IEICE TRANS. INF. & SYST., VOL.E103–D, NO.4 APRIL 2020

PAPER

Model Checking of Real-Time Properties for Embedded Assembly Program Using Real-Time Temporal Logic RTCTL and Its Application to Real Microcontroller Software

Yajun WU[†], ***Nonmember and*** **Satoshi YAMANE**[†a)], ***Member***

SUMMARY For embedded systems, verifying both real-time properties and logical validity are important. The embedded system is not only required to the accurate operation but also required to strictly real-time properties. To verify real-time properties is a key problem in model checking. In order to verify real-time properties of assembly program, we develop the simulator to propose the model checking method for verifying assembly programs. Simultaneously, we propose a timed Kripke structure and implement the simulator of the robot's processor to be verified. We propose the timed Kripke structure including the execution time which extends Kripke structure. For the input assembly program, the simulator generates timed Kripke structure by dynamic program analysis. Also, we implement model checker after generating timed Kripke structure in order to verify whether timed Kripke structure satisfies RTCTL formulas. Finally, to evaluate a proposed method, we conduct experiments with the implementation of the verification system. To solve the real problem, we have experimented with real microcontroller software.

key words: *RTCTL, model checking, dynamic program analysis, timed Kripke structure, embedded system*

1. Introduction

In recent years, embedded systems have been widely used in autonomous car, national defense, medical equipment and IoT (Internet of Things). It is important to verify real-time properties for embedded systems, not only the logical validity. The real-time property of embedded system has to be tested extensively and validated because errors may lead to severe or even fatal events, as in the brake instructions of the autonomous car to the execution time are very important and must be executed within the deadline. Therefore, to verify the real-time properties is the key problem for the design of embedded real-time systems.

Recently software model checking [1] and program verification [2] have received widespread attention. S. Yamane and others have developed a verification system [3] for verifying embedded assembly programs. They generate Kripke structure by simulator (dynamic program analysis) and verifies stack overflow by CTL (Computational Tree Logic) model checking.

This paper mainly focuses on proposed the algorithm of verifying real-time properties, further the implementation of RTCTL model checker and experiments on real microcontroller software. To verify real-time properties of embedded systems, we verify assembly program rather than c program. First of all, interrupt processing can be performed for each one instruction. Second, the interface part with the hardware has the assembly description part. Then, it is possible to compute the execution time accurately. In this paper, we can verify real-time properties by proposing the timed Kripke structure. We develop the simulator to generate timed Kripke structure from assembly program and implement the simulator of the robot's processor to be verified. In our approach, to verify the real-time property of assembly program, we propose the timed Kripke structure which extends Kripke structure by includes the execution time. The property to be verified is described by RTCTL. We will describe model checking using RTCTL (Real-Time CTL) in order to verify whether the timed Kripke structure satisfies real-time properties. In this paper, we propose RTCTL model checking algorithm after generating timed Kripke structure. Also, we implement a model checker in order to verify whether the timed Kripke structure satisfies real-time properties. In our study, to evaluate our proposed method, we conducted experiments with the implementation of the verification system and we have experimented with real microcontroller software. To our knowledge, the model checker that developed this research is the first in the world to verify the real-time property of assembly program.

The paper is organized as follows: Section 2 provides related works about model checking. Section 3 provides a computational model of embedded assembly program used in this paper. Section 4 provides some details of the used temporal logic of Real-Time temporal logic (RTCTL), while Sect. 5 describes the proposed verification system based on the concepts introduced in the previous sections. Finally, Sect. 6 presents experiments with the implementation of the verification system to evaluate a proposed method, and Sect. 7 presents the conclusion of this work, pointing to the next step toward a better description of the model checker while generating timed Kripke structure in order to verify whether timed Kripke structure satisfies RTCTL formulas.

2. Related Works

B. Schlich studied model checking by using static program

Manuscript received June 20, 2019.
Manuscript revised September 11, 2019.
Manuscript publicized January 6, 2020.

[†]The authors are with Graduate School of Natural Science and Technology, Kanazawa University, Kanazawa-shi, 920–1192 Japan.
a) E-mail: syamane@is.t.kanazawa-u.ac.jp

DOI: 10.1587/transinf.2019EDP7172

analysis to verify assembly program, but real-time properties are not verified. B. Schlich can verify the hardware dependency problem by using the assembly program [4]. Jumpei Kobashi and others [5] proposed model checking the assembly code using SMT-Based BMC (Bounded Model Checking). And they proposed the parser and model converter for the assembly code. But the same with B. Schlich the real-time properties are not verified. Matthew Kuo and others [6] efficiently compute the WCRT using reachability to analysis the assembly code. However, the model is different from our study, Matthew Kuo and others generate TCCFG (Timed Concurrent Control Flow Graph) by static analysis from each assembly code, but our study proposes the timed Kripke structure and the purpose of our study is to verify the real-time property of computing the execution time of each assembly code. Matthew Kuo and others have not performed the model checking.

E. A. Emerson and others proposed RTCTL, and also developed the model checking algorithm for RTCTL [7]. But our semantics of RTCTL is quite different from E. A. Emerson's semantics. In E. A. Emerson's semantics, all transitions happen in one time unit. On the other hand, in our study, the execution time of each assembly instruction is assigned to each state. E. A. Emerson points out that the minor extension of the method can be applied to the Kripke structure in the case of nonnegative inter transition times, but differs from this paper in the following points.

- We show the concrete algorithm of expansion.
- We have implemented the algorithm.
- We have experimented with real microcontroller software.

R. Alur and D. L. Dill studied timed automata [8]. Timed automaton is an extension of a finite state automaton and is a model that describes the system by both discrete event and continuous time-lapse according to state transitions. On the other hand, in this study, we develop discrete-time timed Kripke structure for the execution time of assembly program. Our study is different from timed automata. The proposed structure deals with discrete time in our study, and we combine the execution time of each instruction of the assembly program to each state. Therefore, it is possible to verify the real-time property by model checking. Bouyer, P and others [9] have surveyed model checking the real-time systems by using time automaton. Previous studies using automata guarantees safety through the reachability analysis of real-time systems, but does not aim to verify the real-time property. And the benefit of our study for the discrete real-time system is that to check whether the system can be satisfies the real-time property. In our study, we proposed the discrete model of timed Kripke structure can be generated using dynamic analysis and the execution time of each assembly instruction code can be accurately computed by the dynamic analysis.

Mahshid Helali Moghadam and others [10] proposed WCRT (worst-case response time) analysis using simulation, but the experiment is not performed. On the ohter hand, it is different than computing the execution time of C code or assembly code. In our study, the execution time can be computed for each single assembly code by using dynamic analysis. The study of Mahshid Helali Moghadam and others is different from our study, where their study is estimated of the execution scenarios leading to the WCRT using reinforcement learning, but model checking is not performed.

Béchennec and others studied real-time model checking by using the model checker UPPAAL to compute the WCET, and analyze binary program based on an automatic method to compute a CFG [11]. Computing the WCET is reduced to a reachability problem, but in our study, we can verify various properties. On the other hand, computing the WCET does not assume interrupt handling, our study considers the interrupt handling.

S. Yamane and others have developed a verification system. The simulator generates Kripke structure, and model checker verifies CTL formulas. The simulator includes clock cycles while generating models, but real-time properties and execution time are not taken into consideration. S. Yamane and others have developed abstraction techniques by using DND (Delayed NonDeterminism) of the bit level. Our proposed method is quite different from [3] as follows: (1) Generating timed Kripke structure including execution time. (2) Using RTCTL formulas to verify real-time properties. (3) Proposing RTCTL model checking algorithm after generating timed Kripke structure.

We improve the simulator developed by S. Yamane and others. The simulator computes execution time and generates timed Kripke structure by the exhaustive breadth-first search of assembly program with dynamic program analysis. We implement the simulator based on the robot's processor to be verified and we implement to check the real-time property by RTCTL model checking.

3. Computational Model of Embedded Assembly Program

Kripke structure is often used in order to exhaustively search the state space of the model expressing the system when model checking is performed.

In our study, as a computation model, we define a timed Kripke structure which extends Kripke structure [12] by time function. This is different from E. A. Emerson's model.

Definition 1 (Timed Kripke structure): Timed Kripke structure is $M = (S, S_0, R, L, TM)$.

(1) A set of states S

(2) A set of initial states $S_0 \subseteq S$

(3) A transition relation $R \subseteq S \times S$

(4) A labelling function $L: S \rightarrow 2^{AP}$. L is a function which assigns to each state a set of atomic propositions. AP is atomic proposition.

(5) A time function $TM: S \rightarrow N$. TM is a function that assigns the execution time of each instruction to each state. Here N is any natural number.

```
void control_linetrace(void)
{
   float   u_l, u_r;
   tsensor = tsensor_get();
   ...
unsigned char tsensor_get(void)
{
   return  ~IO.PDR3.BYTE & 0x07;

}
```

(1) C code

```
_control_linetrace:
        PUSH.L     ER6
        PUSH.L     ER5
        PUSH.L     ER4
        JSR        @_tsensor_get:16
        MOV.B      R0L,@_tsensor:16
        ...
_tsensor_get:
        MOV.B      @65494:8,R0L
        NOT.B      R0L
        AND.B      #7,R0L
        RTS
```

(2) Assembly code

Fig. 1 An example of timed Kripke structure

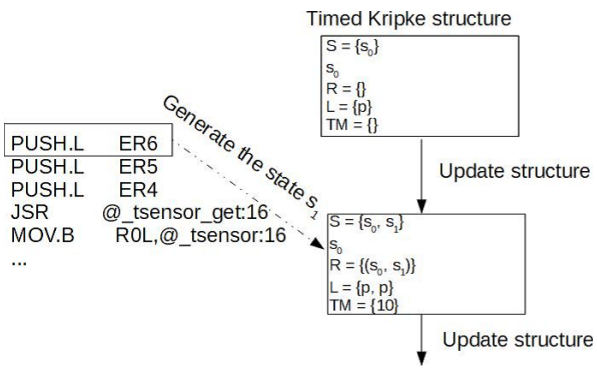

Fig. 2 An example of timed Kripke structure of assembly code

By adding the time function, it becomes possible to handle the execution time when modeling the assembly program. In order to compute accurate execution time, which enables verification of real-time property, it is not targeted for C program but for assembly program. First of all, interrupt processing can be performed for each one instruction. Second, the interface part with the hardware has the assembly description part. Then, it is possible to compute the execution time accurately. For example, the robot needs strict real-time properties between the sensor and the microcomputer. For C program, since the value of the sensor is written to the register of the microcomputer in real time, it is difficult to verify the real time property of the value of the sensor is written to the register. If it is an assembly program it can be accurately calculated every instruction. Figure 1 shows a typical microcontroller C program that identify black and white by sensor. Figure 2 shows an assembly program corresponding to the C program. The program is one of the programs used in the case study described in Sect. 6.

The C program in Fig. 1 that acquires values from the

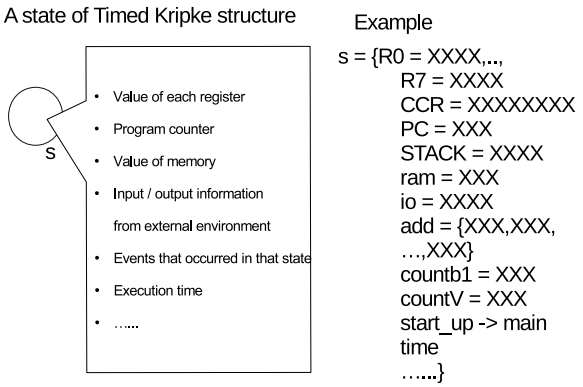

Fig. 3 A state of timed Kripke structure

sensor. When converting the program into an assembly program, it becomes an assembly program in Fig. 1. From the value of the register of the microcontroller, in order to verify whether or not the value of the sensor has been acquired in real time, it can not be verified with C program, but it can be verified with assembly program. It is possible to compute accurate execution time from the number of execution states where there is the number of execution states for each assembly instruction. For example, PUSH.L ER6 execution states is 8.

Then, the assembly program generates a timed Kripke structure as shown in Fig. 2 and a state of timed Kripke structure as shown in Fig. 3. At the top of right is the timed Kripke structure of the initial state in Fig. 2. Then the first instruction PUSH.L ER6 generates as a state, and the timed Kripke structure is updated.

4. RTCTL

In our study, RTCTL model checking inputs verification properties with real-time temporal logic RTCTL and performs model checking on timed Kripke structure generated by the simulator.

RTCTL (Real-Time Computational Tree Logic) is an extension of Computational Tree Logic (CTL), it enables inference about the time-critical accuracy of the program. Since RTCTL is an extension of CTL, the path is branched like a tree structure like CTL, and properties related to multiple paths can be described. The RTCTL formulas are described from temporal operators, path quantifiers, logical operators and a time constraint k. Temporal operators and path quantifiers are as follows.

- Temporal operator
  - Xp is mean that p holds at the next state. (Next state)
  - Fp is mean that p eventually holds. (In the future state)
  - Gp is mean that p holds on the entire subsequent path. (Globally)
  - pUq is mean that q holds at the current or a future state, and p has to hold until that state. (Until)

- Path quantifier
  - Ap is mean that p holds on all paths starting from the current state.
  - Ep is mean that there exists at least one path starting from the current state where p holds.

For model checking of real-time properties, we define Real-Time temporal logic (RTCTL). We define RTCTL based on that developed by E. A. Emerson [7].

Definition 2 (Syntax of RTCTL): Syntax of RTCTL formulas is defined as follows:

(1) Each atomic proposition AP is a RTCTL formula.

(2) If p, q are RTCTL formulas, p ∧ q and ¬p are RTCTL formulas.

(3) If p, q are RTCTL formulas, E(p U q), A(p U q) and EXp are RTCTL formulas.

(4) If p, q are RTCTL formulas and k is any natural number for execution time, E(p $U^{\leq k}$ q) and A(p $U^{\leq k}$ q) are RTCTL formulas.

Some other RTCTL formulas are defined as follows:

$AF^{\leq k}$ q = A(true $U^{\leq k}$ q) $EF^{\leq k}$ q = E(true $U^{\leq k}$ q)

$AG^{\leq k}$ p = ¬$EF^{\leq k}$¬p $EG^{\leq k}$ p = ¬$AF^{\leq k}$¬p

We propose the semantics of RTCTL as follows:

Definition 3 (Semantics of RTCTL): In the semantics of E(p $U^{\leq k}$ q) and A(p $U^{\leq k}$ q) is defined over timed Kripke structure M = (S, S_0, R, L, TM).

(1) E(p $U^{\leq k}$ q)

For a state sequence $s_0, s_1, \ldots, s_j, \ldots, s_i, \ldots$ in M, $\exists i$ $0 \leq i$, $s_i \models$ q and $\sum_{\alpha=0}^{i} TM(s_\alpha) \leq k$ and $\forall j\ 0 \leq j < i$, $s_j \models$ p.

(2) A(p $U^{\leq k}$ q)

For all state sequence $s_0, s_1, \ldots, s_j, \ldots, s_i, \ldots$ in M, $\exists i$ $0 \leq i$, $s_i \models$ q and $\sum_{\alpha=0}^{i} TM(s_\alpha) \leq k$ and $\forall j\ 0 \leq j < i$, $s_j \models$ p.

It is different from E. A. Emerson and others that the constraint time k is a fixed value, every transition takes one unit time for execution. But our study generates a state for each instruction in the assembly program and not only converts easily unit time to execution time for each state. It is necessary to concretize the execution time of each state in order to verify the real-time property by backward.

The challenge of our study to verify real-time properties accurately computing the execution time of each state. Therefore, our study uses dynamic program analysis to generate a timed Kripke structure that includes accurate execution time. We show our proposed algorithm in Sect. 5. It is different from E. A. Emerson and others that the execution time is computed backward in the algorithm and it is verified whether the time constraint is satisfied. The point that verification stops when time constraint k becomes less than 0 during verification is also different.

5. Verification System

The verification system is shown in Fig. 4.

First, we propose simulation and verification algorithm as shown in Algorithm 1. Algorithm 1 is defined based on the verification algorithm of S. Yamane and others [3]. The difference is as follows.

- The model checking is performed after generating the timed Kripke structure of simulator.
- Using RTCTL formulas to perform model checking of real-time properties.

The model checking algorithm of RTCTL formulas (e.g. E(p $U^{\leq k}$ q)) after generating the timed Kripke struc-

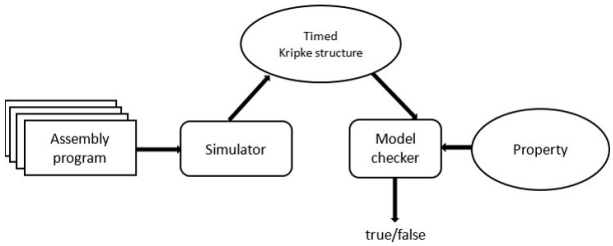

Fig. 4 Verification system

Algorithm 1 Algorithm of simulation and verification system

```
1 f(k)                                          // Property(RTCTL formula)
2 S := {s0}                                     // set of initial state
3 TM := ∅                                       // set of states time function
4 time(s0)                                      //execution time of state
5 funtion Main
6  while !end_flag do
7   s ← generated new state ∈ S
8   if decidable interrupt exist
9    INTERRUPTHANDLING(s)                       // generate state
10  else
11   EXECUTEINSTRUCTION(s)                      // generate state
12  end if
13 end while
14 ModelCheck(f(k))                             //perform model checking
15 end function
16
17 function INTERRUPTHANDLING(s)
18  for all i ∈ Interrupts do
19   if i is interruptible then
20    PC_i = VectorTable[i]                     // get vector address
21    s'.PC = PC_i                              // set PC_i to PC of s'
22    EXECUTEINSTRUCTION(s')                    // generate state
23    GlobalMaskBit_s' ← true                   //mask s'
24    InterruptFlag_s' ← false                  // clear flag of s'
25
26 function EXECUTEINSTRUCTION(s)
27  if s is the last state
28   then end_flag ← true
29  else
30   operation ← memory[s.PC]                  // get operation accroding to PC
31   s' ← execute(s, operation)                // generate a new state
32   ADDNEWSTATE (s, s')
33 end function
34
35 function ADDNEWSTATE(s, s')
36  S := S ∪ {s'}                              // add new state to S
37  R := R ∪ {(s, s')}                         // add new transition from s to s'
38  TM(s'):= TM(s') ∪ {time(s')}               // add execution time of s' to TM(s')
39 end function
```

ture is defined in Algorithm 2. In general, as timed Kripke structure has finite states, our approach is effective. But generation of timed Kripke structure does not stop for infinite state. It can not be verified in the infinite state. The verification system inputs assembly program. We generate states for the verification target and output a timed Kripke structure by the simulator. Simultaneously, model checking is performed by inputting RTCTL formula and timed Kripke structure.

5.1 Simulator

We extend the simulator developed by S. Yamane and others [3]. The simulator computes execution time and generates timed Kripke structure by the exhaustive breadth-first search of an assembly program with dynamic program analysis. The simulator parses program (assembly program), and stores each one in memory, and simulates it.

- First, if the current state is the last state, the end_flag is set to true.
- And if it is not the last state, the simulator can not execute interrupt processing when GlobalMaskBit is true. Interrupt Flag becomes true when interrupt processing occurs.
- When an interrupt occurs, interrupt processing is started, and the program counter (PC) at the top of the interrupt processing is obtained from the VectorTable.
- And an instruction is ececuted. Then the GlobalMaskBit is set to true and the Interrupt Flag is set to false.
- If there is no interrupt processing, execute the instruction as it is. When executing an instruction, if the generated state is the last state, end_flag is set to true.

The feature of our study is that the execution time of each instruction can be computed, and the timed Kripke structure is automatically generated considering the execution time. The state explosion is one of the challenges of model checking. In our study number of states can be reduced by using undefined values based on DND (Delayed NonDeterminism) proposed by B. Schlich and others [4].

DND processes non-deterministic values as they are (eg. External environment input values) or, if they can not do so, concretize them with fewer branches. The undefined values are used in our study will only concretize the necessary bits when concretizing. Unnecessary bits are abstracted as symbols, and branches can be greatly reduced. The occurrence of branching also occurs when concretizing undefined values. When register initialization or an input value from the external environment and so on does not know which value is 0 or 1 for a bit.

On the other hand, when it is necessary to calculate an abstracted value using undefined values (such as addition and subtraction) and the branching occurs when concretizing abstracted value. For example (Fig. 5), in the initial state of CCR (Condition code register indicates the internal state of the CPU), only the interrupt mask bit is 1 and the remaining 7 bits are undefined. Therefore, there are 128 patterns if the undefined values are not used, but there is only 1 pattern if the undefined values are used. The occurrence of an interrupt is also one of the causes of branching.

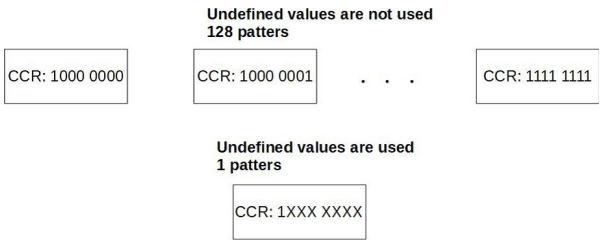

Fig. 5 The initial state of CCR

Next, the main function (simulator) of Algorithm 1 is explained as follows.

(1) First, we determine whether the interrupt handling is present (line 9). When the executable interrupt handling exists, interrupt handling is executed (line 10). After that, when interrupt handling is executed, the state is generated (line 23).

(2) Next, if the executable interrupt handling does not exist, the instruction is executed, then a state is generated (line 12).

(3) Then RTCTL model checker is performed after all the states are generated (line 16).

5.2 RTCTL Model Checking

We describe the RTCTL model checking procedures in Algorithm 2 as the follows:

- First, we verify from the subformula f of length 1 to the length of the RTCTL formula f_0. Subformula $p \wedge q$, $\neg p$ and EX p is the same as E. A. Emerson [7], but A(p $U^{\leq k}$ q) and E(p $U^{\leq k}$ q) are different.
- Then, for example, function ModelCheck_E(p $U^{\leq k}$ q)(p, q, k, f(k)). The simulator generates last state s that satisfies q, add it to S and f(k) is added in L(s). Then we initialize N and H.
- Until T is empty, we select state s from T and remove it from T at the same time. Then we update N with H.
- Next, for all state t such that satisfies R(t, s) (line 65), R(t, s) means state s with transition relation source state t. Then, in case of p added in L(t) and f(k) added in L(t), time constraint is satisfied (line 66). After that H is updated by calculating the execution time of N and state s, and update N (line 67). Next, we add state t to S (line 69) and f(H) added in L(t) (line 68).
- When the time constraint is not satisfied but S is not empty (line 70), it still another path which has not been checked. After that acquire H from the added label f(H) (line 72).
- Then R(t, s) is empty and S is not empty (line 76). It also still has another path which has not been checked.
- Otherwise, there is no transition from source state s (line 80) and S is empty. Therefore, H is updated by

Algorithm 2 Algorithm of RTCTL Model Checking

```
1 p, q                                                    // Formula
2 f(k)                                          // Property(RTCTL formula)
3 S := {s0}                                               // set of initial state
4 R := ∅                                          // set of relations between states
5 TM := ∅                                                 // set of states time function
6 function ModelCheck(f(k))
7  N := k H := k                                          // initialize N and H
8 for i := 1 to length(f0) do begin
9  for each subformula f of f0 of length i do begin
10   case struture of f is of the form
11    p ∧ q : for each s ∈ S do
12          if p ∈ L(s) and q ∈ L(s) then add p ∧ q to L(s)
13    ¬ p :for each s ∈ S do
14          if p ∉ L(s) then add ¬p to L(s)
15    EX p : for each s ∈ S do
16          if p ∈ L(w) for some R-successor w of s then add EX p to
L(s)
17    A(p U^≤k q) : ModelCheck_A(p U^≤k q)(p, q, k, A(p U^≤k q))
18    E(p U^≤k q) : ModelCheck_E(p U^≤k q)(p, q, k, E(p U^≤k q))
19   end case
20  end for
21 end for
22 if f0 ∈ L(s) for some s ∈ S and f(H) ∈ L(s0) and N ≥ 0
23   then Output(S, R, true)
24   else Output(S, R, false)
25 end if
26 function ModelCheck_A(p U^≤k q)(p, q, k, f(k))
27 S := { s | q ∈ L(s) }
28 for s ∈ S do L(s) := L(s) ∪ f(k)
29 while S ≠ ∅ do
30  choose s ∈ S
31  S := S \ {s}                                          // remove s from S
32  N := H                                                // update N by H
33  for all t such that R(t, s) do
34   if f(k) ∉ L(t) and N ≥ 0 and p ∈ L(t) then
35    H := N - TM(s)                                      // update H
36    L(t) := L(t) ∪ {f(H)}
37    S := S ∪ {t}                                        // add t to S
38   else if q ∈ L(t) and S ≠ ∅ then
39     break
40   else L(s0) := ∅
41    goto OUT
42  end for all
43  if S ≠ ∅ and R(t, s) = ∅ then                  //s is last state on some
44   choose y ∈ S                    // path, and other path has not checked
45   choose f(H) from L(y)
46   H := k                                               // updata H by f(k)
47  else if S = ∅ and R(t, s) = ∅
48   H := N - TM(s)              // s is last state and all path has checked
49     if H ≥ 0 then
50      L(s) := L(s) ∪ f(H)
51      break
52     end if
53  end if
54 end while
55 OUT :
56 N := H                                                 // update N by H
57 end function
                                                          // continue with Algorithm 2
58 function ModelCheck_E(p U^≤k q)(p, q, k, f(k))
59 S := { s | q ∈ L(s) }
60 for s ∈ S do L(s) := L(s) ∪ f(k)
61 while S ≠ ∅ do
62  choose s ∈ S
63  S := S \ {s}                                          // remove s from S
64  N := H                                                // update N by H
65  for all t such that R(t, s) do
66   if f(k) ∉ L(t) and N ≥ 0 and p ∈ L(t) then
67    H := N - TM(s)                                      // update H
68    L(t) := L(t) ∪ {f(H)}
69    S := S ∪ {t}                                        // add t to S
70   else if N < 0 and S ≠ ∅ then
71    choose y ∈ S                    // other path has not checked
72    choose f(H) from L(y)
73    H := k                                              // updata H by f(k)
74   end if
75  end for all
76  if S ≠ ∅ and R(t, s) = ∅ then                  //s is last state on some
77   choose y ∈ S                    // path, and other path has not checked
78   choose f(H) from L(y)
79   H := k                                               // updata H by f(k)
80  else if S = ∅ and R(t, s) = ∅
81   H := N - TM(s)              // s is last state and all path has checked
82     if H ≥ 0 then
83      L(s) := L(s) ∪ f(H)
84      break
85     end if
86  end if
87 end while
88 N := H                                                 // update N by H
89 end function
```

calculating the execution time of N and state s (line 81). Then in the case of H ≥ 0, we add f(H) to L(s) and leave the loop (line 84).

- Finally, we update N with H (line 88). Subsequently, if f(H) is added in $L(s_0)$ and time constraint is satisfied (line 22), then verification result is true (line 23).

We use |f| to denote the length of RTCTL formula f(k) (f(k) is the RTCTL formula, like E(p $U^{\leq k}$ q)). The calculation of |f| is the same as that of E. A. Emerson and others [7]. When |f′| is the length of the CTL formula f′ obtained from f by deleting time constraint, |f| = |f′| + c and c is the sum of the lengths of the bit strings representing in binary the time constraint of f. Thus, we see that the complexity of executing the entire procedure f(k) model check is O(|f| (|S| + |R|)).

As shown in the following example, if the state t transition to the state s is exists, where $E(pU^{\leq k}q)_{state\ s}$ holds true at the state s and it can be expressed by mean of a fixpoint as follows:

$$E(pU^{\leq k}q)_{state\ s} = q_{state\ s} \vee (p \wedge EXE(pU^{\leq k}q))_{state\ t}$$

As shown in Fig. 6, verification is performed by backward according to the algorithm. The pre-state t of state s can be labeled and the real-time property can be verified by computing the execution time to the state t. Using Tarski fixpoint theorem [13] we can prove the correctness of the proposed algorithm.

Definition 4 (Tarski fixpoint theorem): Let state sets of functional τ: $2^S \rightarrow 2^S$ be given. Then we say that $\tau(Y)$ is monotonic provided that $P \subseteq Q$ implies $\tau(P) \subseteq \tau(Q)$. We identify a predicate with the set of states which make it true. Thus, false, which corresponds to the empty set, is the bottom element, true, which correspond to S is the top element.

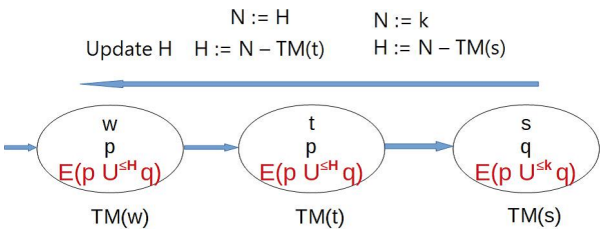

Fig. 6 verfication transition

(1) least fixpoints define:

$\mu Y.\tau(Y) = \cap\{Y : \tau(Y) = Y\} = \cap\{Y : \tau(Y) \subseteq Y\}$.

(2) greatest fixpoints define:

$\nu Y.\tau(Y) = \cup\{Y : \tau(Y) = Y\} = \cup\{Y : \tau(Y) \supseteq Y\}$.

(3) least fixpoints calculation:

$\mu Y.\tau(Y) = \cup_i \tau^i$ (false). i is at most $|S|$.

(4) greatest fixpoints calculation:

$\nu Y.\tau(Y) = \cap_i \tau^i$ (true). i is at most $|S|$.

Next, RTCTL formulas are represented as fixpoints.

Definition 5: (Fixpoints representation of RTCTL formulas) Fixpoints representation of $E(pU^{\leq k}q)$ and $A(pU^{\leq k}q)$ defined by timed Kripke structure M = (S, S_0, R, L, TM).

(1) Fixpoints representation of E(p $U^{\leq k}$ q):

For $E(pU^{\leq k}q)_{state\ s} = \mu Z.q \vee (p \wedge EXZ)_{state\ t}$, the following equation holds true, $E(pU^{\leq k}q)_{state\ s} = q_{state\ s} \vee (p \wedge EXE(pU^{\leq k}q))_{state\ t}$

Here, p ∧ EXE(p $U^{\leq k}$ q) = E(p $U^{\leq r}$ q), where r = k - TM(s_q) and state s_q is satisfied q.

(2) Fixpoints representation of A(p $U^{\leq k}$ q):

For $A(pU^{\leq k}q)_{state\ s} = \mu Z.q \vee (p \wedge AXZ)_{state\ t}$, the following equation holds true, $A(pU^{\leq k}q)_{state\ s} = q_{state\ s} \vee (p \wedge AXA(pU^{\leq k}q))_{state\ t}$

Here, p ∧ AXA(p $U^{\leq k}$ q) = A(p $U^{\leq r}$ q), where r = k - TM(s_q) and state s_q is satisfied q.

There are least fixpoints, and it is verified backward according to the algorithm, and it is verified that real-time property can not be satisfied if the time constraint k becomes less than 0, and the algorithm terminates. The model checking algorithm of E(p $U^{\leq k}$ q) and A(p $U^{\leq k}$ q) are correct according to fixpoints. The proof of E(p $U^{\leq k}$ q) is as follows. The proof of A(p $U^{\leq k}$ q) is the same as E(p $U^{\leq k}$ q) and is omitted here.

Theorem 1: Let M = (S, S_0, R, L, TM) be a structure and s ∈ S, Then M,s ⊨ E(p $U^{\leq k}$ q) iff there exists a path (s_0, s_1, …, s_j, …, s_i, …) from s_0 to s_i such that M,s_j ⊨ p until M,s_j ⊨ q in time constraints k.

proof. (Only if:) Suppose M,s ⊨ E(p $U^{\leq k}$ q). Then there is an infinite path (s_0, s_1, …, s_j, …, s_i, …) in M and a state s_i ∈ S for infinitely many distinct i, such that

(1) s_i is a successor of s_j ($0 \leq j < i$)

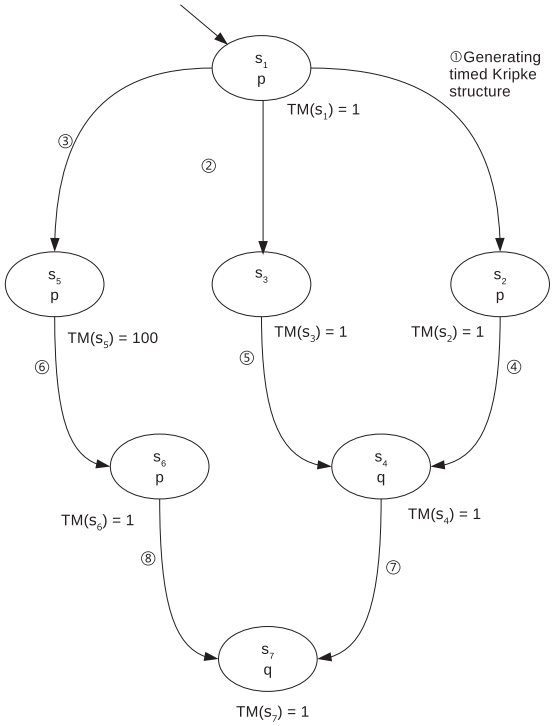

Fig. 7 Generating timed Kripke structure

(2) M,s_i ⊨ q

(3) M,s_j ⊨ p

(4) $\sum_{\alpha=0}^{i} TM(s_\alpha) \leq k$

If in the infinite path (s_0, s_1, …, s_j, …, s_i, …), time constrains k - $\sum_{\alpha=0}^{i} TM(s_\alpha) \geq 0$ according to the fixpoint.

(If:) if there is an infinite path (s_0, s_1, …, s_j, …, s_i, …) according to the algorithm. Since M,s_i ⊨ q and M,s_j ⊨ p, then backward to compute execution time such that time constraints k - $\sum_{\alpha=0}^{i} TM(s_\alpha) \geq 0$. So M,s ⊨ E(p $U^{\leq k}$ q).

5.3 Example

We will give an example in order to explain Algorithm 1 and Algorithm 2.

We specify verification property by RTCTL as follows.

E(p $U^{\leq 10}$ q)

We verify whether timed Kripke structure satisfies E(p $U^{\leq 10}$ q) according to Algorithm 1and Algorithm 2 as follows.

S = {s_1, s_2, s_3, s_4, s_5, s_6, s_7} S_0 = {s_1}

R = {(s_1, s_2), (s_1, s_3), (s_1, s_5), (s_2, s_4), (s_3, s_4), (s_4, s_7), (s_5, s_6), (s_6, s_7)}

L(s_1) = {p}	L(s_2) = {p}
L(s_3) = {}	L(s_4) = {q}
L(s_5) = {p}	L(s_6) = {p}
L(s_7) = {q}	TM(s_1) = 1
TM(s_2) = 1	TM(s_3) = 1
TM(s_4) = 1	TM(s_5) = 100
TM(s_6) = 1	TM(s_7) = 1

Generating timed Kripke structure and model checking

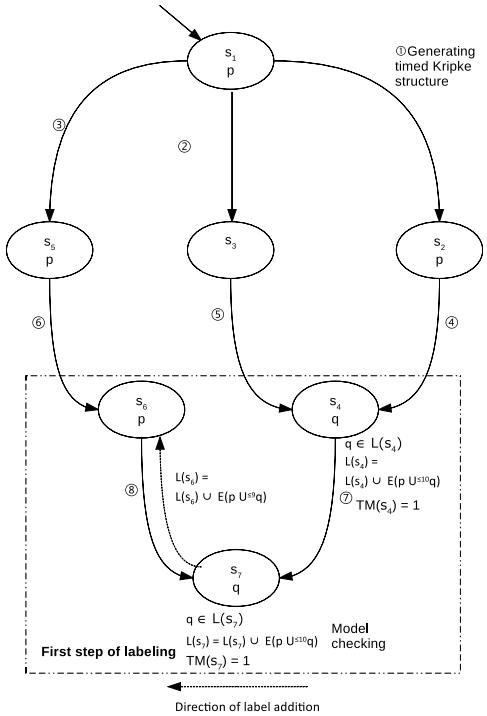

Fig. 8 Model checking (first step of labelling)

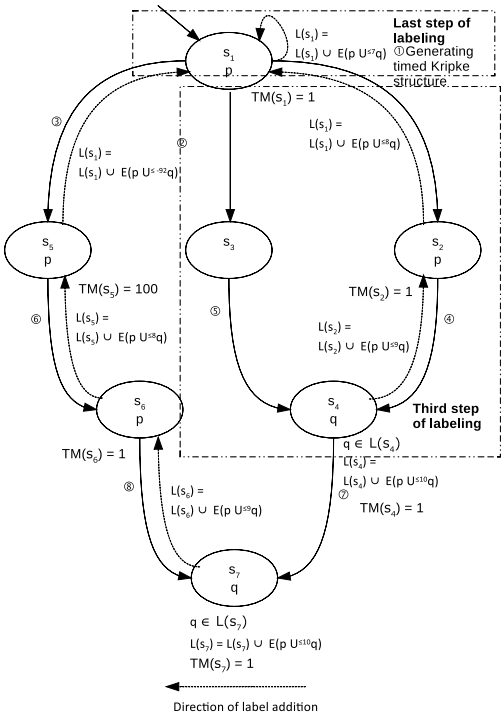

Fig. 10 Model checking (third and last step of labelling)

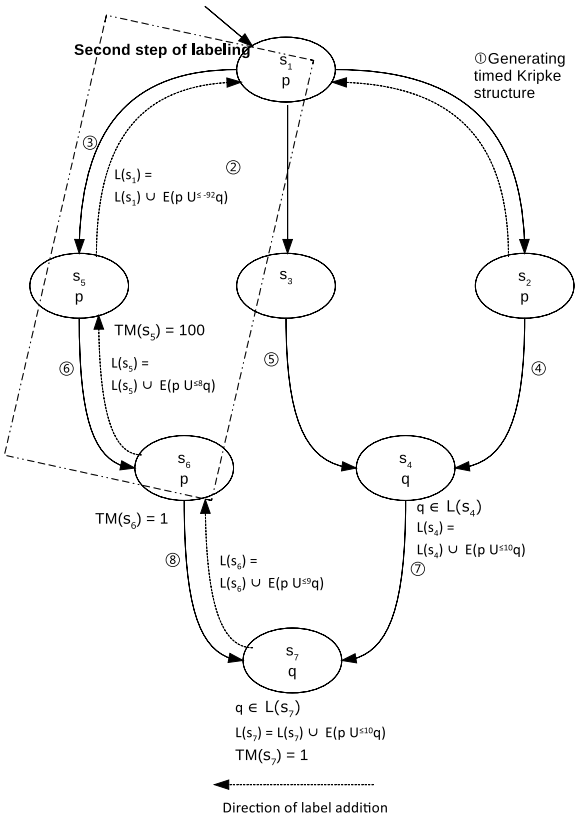

Fig. 9 Model checking (second step of labelling)

are shown in Fig. 7, Fig. 8, Fig. 9, Fig. 10. Timed Kripke structure is generated in the order of ①, ②, ③, ④, ⑤, ⑥, ⑦. Incidentally, this formula is obviously satisfied since there is a path $s_1 \rightarrow s_2 \rightarrow s_4$

To verify RTCTL formula, Algorithm 1 generated timed Kripke structure as Fig. 7. Figure 8, Fig. 9 and Fig. 10 give snapshots of Algorithm 2 in operation on the timed Kripke structure for the labeling function.

The first, add a label to the state that satisfies q as shown in Fig. 7. Next, add a label to the previous state of the state that satisfies q after compute the execution time. And verify it backward as shown in Fig. 8 and Fig. 9 and add a label to that state after compute the execution time. Compute the initial state of one some path and label it. Then it has been labeled and satisfied the time constraints k. So, the result becomes true and the verification result becomes true.

6. Experiments

We used five programs written from H8/3687 microcontroller [14], [15].

The execution time is obtained from the execution state number of the assembly instruction. Since the operating frequency of H8/3687 microcontroller is 20 MHz, the execution time per state is as follows.

$$1 / 20\text{Mhz} = 0.05\mu\text{s}$$

Therefore, we assume the unit time of time constraint k is 0.05μs.

The experiment was conducted on a laptop as follows:

- CPU: Intel(R) Core(TM) i7-7700HQ processors running at 2.80GHz
- Memory: 16GB main memory

Table 1 The results of verifying Program 1

$Total_{led}$	states	relations	verification time (s)	execution time (ms)	result
10	1005	890	19.858	0.1571	true
50	2205	1770	49.381	0.2016	true
100	3705	2870	85.066	0.5576	true
199	6675	5048	162.548	0.9981	true
200	6705	5070	162.592	1.0025	false
300	9705	7270	247.373	1.4476	false

- OS: windows 10
- Simulator is written in a combination of Java and Scala, and Model Checker is written in Scala as follows.
- Java 1.8.0_121, 15000 lines
- Scala 2.11.8, 6000 lines
- Tools: JFlex [16], Jacc [17]

6.1 Program 1

LED program lights up one LED by a sensor of the microcontroller outputs. The sensor can identify black and white. When the sensor of the microcontroller detects white, it outputs 1 and LED lights up.

When it detects black, it outputs 0 and LED turn off. Furthermore, we preset the sensor always detected white and it turns off immediately after the LED lights up. For example, $Total_{led}$ times LED lights up by $Total_{led}$ times the sensor outputs.

We specify timing constraints by RTCTL for this experiment as follows:

$$EF^{\leq k}(Total_{led} = n) = E(trueU^{\leq k}(Total_{led} = n)) \quad (RTCTL\ 1)$$

$Total_{led}$ denotes the total number of lights up times. The n is the number of lights up times, indicated by the value of the register R0 in state. The formula intuitively means that LED lights up $Total_{led}$ times happen at some state on some path from initial states within the timing constraint k. Besides we preset the specific LED light up number while timing constraint k is limited within one millisecond (k = 20000).

The experimental results are shown in Table 1. The first column presents the total number of lights up times. The second column gives the number of states. The third column indicates the number of relations. The fourth column presents the total time of simulator and model checking. The fifth column presents execution time of assembly program for some path satisfies $Total_{led}$ times lights up LED or the execution time of assembly program for a certain path is only not satisfied time constraint. The last column shows the result which shows satisfiability of RTCTL formula. Execution time is obtained from the total execution time of executed states. The execution time of each state is an execution time of assembly instruction executed in each state.

The experimental results showed the effectiveness of the proposed approach. We implement model checker in order to verify whether timed Kripke structure satisfies RTCTL formulas. These resulting data show that it can be verified whether or not lights up $Total_{led}$ times within the time constraint k. Moreover, as a result, we showed that the maximum number of LED lights up within the time constraint k. As shown in Table 1, if $Total_{led}$ increases, the number of states polynomially increases (it is shown in Fig. 11), and RTCTL formula becomes false.

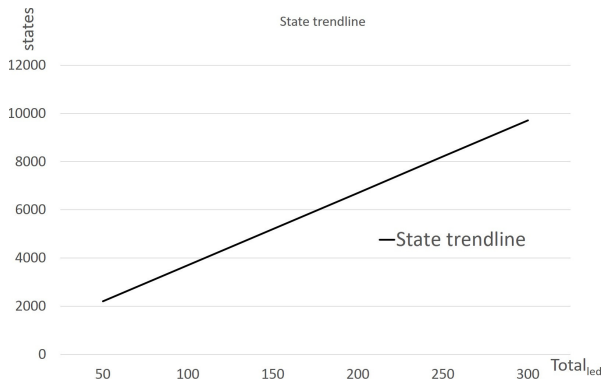

Fig. 11 The results of states in constrast $Total_{led}$

6.2 Program 2

Program 2 (target_motor) controls the two motors on the left and right of the robot by acquires the values of three sensors. The sensor is the same as the sensor in Program 1. After timer initialization, when timer overflow interrupt of timer b1 occurs, acquire sensor value and set new target current value. When a timer overflow interrupt occurs in the timer v, PID control is performed from the current target value and the measured current value, and the value is output to the motor. The robot controls the motor speed as shown in Table 2 by three sensor values. Obviously, when the robot is in the straight state (such that M_3), it can be done by speed control such as going straight ahead. In the same way, the robot turns to the left from the situation inclination right(M_1). The robot only increases motorL speed from 30% to 60%. The robot will be the situation slightly inclination.

In order to verify the real-time property of the DC (direct current) motor speed control, we prepared the following programs for verification. By the speed control of the DC motor, the robot turns to the left from the situation of inclination right(M_1), further the robot turns to the left a little from the situation of slight inclination right(M_2), then the robot goes straight ahead(M_3), subsequently the robot turns to the right a little from the situation of slight inclination left(M_4), then the robot turns to the right from the situation of inclination left(M_5), the final the robot goes straight ahead(M_6). Simultaneously, six RTCTL formulas were prepared and experimented as Table 3.

Incidentally, M_1 - M_6 in RTCTL 2 - 7 shows the target value of the rotational speed of the motor as shown in Table 2. The speed denotes the rotational speed of the motor, indicated by the value of the register ER1. Furthermore, in RTCTL 2, the rotational speed of the motor satisfies the tar-

Table 2 The situation of speed control

motor (state)	sensor	motorL	motorR	robot's situation
M_1	100	30%	90%	inclination right
M_2	110	60%	90%	slightly inclination right
M_3	010	100%	100%	straight
M_4	001	90%	30%	slightly inclination left
M_5	011	90%	60%	inclination left
M_6	010	100%	100%	straight

Table 3 RTCTL formulas of Program 2

RTCTL	formula
RTCTL 2	$EF^{\leq k_1}(M_1 = speed) = E(\text{true } U^{\leq k_1}(M_1 = speed)$
RTCTL 3	RTCTL 2 $\wedge$ E(true $U^{\leq k_2}((M_2 = speed)$
RTCTL 4	RTCTL 3 $\wedge$ E(true $U^{\leq k_3}((M_3 = speed)$
RTCTL 5	RTCTL 4 $\wedge$ E(true $U^{\leq k_4}((M_4 = speed)$
RTCTL 6	RTCTL 5 $\wedge$ E(true $U^{\leq k_5}((M_5 = speed)$
RTCTL 7	RTCTL 6 $\wedge$ E(true $U^{\leq k_6}((M_6 = speed)$

get value within the time constraint k_1 in some path. RTCTL 3 satisfies the target value for the rotation speed of the motor within the time constraint k_1 within some path and the rotation speed of the motor satisfies the second target value within the time constraint k_2 in some path. RTCTL 4 - 7 is similar and will be omitted in here.

In this experiment, when an interrupt is occurred by timer b1 every 1024μs, timer v occurs an interrupt every 400μs. Set the target current value for speed control of the motor when an interrupt occurs in timer b1. Thereafter, when timer v occurs an interrupt, PID control is performed from the target value and the measured current value, and the value is output to the motor. Therefore, set the time constraint value of this time as follows.

$k_1 = k_2 = k_3 = k_4 = k_5 = k_6 = 30000$ (1500μs)

The experimental results are shown in Table 4. The first column presents the RTCTL formula. The second column gives the number of states. The third column indicates the number of relations. The fourth column presents the total time of simulator and model checking. The fifth column presents execution time of assembly program for some path satisfies RTCTL formula or the execution time of assembly program for a certain path is only not satisfied time constraint. The last column shows the result which shows satisfiability of RTCTL formula. In the case of false, RTCTL 3 was created by adding a delay code when generating the second time timer b1 interrupt in Program 2. According to this experiment, RTCTL 2 - 7 of the verification properties increases the total number of states as the number of motor speed control increases. Since the initialization of the microcontroller the number of relations is less than the number of states. Finally, from the example that becomes false for RTCTL 3, it was possible to verify whether the rotation speed of the motor satisfies the target value within the time constraint. This experiment succeeded in verification of the real-time property of DC motor speed control.

Table 4 The results of verifying Program 2

RTCTL	states	relations	verification time (s)	execution time (ms)	result (ms)
RTCTL 2	7249	7077	238.2	1.3062	true
RTCTL 3	13402	13172	625.7	2.6186	true
RTCTL 4	17570	17283	866.7	3.5312	true
RTCTL 5	23725	23380	1245.3	4.8438	true
RTCTL 6	29878	29475	1838.0	6.1565	true
RTCTL 7	34047	33587	2274.2	7.0687	true
RTCTL 3	16146	15852	754.0	3.0551	false

6.3 Program 3

Program (target_timer) acquires the values of sensors and controls the motors to operate the robot. Lights or extinguishes LED same as Program 1. The microcontroller's initialization simultaneously obtains the sensor's value. When timer overflow interrupt of timer_b1 occurs, from the acquired sensor value to set new target current value and to lights up the LED. On the other hand, when a timer overflow interrupt occurs in the timer_v, PID control is performed from the current target value and the measured current value, and the value is output to the motor.

In this experiment, we verify the real-time property of timer_b1 when interrupting occurred. We specify timing constraints by RTCTL for this experiment as follows:

$$EF^{\leq k}(timer\_b1 = true) = E(trueU^{\leq k}(timer\_b1 = true) \quad (RTCTL\ 8)$$

In this experiment, since timer_b1 occurs interrupt only once, acquisition of the sensor value is performed together with initialization. When timer_b1 occurs an interrupt every fixed time, the timer_v occurs an interrupt every 400μs. In RTCTL 8, an interrupt is occurred within the time constraint k of timer_b1 in a certain path. When timer_b1 counts, timer_v may occur interrupt more than once, so time constraints k is not satisfied with this experiment.

k = Interrupt occur time of timer_b1

The experimental results are shown in Table 5. The first column presents the timer_b1 interrupt time. The second column gives the number of states. The third column indicates the number of relations. The fourth column presents the total time of simulator and model checking. The fifth column presents execution time of assembly program for some path satisfies timer_b1 interrupt or the execution time of assembly program for all path are not satisfy RTCTL formula. The six column shows the time constraint k. The last column shows the result which shows satisfiability of RTCTL formula.

Due to the influence of the timer_v, execution time exceeds the set interrupt time of the timer_b1. All verification results of this experiment become false, but verification of real-time property was able to verify conversely. Although the set interrupts time of the timer_b1 increases from 500μs to 700μs and the execution increases according to it, but the total number of states does not increase is due to the timer_v

Table 5 The results of verifying Program 3

time_b1(k) (ms)	states	relations	verification time (s)	execution (ms)	result
0.5024	5615	5374	126.5	0.6138	false
0.6016	5615	5374	138.4	0.7130	false
0.7008	5615	5374	139.4	0.8122	false
0.8064	7938	7650	218.1	1.0295	false
0.9088	7938	7650	226.8	1.1319	false
1.0112	7938	7650	241.4	1.2343	false

interrupt has not occurred second during that time. Since interrupt time of timer_v is shorter than timer_b1, an interrupt has occurred first and when executing an interrupt instruction of timer_v, timer_b1 is not counting because multiple interrupts are disabled. Therefore, actual the interrupt occurrence time of the timer b1 is more than the setting. In this experiment, it is not possible to accurately verify the interrupt time of the timer_b1 due to the influence of the timer_v, and for verification accurately, two verification target programs (PROGRAM 4 and PROGRAM 5) were prepared.

6.4 PROGRAM 4

Program (target_timer_b1) acquires the values of sensors and controls the motors to operate the robot same as Program 2. Lights or extinguishes LED same as Program 1. The microcontroller's initialization simultaneously obtains the sensor's value. When timer overflow interrupt of timer_b1 occurs, from the acquired sensor value to set new target current value and to lights up the LED. After that PID control is performed from the current target value and the measured current value, and the value is output to the motor.

In this experiment, we verify the real-time property of timer_b1 when interrupting occurred like PROGRAM 3 and we eliminate the influence of the timer_v. We specify timing constraints by RTCTL for this experiment as same as RTCTL 8.

This experiment only deals with timer_b1. Eliminate the influence of timer_v and verify more accurately whether timer_b1 occurs an interrupt within the time constraint. The experimental results are shown in Table 6. The columns are similaryl with Table 5. The number of states increases when increasing the interrupt time set by timer_b1. The execution time is 502.35 μs for the set time 502.4 μs, so the verification result is true. As shown in Table 6, when the execution time is less than the time constraint k, the verification result becomes true. And for false examples change PROGRAM 4 (target_timer_b1) as follows:

(1) When initializing timer_b1, set the initial count value TB1.TCB1 = 99 which becomes 502.4 μs to TB1. TCB1 = 89.

(2) At the time of initialization, set interrupt enable to false, and main will enter an infinite loop, interrupt will not occur. Then, a state explosion occurs and it can not be verified.

In order to exclude the influence of timer_v, model checking was carried out with the program handling only timer_b1 as the verification target in this experiment. Since

Table 6 The results of verifying Program 4

timer_b1 (ms)	states	relations	verification time (s)	execution time (ms)	result (ms)
0.5024	4030	3837	93.4	0.50235	true
0.6016	4526	4333	109.9	0.60155	true
0.7008	5022	4829	128.0	0.70075	true
0.8064	5550	5357	156.3	0.0.80635	true
0.9088	6062	5869	177.1	0.90875	true
1.0112	6574	6381	186.9	1.01115	true
0.5024	4190	3997	100.1	0.53435	false
0.5024	-	-	-	-	false

Table 7 The results of verifying Program 5

timer_v (ms)	states	relations	verification time (s)	execution time (ms)	result (ms)
0.4032	3482	3305	80.2	0.40285	true
0.5056	3994	3817	97.4	0.50205	true
0.6016	4474	4297	99.6	0.60125	true
0.7040	4986	4809	119.6	0.70365	true
0.8000	5466	5289	134.6	0.79965	true
0.9024	5978	5801	158.5	0.90205	true
1.0048	6490	6313	172.7	1.00445	true
1.0048	6586	6409	175.6	1.02365	false
1.0048	-	-	-	-	false

the influence of timer_v was eliminated, it was possible to verify more accurately whether or not the interrupt time of timer_b1 satisfies the time constraint. Then the real-time property of timer_b1 was verified.

6.5 PROGRAM 5

Program (target_timer_v) acquires the values of sensors and controls the motors to operate the robot same as Program 2. Lights or extinguishes LED same as Program 1. The microcontroller's initialization simultaneously obtains the sensor's value. When timer overflow interrupt of timer_v occurs, from the acquired sensor value to set new target current value and to lights up the LED. After that PID control is performed from the current target value and the measured current value, and the value is output to the motor.

In this experiment, we verify the real-time property of timer_v when interrupting occurred like PROGRAM 4 and we eliminate the influence of the timer_b1. We specify timing constraints by RTCTL for this experiment as follows:

$$EF^{\leq k}(timer\_v = true) = E(trueU^{\leq k}(timer\_v = true) \quad (RTCTL\ 9)$$

This experiment only deals with timer_v. Eliminate the influence of timer_b1 and verify more accurately whether timer_v occurs an interrupt within the time constraint. The experimental results are shown in Table 7. The columns are similarly with Table 5. The number of states increases when increasing the interrupt time set by timer_v. The execution time is 402.85 μs for the set time 403.2 μs, so the verification result is true. As shown in Table 7, when the execution time is less than the time constraint k, the verification result becomes true. And for false examples change PROGRAM 5 (target_timer_v) as follows:

(1) When initializing timer_v, set the initial count value TV.TCORA = 157 which becomes 1004.45 μs to TV.TCORA = 160. On the other hand, the execution time is 1023.65 μs, and the verification result is false.

(2) At the time of initialization, set interrupt enable to false, and main will enter an infinite loop, interrupt will not occur. Then, because a state explosion occurs, the simulator will not stop and it can not be verified.

In this experiment, in contrast to the experiment of PROGRAM 4 and in order to eliminate the influence of timer_b1, only timer_v handles interrupt processing. Since the influence of timer_b1 was eliminated, it was possible to verify more accurately whether or not the interrupt time of timer_v satisfies the time constraint. Based on the above verification results of PROGRAM 4 and PROGRAM 5, the verification system proposed can verify the real-time properties of the assembly program.

6.6 Summary

From the above experimental results, it can be seen that real-time model checking can be performed by handling the execution time of the target assembly program. In the case of model checking after generating timed Kripke structure, when the timed Kripke structure satisfies CTL formula (like E(p U q)) and does not satisfy the time constraint k (of E(p $U^{\leq k}$ q)), we can figure out the execution time of the path by computation. For the entered validation property (RTCTL formula), it is not only performed the model check but also possible to verify the real-time property. The above experiment showed the effectiveness and usefulness of RTCTL model checking. If the simulator does not stop or the state explodes, it was confirmed that model checking may not be executed in some cases. As all the target programs are finite state systems, we can verify them, But program is an infinite system, there may be two situations as follows. (1) The verification result becomes true while generating timed Kripke structure. (2) Model checking continues indefinitely. It can not be verified in the infinite state.

7. Conclusion

In this paper, we have developed a verification system. RTCTL model checker verifies whether timed Kripke structure satisfies real-time properties. As future work, we will develop model checking of embedded systems using counterexample-guided abstraction refinement (CEGAR).

Beside this, we want to detect other properties in embedded systems. When the real-time model checking result is false, it also makes it possible to output the reason at the same time. Then, make it clear whether the real-time property is not satisfied or the safety is not satisfied. In this verification system, the model checking is performed after generating the timed Kripke structure of simulator, but there is research of CTL model checking while generating the timed Kripke structure of simulator. In the future, we will conduct comparative experiments.

References

[1] R. Jhala and R. Majumdar, "Software model checking," ACM Computing Surveys (CSUR), vol.41, no.4, 2009.
[2] L. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," LNCS, vol.4963, pp.337–340, 2008.
[3] S. Yamane, R. Konoshita, and T. Kato, "Model checking of embedded assembly program based on simulation," IEICE Trans. Inf. & Syst., vol.E100-D, no.8, pp.1819–1826, 2017.
[4] B. Schlich, "Model Checking of Software for Microcontrollers," ACM Transactions on Embedded Computing Systems, vol.9, no.4, 2010.
[5] J. Kobashi, S. Yamane, and A. Takeshita, "Development of SMT-Based Bounded Model Checker for embedded assembly program," GCCE 2014, pp.696–698, 2014.
[6] M. Kuo, R. Sinha, and P. Roop, "Efficient WCRT analysis of synchronous programs using reachability," 48th ACM/EDAC/IEEE Design Automation Conference (DAC), pp.480–485, 2011.
[7] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan, "Quantitative Temporal Reasoning," Real-Time Systems, vol.4, no.4, pp.331–352, 1992.
[8] R. Alur and D.L. Dill, "A theory of timed automata," TCS, vol.126, no.2, pp.183–235, 1994.
[9] P. Bouyer, U. Fahrenberg, K.G. Larsen, N. Markey, J. Ouaknine, and J. Worrell, "Model checking real-time systems. In Handbook of Model Checking," pp.1001–1046, Springer, Cham, 2018.
[10] M.H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, and B. Lisper, "Learning-based Response Time Analysis in Real-Time Embedded Systems: A Simulation-based Approach," ACM/IEEE 1st International Workshop on Software Qualities and their Dependencies, pp.21–24, 2018.
[11] J.-L. Béchennec and F. Cassez, "Computation of wcet using program slicing and real-time model-checking." arXiv preprint arXiv:1105.1633, 2011.
[12] S.A. Kripke, "Semantical Considerations on Modal Logic," Acta Philosophica Fennica, 16, pp.83–94, 1963.
[13] E.M. Clarke and E.A. Emerson, Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic, LNCS 131, pp.52–71, 1981.
[14] R.E. Corporation, Renesas Electronics, Renesas Electronics Corporation (online), http://japan.renesas.com/
[15] nuvo WHEEL: ZMP, http://www.zmp.co.jp/products/wheel
[16] G. Klein, JFlex - The Fast Scanner Generator for Java, CSE UNSW (online), available from (http://jflex.de/)
[17] M.P. Jones, Jacc: just another compiler compiler for Java, Department of Computer Science and Engineering at the OGI School of Science & Engineering at OHSU (online), available from (http://jflex.de/)

Yajun Wu received B.S. degrees from Southwest Minzu Univeristy. M.S degrees from Kanazawa Univeristy. Now he is a doctor course second grade student of Kanazawa University. He is interested in formal verification of real-time system and deep learning.

Satoshi Yamane received B.S., M.S and Ph.D. degrees from Kyoto Univeristy. Now he is a professor of Kanazawa University. He is interested in formal verification of real-time and distributed computing.