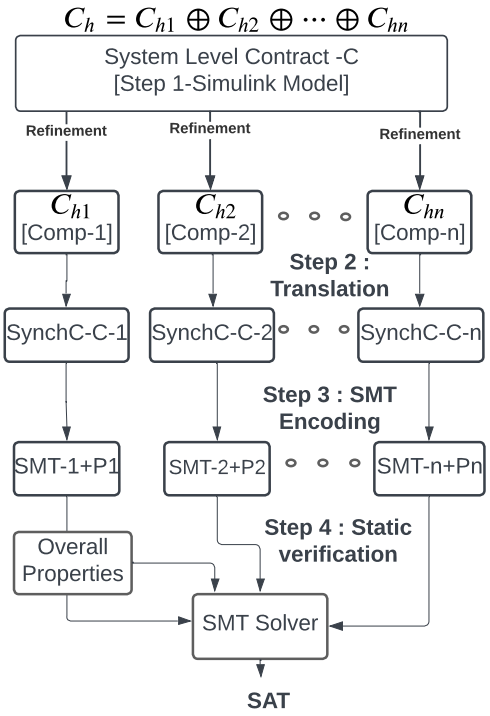


Fig. (1) Proposed Methodology: Here `Ch` is the system level contract for the system and component level contracts `Ch1`, `Ch2`, . . . , `Chn` are the refinements of `Ch`.

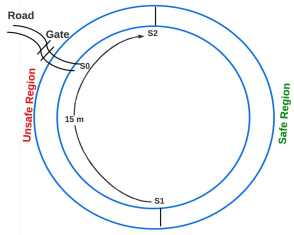


Fig. (2) Train and Gate Setup. Total train trajectory is 25 m.

$$g(x) \stackrel{def}{=} [\dot{x}] = \left[\frac{(1-x)}{5\text{-}\frac{x}{2}}\right] \tag{2}$$

After analyzing the train's position, the controller takes decision to control the gate movement. This is represented as `up` and `down` signals going to the gate model. In order to guarantee correctness of the model's functional and timing behavior, we need to make sure that the controller decisions for gate position are correct w.r.t train movement. To guarantee this, contracts have been defined and they should be satisfied for model correctness. The current prevalent technique for contract verification uses LTL properties [10], [20]–[23], [22] which has it's own issues. For example, let us consider a system level contract: *Whenever gate is down and train is moving then gate eventually opens.* This system level contract is refined by two contracts:① Train Contract: *If train position is between 0 and 15m then eventually it's position will be between 15 and 25m.* ②Gate contract: *If gate is initially closed then it eventually opens.* The system level contract is justified if it's corresponding component level contract refinements are correct and justified, such that their synchronous composition satisfies the system level contract [10]. But this framework of

contract specification and refinement suffers from following issues:

1) The system level contract and it's contract refinements are not real-time, as they miss to define the strict timing system requirements. For example, Train contract refinement says that the train position will be any value between 15 and 25m within sometime. This means that the actual train position is not known with this contract specification. Similarly, the system contract says that the gate will open in some time, but what is the time when it will actually open is not known.
2) Any LTL property cannot reason about the first and last state of any program code (on which contract verification and reasoning heavily depends), even if the property is satisfied on the stack trace of the program. Hence, accurate timing verification of the contracts specified as temporal properties cannot be performed.
3) Any LTL property does not cover various invariant conditions at the system and component level. For example, the current system level contract has invariant condition: the train position on the circular track is always greater than zero and $\sqrt{xc\_out^2 + yc\_out^2} = R$, where R is the radius of the circular trajectory. These invariant conditions apply to all the contracts.
4) Even if we have the correct contract refinements of a system contract and as per [10] the composition of such sub-contracts justify the system level contract. But such type of compositional contract based property verification of Simulink®models is not possible as Simulink®lacks clear semantics and often involve third-party tools which are poorly specified [24].

We remedy these issues by proposing real-time Hoare Logic [15] based contracts definition and composition framework (which is described in subsequent sections) and redefining the contracts with timing intervals. Hence, we have system contract as: *Whenever gate is down and train is moving then gate eventually opens within 110000 clock cycles*, Gate level contract as: *Whenever gate is down then gate opens within 110000 clock cycles*, while the train contract is defined as: *If train position is between 0 and 15m then within 110000 clock cycles it's position will be 23m.*

III. Hoare logic

In this section we give a brief background of Hoare logic. Then we describe Hoare Logic contracts and their soundness.

A. Background of Hoare Logic

Hoare logic [15] is a formal mechanism used for reasoning on the program correctness. The mechanism typically consists of a triple, (popularly know as Hoare Triples), which consists of {P}C{Q}. Here P is the pre-condition, which must hold before any command C is executed. Once the command is executed the post-condition Q must be guaranteed.

B. Hoare Logic contracts

We refer to the pair $C_h = (P, Q)$ as a hoare logic contract, where P is the *pre-condition* and Q is the *post-condition*. We say that a particular implementation, say S satisfies the contract C_h, denoted as $S \models C_h$, if and only if $S \models \{P\}S\{Q\}$. Furthermore this means $[\![S]\!] \subseteq [\![C_h]\!]$, where $[\![S]\!]$is contract-relative denotation of program S and $[\![C_h]\!]$is denotation of contract C_h [25].

Definition 1: Contract Denotation: The denotation of a contract C_h is defined as follows:
$[\![C_h]\!] \stackrel{def}{=} \left\{ (s, s') | \forall I.(s \models_I P \implies s' \models_I Q) \right\}$. This means that over a given interpretation I of the program S, the Hoare triple is valid if for every state s such that $s \models_I P$, if execution of S from state s terminates in a state s', then $s' \models_I Q$. Such contract semantics express an *assume-guarantee* style treatment of programs [25].

Definition 2: Soundness: A hoare logic contract C_h is said to be sound if it's post condition is guaranteed when a program is executed based on the satisfaction of pre-condition. Mathematically, C_h is sound iff $S \models C_h$.

As an example, we describe train contract from our running example as Hoare Logic contract. The train contract says: *If train position is between 0 and 15m then within 110000 clock cycles it's position will be 23m.* For this contract the *pre-condition* is: $0 \leq y\_out \leq 15$, here y_out is the train position. The *post condition* is: $cycles \leq 110000 \wedge y\_out == 23m$. Between these pre and post conditions the train ODE described in Equation 1 evolves. Our train model implementation satisfies this contract, hence we say that $S \models C_h$ and based on Definition 2 the contract is sound.

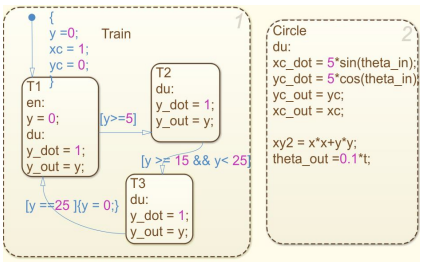
IV. Contract based compositional reasoning

In this section we describe refinement of a system level contract and then show that the composition of refinements justifies a system level contract. We use Hoare logic contracts as described in Section III-B to define the component level contract refinements. We briefly describe contract composition techniques (which are based on system decomposition and contract refinement) described in [10]. First we describe decomposition of a system S among it's components, which is established over subset of connections of a system interface. This is formally described in Definition 3. Later on a contract refinement is defined over these components which is formally described in Definition 4. We show that the conjunction of contract refinements justifies a system level contract. This is proven in Theorem 1.

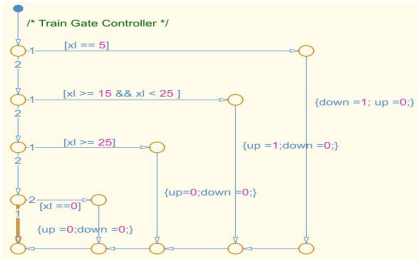
Definition 3: Decomposition: Let S be a system. Then the architectural decomposition of S is a pair $\sigma =< Sub, \gamma >$, such that for every component of S:① $Sub(S)$ is the set of subcomponents of S and $Sub(S) \neq \phi$, if S is a composite component. ② $\gamma(S)$ represents the connections among subcomponents.

Definition 4: Refinement: Let $\psi(S)$ denote the set of contracts of a system S and if $\mathcal{S}$ denotes a set of components of S, then $\psi(\mathcal{S}) = \bigcup_{S \in \mathcal{S}} \psi(S)$. Given a decomposition $\sigma =< Sub, \gamma >$ and a system S, a set of contracts $\mathcal{C}_\mathrm{h} \subseteq \bigcup_{S' \in Sub(S)}$ is a refinement of C_h, written as $\mathcal{C}_\mathrm{h} \leq_{ref} C_h$, if and only if following are true:

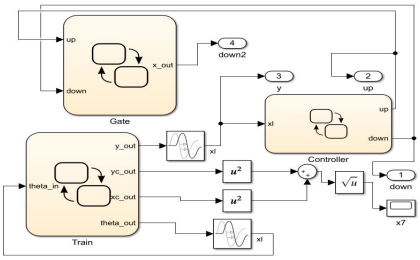
1) If correct implementations of sub-contracts satisfy the system contract C_h and



(a) Train model's Stateflow Chart.



(b) Controller Simulink® model.



(c) Train Gate Controller model in Simulink®

Fig. (3) Train Gate controller model is shown in Figure 3c, Stateflow chart for controller and plant is shown in Figures 3b and 3a

.

2) For each Hoare sub-contract $C_{U,h}$ of every sub-component U, the correct implementation of other Hoare sub-contracts and a correct environment of C_h form a correct environment of U. This is described mathematically in Equation 4.

Remark 1: If there is only one sub-component having contract C'_h, then C'_h refines C_h, written as $C'_h \leq_{ref} C_h$, if $C'_{h,impl} \subseteq C_{h,impl}$ and $C'_{env,h} \supseteq C_{env,h}$.

Theorem 1: For a component S, a Hoare contract C_h of S, a decomposition $\sigma = \langle Sub, \gamma \rangle$ and $\mathcal{C}_h \subseteq \psi(Sub)$. $\mathcal{C}_h \leq_{ref} C_h$ iff following conditions hold:

1) Conjunction of hoare sub-contracts combined over a given set of connections γ with projection on the variables V_S of S (with $V_{Sub} \in V_S$) should satisfy the system level contract C_h. This is described mathematically in Equation 3.

$$\exists V_{Sub}(\bigwedge_{C'_h \in \psi(S') \cap \mathcal{C}_h, S' \in Sub} C'_h \wedge \gamma) \models (C_h) \quad (3)$$

2) Conjunction of pre-conditions (P) of S with contracts of other components over a given set of component interface connections γ which are projected on variables of the component U should satisfy the pre-condition of U.

$$\exists, V_{Sub} \backslash U (P_S \wedge \bigwedge_{C'_h \in \psi(S') \cap \mathcal{C}_h, S' \in Sub \backslash \{U\}}^{\forall U \in Sub, \forall < C_U > \in \psi(U) \cap \mathcal{C}_h} C'_h \wedge \gamma) \models (P_U) \quad (4)$$

Proof 1: We refer to [10] for the proof of this theorem.

Once the component level contract refinements are guaranteed, then as per Theorem 1 the synchronous composition of these properties, guarantee the system level contract. We specify the system and component level contracts for our running example to describe this as follows: C_S = *If train is moving and gate is closed then it will open within 110000 clock cycles,* is the system level contract, $C_{S,P}$: $y\_out \geq 0 \wedge Down == 1 \wedge Up == 0$ is the system contract's pre-condition and $C_{S,Q}$: $cycles \leq 110000 \wedge y\_out > 0 \wedge Down == 0 \wedge Up == 1$ is the post condition. Similarly, $C_T =< C_{T,P}, C_{T,Q} >$ where C_T is train contract with $C_{T,P}, C_{T,Q}$ with pre and post condition respectively where, C_T= *If train position is between 0 and 15m then within 110000 clock cycles it's position will be 23m.*, $C_{T,P}$: $0 \leq y\_out \leq 15$ and $C_{T,Q}$: $cycles \leq 110000 \wedge y\_out == 23$. Here, y_out is train's position. Finally, gate contract is C_G: *If gate is initially closed then it opens within 110000 clock cycles.* with $C_{G,P}$: $Down == 1 \wedge Up == 0$ as pre condition and $C_{G,Q}$: $cycles \leq 110000 \wedge Down == 0 \wedge Up == 1$ as post condition. It should be noted that the timing for all the Hoare logic contracts is computed based on the micro architectural assembly language timing analysis described in Section V.

It is obvious that $C_{T,Q} \subseteq C_{S,Q}$ (this is because if train covers 23m, then gate will definitely open in 110000 cycles) and $C_{S,P} \wedge C_{G,P} \models C_{T,P}$ (This means that if gate is closed and train is moving which means train position is ≥ 0). Similarly, $C_{G,Q} \subseteq C_{S,Q}$ and $C_{S,P} \wedge C_{T,P} \models C_{G,P}$. This means that, $C_T \wedge C_G \models C_S$. Hence, the component level contracts refine the system level contract and the conjunction of the refinements of component level contracts as per Equation 3 of the Theorem 1 guarantees the system level contract. Obviously, the contracts are sound if they justify soundness criteria as per Definition 1 and 2. The cycle time of every component guarantees is derived from its SMT encodings, these are generated by applying WCET based real-time Hoare rules on Synch-C programs and is described in Section VI.

V. Translating Simulink® Model into the Imperative Language

To perform micro-architectural instruction timing analysis, we translate the Simulink® model into an imperative language called Synch-C, which is a subset of Pret-C [16]. We first describe the syntax of our imperative language and then we describe how the Simulink® model constructs are mapped to this language syntax. Finally, we use the Synhc-C program to create SMT encodings for static reasoning.

A. Syntax of the imperative Language

For our imperative language Synch-C, we consider subset of the constructs defined in Pret-C [16] language. We chose this language because it supports concurrency and is time analyzable. It considers all the standard arithmetic and Boolean expressions besides also considering the assignments, conditional statements and iterative control structures i.e. while loops. The syntax of our language is defined inductively in Equation 5. Here, `e` is a standard arithmetic expression, `b` is a standard boolean expression and `C` is the standard command being executed. Also, `skip` indicates a NO operation and $C_1 \parallel C_2$ means that two commands are executed synchronously on

different processors. Finally, F describes a function call having arguments $X_1^{'} \dots X_m^{'}$. The function calls made are *call by value*.

$$C \stackrel{def}{=} skip|x = e|C_1; C_2|if(b)\{C_1\}else\{C_2\} \\ |while(b)\{C_w\}|C_1 \parallel C_2|\langle Y_1^{'} \dots Y_n^{'}\rangle = F\langle X_1^{'} \dots X_m^{'}\rangle \quad (5)$$

B. Semantics of the Imperative language

In this section we define the semantics of Synch-C whose syntax is defined in the previous Section. The main aim of defining the semantics of Synch-C is to determine how a Synch-C program (p) executes. We explain this by describing a program state before and after the application of the syntax command. For this we assume s as the program state before applying command C and s′ as the program state after the application of the command.

1) $skip$ command: Here the initial and final state of the program remains same, which is s for program p.
2) $assignment$ command: If the command is an assignment $x ::= e$ and the initial state is s then the final state is also s, where the value of x is replaced by evaluation of expression e.
3) $sequence$ command: If the assignment is a sequential composition, then we have a combined sub-commands $C_1; C_2$, such that if starting program state before execution of C_1 is s_1 then after C_1 finishes execution, then the program state changes to s_2, which is the initial state for sub-command C_2. When C_2 finishes execution the program state terminates in to s_3.
4) $conditional$ command: This command comprises of *if-then-else* kind of rule. Here the current program state is s, if the boolean evaluation of the *If-then* part is *true*, then the sub-command C_1 executes and program terminates in the state s′. The *else* part does the same for the sub-command C_2.
5) $while$ loop: Here if the $while$ loop condition evaluates to false, the whole loop is skipped. However, if the condition evaluates to *true* in the program state s_1, and the body (command C_w basically) of the loop starts in state s_2, then the program terminates in state s_3.
6) $function\ call$: In this case a function call is made by the program when it is in state s_1. The function is called by value. Upon completion of the function call, the value returned by the function is copied to the program stack and it's state is changed to s_2.
7) $Synch\ parallel$ command: In this case two programs say p_1 and p_2 are allocated simultaneously to two processors, which are called by a main program which is in state s. They run independent of each other and update their respective program states and stacks. On termination the resulting values of these programs are updated in the main program stack. Eventually, the main program terminates in state s′.

Code Listing (1) Code Snippet for Train model

```
train(tp,delta)
/*store train position tp into local stack tstack*/
 tstack[ts_y] = tp
if(tstack[ts_y] <= 5) /*T1*/
  tstack[ts_y]=tstack[ts_y]+tstack[ts_y]*delta/*c1*/
else if(tstack[ts_y] >5 and tstack[ts_y] < 15)/*T2*/
  tstack[ts_y]=tstack[ts_y]+tstack[ts_y]*delta/*c2*/
else if(tstack[ts_y]>=15 and tstack[ts_y]<25)/*T3*/
  tstack[ts_y]=tstack[ts_y]+tstack[ts_y]*delta/*c3*/
else
  tstack[ts_y] = 0/*c4*/
/*local stack values are passed to caller*/
  return tstack[ts_y]
```

C. Translating Simulink® model to imperative language

In order to get the Synch-C program from the Simulink® model, we map every Simulink® block to the language syntax. This is briefly described below:① Every state in the Stateflow® corresponds to a specific conditional block of if-then-else, ② the evolution of ODEs in a given state is implemented as a while loop, ③ assignment syntax maps every assignment in Stateflow®, ④ any function call from any Stateflow® block is handled by the corresponding Synch-C function syntax, ⑤ parallel states are mapped as parallel functions executed simultaneously, while, ⑥ the skip handles any NO operation. The train Stateflow®chart is shown in Figure 3a. The corresponding Synch-C code obtained is shown in Listing 1, while the complete code for running example is shown in [26]. As can be seen from the listing every state in Simulink®maps to a specific conditional block in Synch-C. The initial values are assigned before the conditional blocks are executed. Every execution of a conditional block updates the stack state of the program, here we define the stack as tstack which is updated on every assignment statement. The entire conditional block is wrapped by a *while* loop to execute it a number of times. The ODEs evolve on every iteration of the *while* loop until the loop condition holds. The ODE evolution step selection is based on robustness criteria [27] and the iteration is done number of times based on timing analysis described in Section VI.

VI. WCET based Real-time Hoare Logic rules

In this section we describe real-time Hoare rules based on Hoare logic described in Section III. These rules are then applied on our running example to reason the timing and functional correctness of the program, consequently, we guarantee that these rules are sound.

A. WCET syntactic derivation rules

We extend the Hoare Logic rules specified in [15] as real time Hoare logic rules, and then later on we prove that these rules are sound. All the rules are specified in Figure 4 and are explained briefly below.

- *The **skip** Rule*: This is an empty statement or NOP rule. The state of the program is maintained the same, before and after the execution of the rule. Here Γ (ref Figure 4) is the global stack to save different global variables, P is the pre-condition while, W is the time before the execution of the skip statement. The rule states that the worst time taken to execute the skip is zero time units.
- *The **assignment** Rule*: The *assignment rule* states that the worst case execution time taken by an assignment

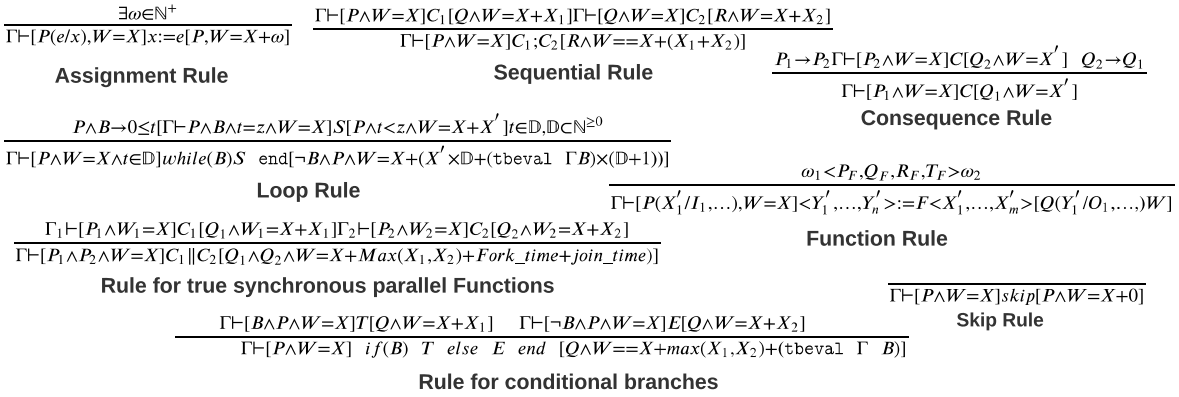


Fig. (4) Hoare Logic rules with WCET.

in a program is ω time units, where W is the time before the assignment statement execution and updation of the contents on the stack Γ.

- *Rule for* ***Conditional Branches***:The *conditional rule* states the following: ①The post condition after the execution of `then` part is also the post condition after the execution of `else` part. ② The worst case execution time taken by the `if-then-else` is the maximum of the time taken by the `if-then` and `else` blocks plus the time taken in evaluating the `if` condition which is represented by `tbeval`. Here B (ref Figure 4)is the condition which can be either true or false, P is the pre-condition, Q is the post-condition, W is the time before execution of the condition, while X_1 and X_2 are the times of `If` and `Else` blocks.
- *The* ***Sequential*** *Rule*: The *sequential rule* states that the WCET taken in executing two sequentially executed programs C_1 and C_2, where C_1 executes before C_2 is the sum of WCET taken in executing C_1 and C_2. The rule is formally stated in Figure 4, where Q is the mid-condition, P is the pre-condition and R is the post condition.
- *The* ***Loop*** *rule*: The *Loop rule* applies to control loops like *while*. The total time taken by the loop to execute once is X$^{'}$ and the number of iterations it takes $|\mathbb{D}|$. Hence the WCET of the loop will be $|\mathbb{D}|\times X^{'}$ plus one extra iteration when the `while` condition is false, which is indicated by $\mathbf{B} \times (|\mathbb{D}| + 1)$. Hence, the WCET = X + $|\mathbb{D}|\times X^{'}$ + $\mathbf{B} \times (|\mathbb{D}| + 1)$, where X is the time when the loop has started. P specified in the rule is loop invariant which should hold outside of the loop as well.
- ***Consequence*** *Rule*: The WCET for this rule is actually the time spent in proving the post condition Q_1, if the starting time was X when the pre-condition P_1 was true.
- ***Function*** *Rule*: This rule is applied when a function call is made from within a function. Here (ref Figure 4) T_F is the time to do the body of the function. R_F is the proof that the time T_F is actually equal to the function body's execution time. There is also a time to copy the contents from caller's function stack to the callee's function stack and is represented as $(X_1, \dots)$, and the time to copy results back to the caller's stack is represented as: $(Y_1, \dots)$. These are shown in the antecedent as ω_1 and ω_2. Where $\omega_1 = (teval - copy - arg)$ and $\omega_2 = (teval - copy - return)$. The WCET is given by W, where $W = (T_F + X + m \times \omega_1 + n \times \omega_2)$. It should be noted that the precondition P_F on function F is always on the input pins while the post condition Q_F is always on the output of the function.
- *Rule for* ***True Synchronous parallel function*** *execution*: True parallel means that two or more functions are being executed on different machines. The rule states the parallelism of two component programs C_1 and C_2. Each component maintains it's own stack (which is Γ_1 for C_1 and Γ_2 for C_2) and is updated during the course of execution.The time to copy arguments to it's own stack and also on to the caller's stack (Γ) are included in X_1 and X_2 respectively. P_1 and Q_1 are the pre and post conditions of component C_1, while P_2 and Q_2 are the conditions for component C_2 respectively. As described in the rule, total time taken by the components to finish is the sum of maximum of the X_1 and X_2 of the components plus *fork* and *join* time these components take. The $max()$ operator follows the time taken by the last component to finish, once this is done then the caller's stack is updated. If there is a shared variable between the two component programs, then ① at the time, whenever both of the components are called, they read value of the variable from the caller's stack Γ and, ② the final value of the shared variable is written to the caller's stack by the component program which finishes last.

B. Generating SMT encoding by applying Real-time Hoare rules in contract based design

We apply rules specified in Section VI-A to generate SMT encodings which are used for static reasoning of the corresponding contract based designs.

The overall procedure to generate SMT encodings is described as follows:

1) We use PT-Arm [17] compiler to get the assembler instruction timings of the Synch-C program.
2) We apply WCET rules to arrive at the SMT encodings, which constitute pre and post conditions of Hoare logic along with WCET information of every program block.

3) All encodings (from previous step) are written in Python and are proved by Z3®solver. If all properties are proven then this means that our program is correct and it also guarantees that our Hoare logic based WCET rules are sound. We chose a specific lock step value on which our code executes. It is worth mentioning that all the ODEs also evolve on this lockstep. Reason for choosing a specific lock step is to guarantee robustness of our properties and hence we meet the robustness criteria, which is described in [27], [28].
4) Once all the properties are proven we claim that the Simulink/Stateflow®is guaranteed to behave as per functionality within strict timing range.

We explain the above procedure using our running example. The Synch-C code has a `top` function which calls the $train()$ and $circle()$ functions running in parallel. This function also calls the other functions as well, but we only show `train()` function (Listing 1) because of space constraints. The `train()` and `circle()` functions are called using Synch-C syntax as shown below:
$\Gamma[T_p], \Gamma[xc\_out], \Gamma[yc\_out], \Gamma[theta] = train(tp, delta) \parallel circle(xc\_out, yc\_out, delta)$. Here Γ is a global stack which store the values returned by $train()$ (returns the current train position) and $circle()$ (returns the angular position $theta$ along with xc_out, yc_out co-ordinates) respectively. We apply the derivation rules in a bottom up manner on the program to arrive at the SMT encodings. We assume that the *top* function contains this parallel function execution only, for the sake of simplicity. First we apply `function rule` to $train()$ function to arrive at the interim pre condition.

$$\frac{\omega_{21}\ \ train \overset{def}{=} < P_{train}, Q_{train}, T_{train} > \omega_{22}}{[P(\Gamma[T_p]/I_1, \Gamma[delta]/I_2), W-(X)]train = expr} \quad (6)$$

Here, $expr$=$train < \Gamma[T_p], delta >$$[Q(xc_out/O_1), Q(yc_out/O_2), Q(theta/O_3), W]$, and $W - X$ is the time before the function execution, then after the function execution it is W. Also, ω_{21} and ω_{22} is the time to copy the contents from caller to callee stack and vice-versa. T_{train} is the time taken by the body of the function to execute, and we take ω_{23} as T_{train}. All these timings are obtained by translating the Synch-C program into PT-Arm [17] assembly for assembler instruction timing analysis. Hence the overall time is $\omega_t = \omega_{21}+\omega_{22}+\omega_{23}$. Similarly, for $circle()$, we take $\omega_c = \omega_{31} + \omega_{32} + \omega_{33}$. The overall SMT encoding is computed by applying the rule for `True synchronous parallel` functions and is shown below:

$$\frac{\begin{array}{c}\omega_t\ tstack \vdash train(angle, delta) \parallel \\ th \vdash circle(xc\_out, yc\_out, theta)\ \omega_c\end{array}}{\Gamma \vdash [P_1 \wedge P_2 \wedge W - \tau]C1 \parallel C2[Q1 \wedge Q2 \wedge W]} \quad (7)$$

In Equation 7, $\tau = (max(\omega_t, \omega_c)+Fork\_time+join\_time)$, $C1, C2$ are the train and gate functions respectively. Also, `tstack` and `th` are the train and circle stacks, which are updated whenever train and circle are executed. P_1, P_2 are the preconditions, while Q_1, Q_2 are the post conditions.

Now we derive SMT encodings for train function whose Synch-C code is given in Listing 1. It consists of `If-then Else` statements which describe the States of Simulink®train model. We apply the derivation rules in a bottom up manner on the program to arrive at the SMT encodings. We always start with the assertion $[W]$ as the assertion `Q` of the last construct(the last construct in this case is the final `Else` block). Now we'll apply derivation rules to obtain $[P]$. We apply `If-else` rule on the last `If-Else` block of the program. The last `Else` block consists of single assignment statement, so we apply `Assignment Rule` to get $[P]$. Hence we have:

$$\frac{= \omega_4}{[W-(\omega_4)]tstack[ts\_y] = 0[W]}$$

The assertion $[W - \omega_4]$ states that,if the WCET after the execution of statement `tstack[ts_y]=0` is W then before the statement execution,WCET must be $W - \omega_4$, where ω_4 is the clock cycles required to execute the assignment and updating it's value on the local stack. The value ω_4 is obtained from the micro-architectural timing analysis of the PTArm processor [17]. Similarly, we apply `assignment rule` to clause `c3` as follows:

$$\frac{= \omega_3}{[W-(\omega_3)]c3[W]}$$

Here clause $c3 = tstack[ts\_y] = tstack[ts\_y] + tstack[ts\_y] \times delta$. Applying `conditional rule` we get:

$$\begin{array}{c}[(P \wedge B_3 \wedge \neg B_4 \wedge \omega_3) \oplus (P \wedge \neg B_3 \wedge B_4 \wedge \omega_4)] \overset{def}{=} \\ [(X - max(\omega_3, \omega_4) \wedge P \wedge B_3 \wedge \neg B_4) \oplus \\ ((X - max(\omega_3, \omega_4) \wedge P \wedge \neg B_3 \wedge B_4)]\end{array} \quad (8)$$

In Equation 8 ω_3,ω_4 are the WCET for clauses `c3` and `c4` respectively, while B_1, B_2 are the conditions of `If-else`. Likewise we apply `assignment rule` to clauses `c2` and `c1` as follows:

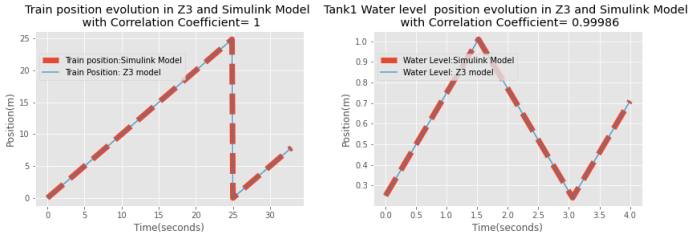
$$\frac{\begin{array}{c}= \omega_2 \\ \overline{[W-(\omega_2)]c2[W]} \\ = \omega_1\end{array}}{[W-(\omega_1)]c1[W]}$$

The maximum WCET is taken as $L = max(\omega_1, \omega_2, \omega_3, \omega_4)$ and is used to arrive at the overall pre-condition.
Every component updates local variables on it's local stack and any value which is to be passed to the caller is updated in the caller's stack. It should be noted that all the components execute on a common lockstep.

These functions along with the Gate model are called in the *top* function and they are iteratively executed until one train circle is complete. The Real-time Hoare rules defined above are sound if the SMT encodings derived in this manner using these rules satisfy the system contracts such that they meet the soundness criteria as per Definition 2. This is guaranteed when all the pre and post conditions are satisfied for a given program by SMT solver. Since it is difficult to theoretically prove all the rules, given the system complexity, that is why, we choose Z3®SMT solver for dynamically proving all these properties. Once all the properties are proven, we compare the traces of Simulink®with the SMT solver traces. If we have same traces over a given lockstep value, this means that, these properties should pass in Simulink®model as well, and it guarantees that both Synch-C and Simulink®model behave in a similar fashion.

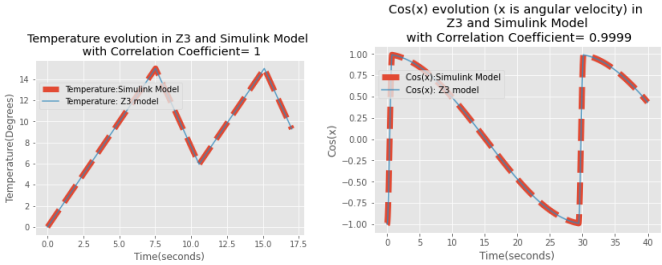
TABLE (I) Z3 modeling for Benchmarks

S.No.	Benchmark	Lockstep (sec)	WCET
1	Train Gate Controller	0.125	118144 cycles
2	Switch Tank Controller	0.02	33078 cycles
3	Nuclear Temperature Controller	0.01	43000 cycles
4	Steering Wheel Controller	0.001	1844430 cycles



(a) Train position evolution.

(b) Evolution of water level in first tank.



(c) Evolution of nuclear reactor temperature.

(d) Evolution of cos(x).

Fig. (5) Figures showing correlation of evolution (in Z3 and Simulink®) of: Train position in Figure 5a, water level in Figure 5b, temperature in Figure 5c and cosine of angular velocity in Figure 5d.

VII. Benchmarks Tested

We implemented our methodology on four benchmarks which are presented below. The WCET and lockstep value for the properties proven in Z3-SMT®model of these benchmarks is presented in Table I. In all the benchmarks described, `pre` means `Pre condition`, `Inv` means`Invariant` and `post` means `post condition`.

A. Train Gate Controller

This is our running example. The Simulink®model is shown in Figure 3c. The train movement is controlled by the ODE specified in Equation 1. The gate motion is follows the ODE described in Equation 2.

TABLE (II) Hoare Logic based Train Gate Controller timing Properties.

S.No.	Property	Conditions
C_{S1}	Train completes one circle in 118144 cycles	Pre: $y\_out == 0 \wedge up == 1 \wedge down == 0$. Inv: $y\_out \geq 0$. Post:$y\_out == 25 \wedge up == 1 \wedge down == 0 \wedge cycles \leq 118144$.
C_{S2}	Whenever ($Down == 1$) and train is moving then gate eventually opens ($up == 1$) in 110000 clock cycles	Pre: $y\_out == 0 \wedge Up == 0 \wedge Down == 1$. Inv: $y\_out \geq 0$. Post:$y\_out \geq 15 \wedge y\_out < 25 \wedge Up == 1 \wedge Down == 0 \wedge cycles \leq 110000$.

TABLE (III) Train and Gate Hoare Logic Properties

S.No.	Property	Conditions
Hoare logic properties for Train Component		
C_{T1}	Train starts and covers 25m in 118144 cycles	Pre: $y\_out == 0$. Inv: $y\_out \geq 0$.Post: $y\_out == 25 \wedge cycles \leq 118144$
C_{T2}	If train is between 0 and 15m then within 110000 cycles it's position will be 23m.	Pre:$0 \leq y\_out \leq 15$. Inv:$y\_out \geq 0$. Post: $cycles \leq 110000 \wedge y\_out == 23$.
Hoare logic properties for Gate Component		
C_{G1}	Gate closes in 118144 cycles.	Pre: $up == 1 \wedge down == 0 \wedge x\_out == 10$. Inv: $up \neq down$. Post: $up == 0 \wedge down == 1 \wedge cycles \leq 118144 \wedge x\_out = 0$
C_{G2}	When gate is down it opens in 110000 cycles.	Pre: $up == 0 \wedge down == 1 \wedge x\_out == 0$. Inv: $up \neq down$. Post: $up == 1 \wedge x\_out == 10 \wedge down == 0 \wedge cycles \leq 110000$
x_out and y_out is Gate and Train Position respectively.		

We statically compute the timing of every function/loop/-condition/assignment statement in the program which finally lead to overall WCET. The overall time is then used in a given property to be justified and is used in the post-condition. For Example, WCET for the Train to complete one circle is obtained as 118144 clock cycles. The pre and post conditions for this property are listed in Table II, which say that gate is initially open and the train is at the initial position. On completion of one circle the train should have covered 25 m of distance and the gate position should be open. The invariant condition in this case is the train and gate position is always ≥ 0. It should be noted that it is a system level contract which is refined by it's component level contracts, and is a synchronous parallel composition of those. Hence, Hoare property $C_{S1} = C_{T1} \parallel C_{G1}$. C_{T1} and C_{G1} are listed in Table III. The validity of composition is correct as described in Section III and Section IV. Similarly, the second property we verified says that the gate opens in 110000 clock cycles, since the time it was fully closed. In order to guarantee that the properties passing in Z3 solver should also behave the same way in Simulink®model, we compare the traces of Z3 model with Simulink®model. If traces are same then the properties should be passing in Simulink®model as well. Figure 5a shows the trace of the train position generated by Simulink®and Z3 solver. We found similar traces for both the models on a lockstep of 0.125 sec, with correlation coefficient of 1. This means that the real time Hoare logic based properties also pass in Simulink®. This also proves the efficacy of our approach.

B. Switch Tank Controller

A switch tank system consist of two water tanks [29]. Water leaks from both the tanks at a constant rate. Water level in these tanks is represented by continuous variables x_1 and x_2 respectively. The job of the controller is to keep water level above a threshold level which we have taken as 0.25m for both the tanks. Whenever water level goes below threshold value(which is taken as 0.25 m), the inflow is switched towards it, at this time, water level in second tank should be above the

upper threshold value of 1m. The ODE for water flow when in first tank is given in Equation 9, while for the second tank is given in Equation 10.

$$f(x_1) \stackrel{def}{=} \left[\dot{x_1}\right] = \begin{bmatrix} w - v_1 \\ -v_2 \end{bmatrix} \tag{9}$$

$$f(x_2) \stackrel{def}{=} \left[\dot{x_2}\right] = \begin{bmatrix} w - v_2 \\ -v_1 \end{bmatrix} \tag{10}$$

Here, w is constant flow of water into the tanks and is taken as one, while, v_1 is rate of water outflow from the first tank and v_2 is rate of water outflow from the other tank. We have taken v_1 = 0.6 while v_2 = 0.5. The WCET for this benchmark is 33078 cycles over at a lockstep of 0.02 seconds. The timing properties verified based on Hoare logic are described in Table IV. The first property in the Table is a composition of properties $CT0_1$ and $CT1_1$. The individual properties are refinement of system level property ST_1. Similarly, ST_2 is satisfied by parallel composition of $CT0_2$ and $CT1_2$. Traces of Z3 and Simulink®model is shown in Figure 5b. Similar trace shows that the properties are proven for Simulink®model as well.

TABLE (IV) Hoare Logic based Switch Tank Controller timing Properties.

S.No.	Property	Conditions
ST_1	Whenever $CT_0 == 0$ and $CT_1 == 1$,then $CT_0 == 1$ within 33078 clocks.	Pre: $CT_1 == 1 \wedge CT_0 == 0 \wedge x_1 == 1 \wedge x_0 == 0.2$. Inv: $0.25 \leq x_1 \leq 1 \wedge 0.25 \leq x_0 \leq 1$. Post: $CT_1 == 0 \wedge CT_0 == 1 \wedge x_0 \geq 1 \wedge x_1 \leq 0.25 \wedge cycles \leq 33078$
ST_2	Whenever $CT_0 == 1$ and $CT_1 == 0$, then $CT_1 == 1$ within 33078 clocks.	Pre: $CT_0 == 1 \wedge CT_1 == 0 \wedge x_0 == 1 \wedge x_1 == 0.2$ Inv: $0.25 \leq x_1 \leq 1 \wedge 0.25 \leq x_0 \leq 1$. Post: $CT_1 == 1 \wedge CT_0 == 0 \wedge x_0 \leq 0.25 \wedge x_1 \geq 1 \wedge cycles \leq 33078$.
x_0 and x_1 are water levels of $Tank_1$ and $Tank_2$ respectively.		

C. Steering Wheel controller

This benchmark is taken from [30]. The steering wheel of an autonomous vehicle needs to be maintained within the upper half plane as described in [30]. The controller of steering wheel is an example of sliding mode control without chattering. The angular position of the steering wheel is represented by variable x while the angle of the steering wheel is initially at $\frac{\pi}{2}$ radians.Following standard convention, clockwise movement of the steering wheel is considered negative and anticlockwise as positive. Whenever the steering wheel rotation reaches the left limit which is indicated by the condition cos(x) $\leq$ -0.99, the $turn$ is set to 1, the controller produces a clockwise angular velocity (dr) of -4 rad/sec. Similarly, when the right limit is detected which is indicated by the condition: $cos(x) \geq 0.99$, the angular velocity (dr) selected is 0.1 rad/sec. The plant model outputs the angular position of the steering wheel indicated by x and it's cosine which is horizontal projection of rotation. The ODE for steering wheel movement is given in Equation 11. The system level properties are composition of component level properties, if they respect the criteria mentioned in Section III and Section IV. In this case $SW1 = ST_1 \parallel STC_1$ and $SW2 = ST_2 \parallel STC_2$. We also compared the trace from Simulink®and Z3 solver and we found the trace values to be similar for both of the properties on the given lockstep (which is 0.001 and is base don ribustness criteria [27]). One such trace is shown in Figure 5d which compares $cos(x)$ trace. Since properties are valid for both of the traces, this benchmark as a result also confirm the efficacy of our approach.

$$f(x) \stackrel{def}{=} \left[\dot{x}\right] = \left[dr\right] \tag{11}$$

The timing properties based on Hoare Logic are shown in Table VI and are justified using these values.

TABLE (V) Component level Tank Properties for Switch Tank system

S.No.	Property	Conditions
Hoare logic properties for Tank1 Component		
$CT0_1$	When Tank1 is empty it becomes full in 33078 clocks	Pre: $X_0 \leq 0.25$. Inv: $0.25 \leq 1$. Post: $x_0 \geq 1 \wedge cycles \leq 33078$.
$CT0_2$	When Tank1 is full it becomes empty in 33078 clocks	Pre: $X_0 \geq 1$. Inv: $0.25 \leq 1$. Post: $x_0 \leq 0.25 \wedge cycles \leq 33078$.
Hoare logic properties for Tank2 Component		
$CT1_1$	When Tank2 is full it becomes empty in 33078 clocks	Pre: $x_1 \geq 1$. Inv: $0.25 \leq 1$. Post: $x_1 \leq 0.25 \wedge cycles \leq 33078$.
$CT1_2$	When Tank2 is empty it becomes full in 33078 clocks	Pre: $X_1 \leq 0.25$. Inv: $0.25 \leq 1$. Post: $x_1 \geq 1 \wedge cycles \leq 33078$.

TABLE (VI) Hoare Logic properties for steering wheel controller.

S.No.	Property	Conditions
$SW1$	Whenever $turn == 1$ and $cos(x) \leq -0.99$ then $cos(x) \geq 0.99$ in 1844430 clock cycles.	Pre:$turn == 1 \wedge cos(x) \leq -0.99$. Inv:$cos(x) \leq 0.99 \wedge cos(x) \geq -0.99$ Post: $cos(x) \geq 0.99 \wedge cycles \leq 1844430$.
$SW2$	Whenever $turn == 0$ and $cos(x) \geq 0.99$ then $cos(x) \leq -0.99$ in 11000 cycles.	Pre: $turn == 1 \wedge cos(x) \geq 0.99$. Inv: $cos(x) \leq 0.99 \wedge cos(x) \geq -0.99$. Post: $cos(x) \leq -0.99 \wedge cycles \leq 11000$

D. Nuclear Temperature Controller

This benchmark is presented in [31]. There are two control rods in a nuclear reactor which act as coolants. The coolant temperature must be kept within the range [θ_m, θ_M]. When the temperature reaches a maximum value of θ_M, the tank is cooled with one of the rods and only that rod should be selected for which the time elapsed since it was last used $\geq$ T time units. The ODE for rate change of temperature and the time related to the use of control rods is described in Equation 12. Here v_i represents the rate change of temperature in the nuclear reactor, such that whenever $i = 1$, then rod_1 is selected and if $i = 2$, then rod_2. Finally, the temperature in the nuclear reactor rise at the rate v_3(here i=3) and is given as $\dot{\theta}$. The WCET of the benchmark is 43000 clock cycles. The timing properties verified based on Hoare logic are described in Table VIII. The system level properties

TABLE (VII) Component level Properties for Steering wheel system

S.No.	Property	Conditions
Hoare logic properties for Steering Component		
ST_1	Time between $cos(x) \leq$ -0.99 to $cos(x) \geq 0.99$ is 1844430 cycles	Pre: $cos(x) \leq -0.99$. Inv: $0.99 \leq cos(x) \geq -0.99$ Post: $cos(x) \geq 0.99 \wedge cycles \leq 1844430$.
ST_2	Time between $cos(x) \geq 0.99$ to $cos(x) \leq -0.99$ is 11000 cycles	Pre: $cos(x) \geq 0.99$. Inv: $0.99 \leq cos(x) \geq -0.99$. Post:$cos(x) \leq -0.99 \wedge cycles \leq 11000$
Hoare logic properties for Controller Component		
STC_1	Time between $Turn = 1$ and $Turn = 0$ is 1844430 cycles	pre: $Turn == 1$. Inv: $\neg(Turn == 0 \wedge Turn == 1)$. Post: $Turn == 0 \wedge cycles \leq 1844430$.
STC_2	Time between $Turn = 0$ and $Turn = 1$ is 11000 cycles.	Pre: $Turn == 0$. Inv: $\neg(Turn == 0 \wedge Turn == 1)$. Post: $Turn == 1 \wedge cycles \leq 11000$

are listed in Table VIII are a synchronous composition of component level properties described in Table IX. The trace equivalence between Simulink®and Z3 model is established with a correlation co-efficient of 1 is shown in Figure 5c, which indicates the efficacy of our approach.

$$f(\theta, x_1, x_2) \stackrel{def}{=} \begin{bmatrix} \dot{\theta} \\ \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} v_i \\ 1 \\ 1 \end{bmatrix} \tag{12}$$

TABLE (VIII) Hoare Logic based Nuclear Temperature controller timing properties.

S.No.	Property	Conditions
1	Whenever rod_2 is selected then temperature in the nuclear reactor drops below 6 Degrees within 40200 clock cycles	Pre: $sel == 2 \wedge x2 > 6 \wedge \theta \geq 15$ Inv: $x_1 \wedge x_2 \geq 0 \wedge \theta \geq 0$ Post:$\theta < 6 \wedge x_1 > 6 \wedge x_2 == 0 \wedge cycles \leq 40200$.
2	Whenever rod_1 is selected then temperature in the nuclear reactor drops below 6 Degrees within 40200 clock cycles	Pre: $sel == 1 \wedge x1 > 6 \wedge \theta \geq 15$ Inv: $x_1 \wedge x_2 \geq 0 \wedge \theta \geq 0$ Post:$\theta < 6 \wedge x_2 > 6 \wedge x_1 == 0 cycles \leq 40200$

TABLE (IX) Component level Nuclear temperature controller Properties.

S.No.	Property	Conditions
Hoare logic properties for temperature Component		
T_1	When rod_2 counter $x_2 > 6$, then it is reset in 40200 clocks	pre: $x_2 > 6$. Inv: $x_2 \geq 0$. Post: $x_2 == 0 \wedge cycles \leq 40200$.
T_2	When rod_1 counter $x_1 > 6$, then it is reset in 38000 clocks	pre: $x_1 > 6$. Inv: $x_1 \geq 0$. Post: $x_1 == 0 \wedge cycles \leq 38000$.
Hoare logic properties for controller Component		
CT_1	When rod_2 is selected, then it is reset in 40200 clocks	Pre: $sel == 2$. Post : $sel == 0 \wedge cycles \leq 40200$.
CT_2	When rod_1 is selected, then it is reset in 40200 clocks	Pre: $sel == 1$.Post: $sel == 0 \wedge cycles \leq 40200$.

VIII. Related work

Model checking is about the reachability analysis of hybrid systems and lot of work has been done in this regard. For instance, work done by [32] employs a rule based approach, in which a rule-based logical model of the controller is obtained from the Unifying Modelling Language (UML) based state machine diagram. Then the static verification of the model is done by plugging it to a model checker. Once all the properties are justified, the rule based model is transformed into a synthesizable model to be prototyped in Field Programmable Gate Array (FPGA) device. The type of properties considered are safety and liveness properties. Another work [33] which uses Z language constructs and prototype verification system (PVS) as formal verification tools, for doing the static verification of CPS applications which are modeled using UML. Another work [34] which does contract based formal verification of intelligent hybrid systems modeled in Simulink®. To achieve this the Simulink®models are transformed into differential dynamic logic, the deductive formal verification of such model is done using interactive theorem prover called KeYmaera X. Work done in [35] provides a framework for compositional verification of Simulink®models with interacting components. Many works focusing on *assume-guarantee* based reasoning of contracts [10]–[14]. But none of the works focus on real time contract based reasoning for hybrid systems. Secondly, majority of the works described on static reasoning of contracts use LTL based contract verification. It is worth noting that static reasoning of contracts is not possible using standard LTL (described in Section II). Any LTL property cannot reason about the first and last state of any program code, but still the property holds on the stack trace of the code snippet. Such assumptions and guarantees on any program can be very well justified by using Hoare logic [3] not by using standard LTL. We claim that our technique of static reasoning using Hoare logic based real time contracts is more sound than the existing ones. It not only guarantees the safety-critical behavior of hybrid systems along with reachability and liveness guarantees, but it also has a sound reasoning capability of the program stack trace.

IX. Conclusion

We have presented a novel real time Hoare logic contract based reasoning technique for hybrid system models. None of the works presented so far have used WCET based real-time Hoare rules to derive SMT encodings based on Hoare logic contracts. Contrasting existing works, we have not used LTL to justify the system contracts as they are not fit to do so, rather we have used hoare logic based contracts, which effectively reason the behavior of hybrid system models. Our real-time Hoare logic rules are also sound since all the properties based on them are justified by the Z3®SMT solver.

REFERENCES

[1] T. Liebrenz, P. Herber, and S. Glesner, "A service-oriented approach for decomposing and verifying hybrid system models," in *Formal Aspects of Component Software*, F. Arbab and S.-S. Jongmans, Eds. Cham: Springer International Publishing, 2020, pp. 127–146.

[2] L. Aştefănoaei, S. Bensalem, and M. Bozga, *A Compositional Approach to the Verification of Hybrid Systems*. Cham: Springer International Publishing, 2016, pp. 88–103. [Online]. Available: https://doi.org/10.1007/978-3-319-30734-3_8

[3] B. Meyer, "Applying'design by contract'," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[4] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming dr. frankenstein: Contract-based design for cyber-physical systems," *European journal of control*, vol. 18, no. 3, pp. 217–238, 2012.

[5] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. G. Larsen, "Contracts for system design," Ph.D. dissertation, Inria, 2012.

[6] W. Damm, H. Hungar, B. Josko, T. Peikenkamp, and I. Stierand, "Using contract-based component specifications for virtual integration testing and architecture design," in *2011 Design, Automation & Test in Europe*. IEEE, 2011, pp. 1–6.

[7] S. Quinton and S. Graf, "Contract-based verification of hierarchical systems of components," in *2008 Sixth IEEE International Conference on Software Engineering and Formal Methods*. IEEE, 2008, pp. 377–381.

[8] A. Benveniste, B. Caillaud, and R. Passerone, "A generic model of contracts for embedded systems," *arXiv preprint arXiv:0706.1456*, 2007.

[9] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, "Multiple viewpoint contract-based specification and design," in *International Symposium on Formal Methods for Components and Objects*. Springer, 2007, pp. 200–225.

[10] A. Cimatti and S. Tonetta, "Contracts-refinement proof system for component-based embedded systems," *Science of computer programming*, vol. 97, pp. 333–348, 2015.

[11] M. Lindgren, "Practical verification of stateful embedded c code using finite state machines and vcc," 2020.

[12] W. Dong, Z. Chen, and J. Wang, "A contract-based approach to specifying and verifying safety critical systems," *Electronic Notes in Theoretical Computer Science*, vol. 176, no. 2, pp. 89–103, 2007.

[13] I. Sljivo, O. Jaradat, I. Bate, and P. Graydon, "Deriving safety contracts to support architecture design of safety critical systems," in *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, 2015, pp. 126–133.

[14] C. Nandi, "Contracts for real-time, safety critical systems," Tech. Rep., 2014.

[15] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[16] S. Andalam, P. Roop, A. Girault, and C. Traulsen, "Pret-c: A new language for programming precision timed architectures," Ph.D. dissertation, INRIA, 2009.

[17] I. Liu, J. Reineke, D. Broman, M. Zimmer, and E. A. Lee, "A pret microarchitecture implementation with repeatable timing and competitive performance," in *2012 IEEE 30th international conference on computer design (ICCD)*. IEEE, 2012, pp. 87–93.

[18] Q. Carbonneaux, J. Hoffmann, and Z. Shao, "Compositional certified resource bounds," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2015, pp. 467–478.

[19] R. Alur, "Timed automata," in *International Conference on Computer Aided Verification*. Springer, 1999, pp. 8–22.

[20] P. Nuzzo, "Compositional design of cyber-physical systems using contracts," Ph.D. dissertation, UC Berkeley, 2015.

[21] P. Nuzzo, M. Lora, Y. A. Feldman, and A. L. Sangiovanni-Vincentelli, "Chase: Contract-based requirement engineering for cyber-physical system design," in *2018 Design, Automation and Test in Europe Conference and Exhibition*, 2018, pp. 839–844.

[22] A. Permenev, D. Dimitrov, P. Tsankov, D. Drachsler-Cohen, and M. Vechev, "Verx: Safety verification of smart contracts," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1661–1677.

[23] P. Nuzzo, J. B. Finn, A. Iannopollo, and A. L. Sangiovanni-Vincentelli, "Contract-based design of control protocols for safety-critical cyber-physical systems," in *2014 Design, Automation and Test in Europe Conference and Exhibition*, 2014, pp. 1–4.

[24] T. Dreossi, A. Donzé, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," *Journal of Automated Reasoning*, vol. 63, no. 4, pp. 1031–1053, 2019.

[25] D. Gurov and J. Westman, "A hoare logic contract theory: An exercise in denotational semantics," in *Principled Software Development*. Springer, 2018, pp. 119–127.

[26] surinder sood, "train gate controller code," https://bitbucket.org/surindersood/z3code/src/master/tgt.py, 2022, [Online;].

[27] G. E. Fainekos and G. J. Pappas, "Robust sampling for mitl specifications," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2007, pp. 147–162.

[28] S. Sood, A. Malik, and P. Roop, "Robust design and validation of cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 6, pp. 1–21, 2019.

[29] J. Lygeros, G. Pappas, and S. Sastry, "An introduction to hybrid system modeling, analysis, and control," *Preprints of the First Nonlinear Control Network Pedagogical School*, pp. 307–329, 1999.

[30] J. W. Ro, A. Malik, and P. Roop, "A compositional semantics of simulink/stateflow based on quantized state hybrid automata," in *Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design*, 2019, pp. 1–11.

[31] R. Alur, C. Courcoubetis, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," in *11th International Conference on Analysis and Optimization of Systems Discrete Event Systems*. Springer, 1994, pp. 329–351.

[32] I. Grobelna, "Formal verification of control modules in cyber-physical systems," *Sensors*, vol. 20, no. 18, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/18/5154

[33] G. Magureanu, M. Gavrilescu, and D. Pescaru, "Validation of static properties in unified modeling language models for cyber physical systems," *Journal of Zhejiang University SCIENCE C*, vol. 14, no. 5, pp. 332–346, 2013.

[34] P. Herber, J. Adelt, and T. Liebrenz, "Formal verification of intelligent cyber-physical systems with the interactive theorem prover keymaera x." in *Software Engineering (Satellite Events)*, 2021.

[35] T. Liebrenz, P. Herber, and S. Glesner, "A service-oriented approach for decomposing and verifying hybrid system models," in *Formal Aspects of Component Software*, F. Arbab and S.-S. Jongmans, Eds. Cham: Springer International Publishing, 2020, pp. 127–146.