Modelling and Verification of ARINC 653 Hierarchical Preemptive Scheduling

Ning Fu, Lijun Shan, Chenglie Du, Zhiqiang Liu, and Han Peng
School of Computer Science, Northwestern Polytechnical University, China

Abstract: *Avionics Application Standard Software Interface (ARINC 653) is a software specification for space and time partitioning in safety-critical avionics real-time operating systems. Correctly designed task schedulers are crucial for ARINC 653 running systems. This paper proposes a model-checking-based method for analyzing and verifying ARINC 653 scheduling model. Based on priced timed automata theory, an ARINC 653 scheduling system was modelled as a priced timed automata network. The schedulability of the system was described as a set of temporal logic expressions, and was analyzed and verified by a model checker. Our research shows that it is feasible to use model checking to analyze task schedulability in an ARINC 653 hierarchical scheduling system. The method discussed modelled preemptive scheduling by using the stop/watch features of priced timed automata. Unlike traditional scheduling analysis techniques, the proposed approach uses an exhaustive method to automate analysis of the schedulability of a system, resulting in a more precise analysis.*

Keywords: *ARINC653, schedulability analysis, model checking, UPPAAL.*

Received June 15, 2016; accepted March 19, 2019
https://doi.org/10.34028/iajit/17/1/12

1. Introduction

The Avionics Application Standard Software Interface (ARINC653) is a software specification for space and time partitioning in safety-critical avionics real-time operating systems [8]. In an integrated avionics electronic system, correctness of calculation depends not only on the logical result but also on the result arrival time. Schedulability of a real-time system is critical for the system to respond to its applications. Therefore, it is essential to study the schedulability analysis problem for partition management models (ARINC 653 standards).

In this paper, we study the schedulability analysis of ARINC 653 hierarchical schedulers based on model checking. The partitions, hierarchical schedulers, and tasks in ARINC 653 were modelled as a network of priced timed automata. Schedulability was described as a set of temporal logic formulas. We analyzed and verified the schedulability of hierarchical schedulers using the real-time model checker UPPAAL. By taking into account more detailed specifics of individual tasks, this allowed a safe but far less pessimistic schedulability analysis. Moreover, the model based approach provided a self-contained visual representation of the system with formal, non-ambiguous interpretation. This make it possible for simulation, verification and validation.

2. Related Work

Classic schedulability analysis methods calculate the schedulability of a system by comparing the Central Processing Unit (CPU) utilization with a specific boundary value [7, 10, 15]. The disadvantage of those methods is that boundary conditions satisfaction is only a sufficient condition to determine the schedulability of a system. Many high CPU utilization schedulable tasks sets often cannot pass the verification and would be considered non-schedulable. Therefore, the results of such analysis methods are often too pessimistic.

In recent years, the model based method of schedulability analysis and verification has garnered much attention [4, 9, 11]. Paper [11] used time automata to model the hierarchical scheduling process, used the model checker Times to analyze and verify a scheduling model in VxWorks. Since pure timed automata itself does not support clock stopwatch, on the modelling of preemption, the paper assumed that task could be preempted only at the time points of the integral multiple of the tiny time slice. The approach would inevitably lead to the terrible growth of the model state space.

Paper [12] used the UPPAAL to model and verify Architecture Analysis and Design Language (AADL) thread component schedulability under a non-preemptive dispatch strategy. It studied the schedulability problem at only the thread scheduling level. Paper [19] used TA theory to model and check real-time multiple task scheduling system that was in accordance with the Open Systems and the corresponding interfaces for automobile Electronic/Vehicle Distributed eXecutive (OSEK/VDX) standard. Paper [5] focus on finding the Worst-Case Execution Time (WCET) of parallel embedded software by generating the test-data using a meta-heuristic optimizing search algorithms. The search-based optimization used yielded the input vectors of the parallel embedded software that cause maximal execution times.

Paper [14, 16] modelled recurrent real-time applications as a set of parallel Directed Acyclic Graph (DAG) tasks and analyzed the schedulability by exploring the space of all possible schedules using the notion of a schedule-abstraction graph. Yalcinkaya *et al.* [20] presents a model checking based exact schedulability test for sets of periodic and sporadic self-suspending tasks with fixed preemption. These work did not take into consideration the characteristics of the ARINC 653 partition scheduling system, therefore cannot be used directly for analyzing the ARINC 653 scheduling model.

Paper [17] explored how to analyze the end-to-end timing characteristics of an AADL model using real-time calculus. Times is another kind of scheduling system analysis tool based on task automata [3]. It can analyze the schedulability of a system precisely. But the Times definition of a periodic task is not consistent with that of an ARINC 653 periodic task; moreover, the tool does not support preemptive scheduling system modelling.

Unlike related work, our work focused on the analysis and verification of the ARINC 653 partition scheduling system. We use priced TA to model the hierarchical scheduling process, and analyzed the schedulability by model checking. The method discussed in this paper solved the modelling of preemptive scheduling by introducing priced TA. The formal method we propose uses an exhaustive method to automate the analysis of the schedulability of a system, therefore we obtained a more precise analysis result. The details of a scheduling process could be studied by observing the scheduling traces of a system given by the simulator. For cases where task sets are not schedulable, the checker provides a counterexample. Though the method proposed in this paper is aimed at ARINC 653 standard, the model could also be used for analysis of other scheduling systems after modification.

3. Properties of The ARINC 653 Avionics System

The ARINC 653 avionics partition operating system has a hierarchal scheduling model: partition level scheduling and task level scheduling. Partition level scheduling refers to the allocation of fixed time slices to different partitions. Task level scheduling refers to the process of permitting tasks to use CPU resources.

The main features of partition level scheduling are:

1. A partition is the scheduling unit of the scheduling system, which contains a set of partitions.
2. Partitions are independent and have no priority preemptive relationship.
3. Partitions are cyclically activated. When a system is initialized, a minimum period time framework is designed and configured statically by the system administrator, where time slices are allocated and dispatched in sequence to the partition set. A minimum period time framework is shown in Figure 1.

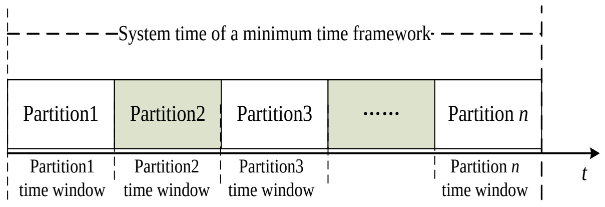


Figure 1. Minimum time cycle framework of ARINC 653.

The main features of task level scheduling are:

1. Tasks to be scheduled are mounted on different partitions. Each partition has its own task set to be scheduled. Scheduling policies belonging to different partitions are independent
2. A partition's task set is scheduled according to a specific policy within the time slices assigned to the partition. During a task set execution, if a time slice is exhausted the execution is suspended and is resumed at the arrival of the next time slice belonging to the partition.
3. All tasks are periodic, and the first time arrival of a task is in a partition's time window that first gets its time slice. The ARINC 653 avionics partition scheduling model discussed in this paper is shown in Figure 2.

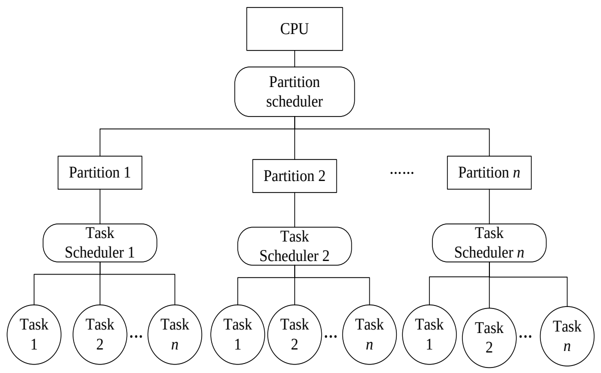


Figure 2. Partition scheduling model of ARINC 653.

4. Priced Timed Automata

The theoretical basis of our method is Networks of Priced Timed Automata (NPTA), based generally on regular timed automata [2] in that clocks may have different rates in different locations. In fact, the expressive power (up to timed bisimilarity) of NPTA equals that of general Linear Hybrid Automata (LHA) [1]. Priced TA supports a clock stopwatch, and thus provides an effective mechanism for describing preemptive scheduling.

Let X be a finite set of variables, called *clocks*. A *clock valuation* over X is a mapping v: $X \rightarrow IR_{\geq 0}$, where $IR_{\geq 0}$ is the set of nonnegative reals. We write $IR^X_{\geq 0}$ for the set of clock valuations over X. Let r: $X \rightarrow IN$ be a *rate vector*, assigning a rate to each clock of X. Then, for $v \in IR^X_{\geq 0}$ and $d \in IR_{\geq 0}$ a delay, we write $v+r{\cdot}d$ for the clock valuation defined by $(v+r{\cdot}d)(x)=v(x)+r(x){\cdot}d$ for any clock $x \in X$. We denote by IN^X the set of all rate vectors. If $Y \subseteq X$, the valuation $v[Y]$ is the valuation assigning 0 when $x \in Y$ and v(x) when $x \notin Y$. An *upper bounded* (*lower bound*) *guard* over X is a finite conjunction of simple clock bounds of the form $x{\sim}n$ where $x{\sim}n$ where $x \in X$, $n \in IN$, and $\sim \in \{<, \leq\}$ ($\sim \in \{>, \geq\}$). We denote by $U(X)$ ($L(X)$) the set of upper (or lower) bound guards over X, and write $v|{=}g$ whenever v is a clock valuation satisfying the guard g. Let $\Sigma=\Sigma_i \uplus \Sigma_o$ be a disjoint set of input and output actions.

- *Definition* 1. A Priced Timed Automaton (PTA) is a tuple A= 〈L, l_0, X, Σ, E, R, I〉 where:

  1. L is a finite set of locations.
  2. $l_0 \in$ L is the initial location.
  3. X is a finite set of clocks.
  4. $\Sigma=\Sigma_i \uplus \Sigma_o$ is a finite set of actions partitioned into inputs (Σ_i) and outputs (Σ_o), (v) $E \subseteq L \times \mathcal{L}(X) \times \Sigma \times 2^X \times L$ is a finite set of edges.
  5. R: $L \rightarrow IN^X$ assigns a rate vector to each location.
  6. I:$L \rightarrow U(X)$ assigns an invariant to each location.

The semantics of NPTAs is a timed labeled transition system whose states are pairs $(l,v) \in L \times IR^X_{\geq 0}$ with $v|{=}I(l)$, and whose transitions are one of the following two types:

1) In a *time* or *delay transition* some $d \in Time$ elapses, but the location is left unchanged. Formally, $\langle l,v \rangle \xrightarrow{d} \langle l,v' \rangle$, with $d \in IR_{\geq 0}$ and $v'=v+R(l){\cdot}d$ iff $(v' = v+R(l){\cdot}d') \models I(l)$ for all $d' \in [0,d]$.
2) In an *action* or *discrete transition* an action $\alpha(\alpha \in \sum)$ occurs and some clocks may be reset, but time does not advance. Formally, $\langle l,v \rangle \xrightarrow{\alpha} \langle l',v' \rangle$ iff there exist an edge $(l, g, \alpha, Y, l') \in E$ with $v|{=}g$ and $v'=v[Y{:=}0]$ and $v'|{=}I(l')$

- *Definition* 2. *Let* $\mathcal{A}^j$= 〈L^j, l_0^j, X^j, Σ, E^j, R^j, I^j〉 (*with* j=1,2,…,n) *be composable NPTAs. Their composition* ($\mathcal{A}_1 | \mathcal{A}_2 | ... | \mathcal{A}_n$) *is the NPTA* $\mathcal{A}$= 〈L, l_0, X, Σ, E, R, I〉, where (*i*) $L=\times_j L^j$, (*ii*) l_0= 〈l_0^1; $l_0^2,\ldots,l_0^n$〉; (*iii*) $X=\cup_j X^j$, *where* $X^j \cap X^k = \emptyset$, *when* $j \neq k$; (*iv*) $R(l)(x)=R^j(l^j)(x)$ *when* $x \in X^j$, (*v*) $I(l)=\cap_j I(l^j)$; and (*vi*) $(l, \cap_j g_j, a, \cup_j r_j, l') \in E$, whenever $(l_j, g_j, a, r_j, l_j') \in E^j$ for j=1,2,…,n.

Whenever A^j= 〈L^j, X^j, Σ^j, E^j, R^j, I^j〉 (j=1,2,…,n) are NPTAs, they are *composable* into a *closed network* iff their clock sets are disjoint ($X^j \cap X^k = \emptyset$, when $j \neq k$), and they have the same action set ($\Sigma=\Sigma^j=\Sigma^k$ for all j,k). Automata in an NPTA may perform their actions independently, or may perform synchronous communications by simultaneous input/output actions through shared channels. Synchronous communications actually correspond to the interactions between entities.

5. A Hierarchical Scheduling Model with PTA

5.1. Periodic Task

The dispatch policy of a periodic task is that when a new period is coming, the task is dispatched and begins waiting for its start. By default, the deadline of a task is the same as its period. The time interval between adjacent task dispatches cannot be less than the task's period. That is, when a new dispatch of a task is coming, the last dispatch of the task must have executed completely. Otherwise, a timeout error occurs.

As shown in Figure 3, six locations are defined in the priced TA. To keep track of the task execution time and the dispatch time, 2 clocks are defined: an execution clock *e* and a deadline clock *t*.

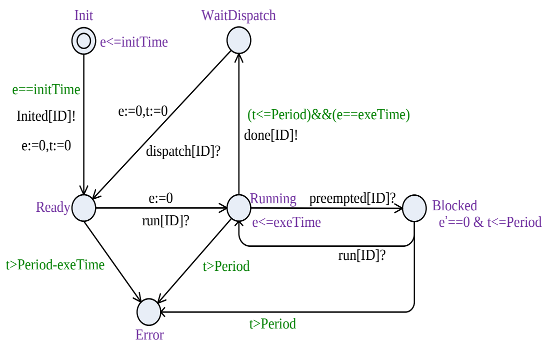


Figure 3. Template for a periodic task.

When initialization completes, Task sends an inited event to the task trigger, moves to the Ready location, and then sets execution clock e and deadline clock t to zero. At the Ready location, if Task receives the permit execution event run, Task sets execution clock e to zero, and moves to location running. If the value of the current deadline clock t is greater than the difference between the deadline and execution times (Period-exeTime), the task execution must not be completed before the deadline, a timeout error occurs and Task moves to the Error location.

At the Running location, if a preempted event is received; i.e., during the course of execution a scheduling request of a higher priority task emerges, the current task is preempted and Task moves to the Blocked location. At the Blocked location, execution clock e is suspended and timekeeping is paused. If a permit execution event run is received, Task moves to location running and clock e resumes. If deadline clock t is greater than deadline Period, a timeout error occurs and Task moves to the Error location.

5.2. Task Trigger and Partition Scheduler

The role of the task trigger is to generate periodic scheduling requests. As shown in Figure 4, the model has Task triggered and dispatched by period cycles. There are 2 temporary locations defined in the automata. The time delays of the temporary locations are 0.

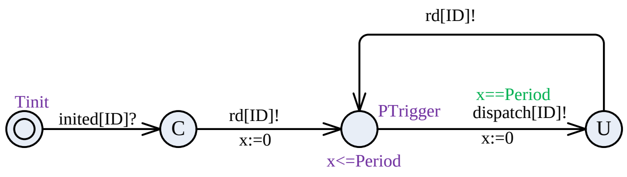


Figure 4. Automaton modelling the periodic task trigger.

The partition scheduler (Figure 5) studied in this paper contains 2 partitions, but in the proposed method a scheduler with *n* partitions could be analyzed similarly. When the time slice is exhausted, the partition scheduler moves to the Temp location, and then next partition is scheduled.

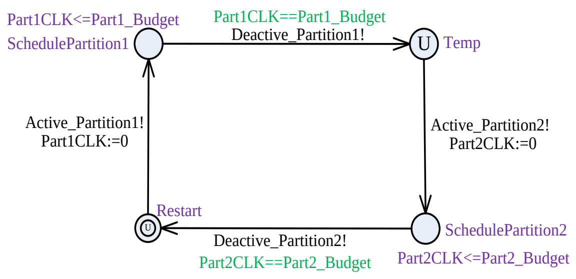


Figure 5. Partition scheduler model.

5.3. Task Scheduler

The task scheduler is responsible for second level scheduling. We take the task scheduler of partition 1 as an example (Figure 6). The scheduling policy of is priority-based preemptive scheduling. The array Partition1_Queue is introduced as the task queue to be scheduled.

The initial location of the task scheduler is pended. When a task dispatch event is received-*rd*[*e*] (*e* is a task ID)-the task scheduler adds the task ID into the task queue. In location *Idle*, if there are tasks in the task waiting queue to be scheduled, the task scheduler gets the highest priority task ID from the queue, sends an execution event to the task, and then moves to location *Occ*. If a partition suspend event is received (*Deactive_Partition1*), the task scheduler moves to location *Pended*.

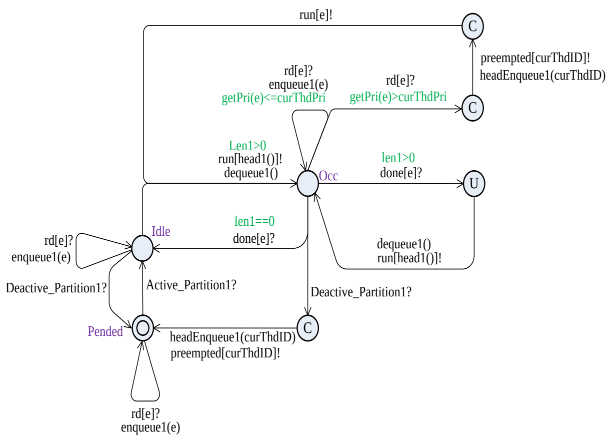


Figure 6. Automaton modelling the partition 1 thread scheduler.

From location *Occ*, if a scheduling request event *rd*[*e*] is received and the priority of the request task is higher than that of the currently running task, the task scheduler sends a hung up event (*preempted*[*curThdID*]) to the running task, adds the running task ID to the waiting list, sends a run event to the preemptive task, and moves back to location *Occ*. If a task execution completion event is received (*done*[*e*]), and there are tasks in the waiting list, the task scheduler sends a run event to the head task of the task queue, deletes the head of the waiting list, and moves back to *Occ*. If a task execution completion event is received (*done*[*e*]), and the waiting list is empty (*len*1==*0*), the task scheduler moves to the *Idle* location. If a partition suspend event *Deactive_Partition*1 is received, the task scheduler sends a hung up event to the task currently using the CPU, adds the task ID to the waiting list, and moves to location Pended.

The scheduling policy of partition 2 is round robin, and the implementation mechanism of task scheduling is similar to that of partition 1. Each time the ID of the task to be scheduled is obtained from the head of the queue, an execution event run [ID] is sent to the task.

5.4. Automata Network of the Partition Scheduling System

Based on the formal descriptions the ARINC 653 partition scheduling system could be modeled as a parallel composition of a network of priced TA.

$T_0||T_1||\dots||T_n||Trigger_0||Trigger_1||\dots||Trigger_n||$PartitionScheduler||TaskScheduler1||TaskScheduler2,

where $T_0, T_1, \dots, T_n$ are tasks to be scheduled, $Trigger_0, Trigger_1, \dots, Trigger_n$ are task triggers, and Partition Scheduler, TaskScheduler1, and Task Scheduler 2 are a partition scheduler and task schedulers respectively. The automata modelling different parts of the scheduling system are run in parallel. The synchronization of state transitions is carried out by sending and receiving actions via shared channels.

6. Logical Description of Properties

The properties to be verified can be described in temporal logic formulas [6]. For the partition scheduling system, we studied system properties at the level of partition scheduling and task scheduling.

6.1. Partition Scheduling Level Verification

At the partition scheduling level, we were concerned with the following scheduling properties:

- *Property* 1: During a partition schedule period a partition *Si* (with index *i*) should never get more time budget than is initially configured for *Si*.

$\forall\Box$(PartitionScheduler.SchedulePartition$_i$ *imply* (PartitionScheduler.Part$_i$CLK <= PartitionScheduler. Part$_i$_Budget))

- *Property* 2: During a partition schedule period, partition *Si* (with index *i*) should never get less time budget than the initially configured budget for *Si*.

($\neg$ PartitionScheduler.SchedulePartition$_i$) *imply* (PartitionScheduler.Part$_i$CLK >= PartitionScheduler. Part$_i$_Budget)

- *Property* 3: During a partition schedule period, partition *Si* should always be released when its scheduling time reaches its initially configured budget.

(PartitionScheduler.Part$_i$CLK > Partition Scheduler. Part$_i$_Budget) *imply* ($\neg$ PartitionScheduler. SchedulePartition$_i$)

- *Property* 4: CPU could be allocated to TaskScheduler$_i$ (i=1,2).

$\forall\Diamond$TaskScheduler$_i$.Occ.

- *Property* 5: Task scheduler 1, Task scheduler 2, ... , and Task scheduler *n* are always mutually exclusive using a processor.

$\forall\Box$(TaskScheduler1.*Busy* + TaskScheduler2.*Busy* +... + TaskScheduler$_n$.*Busy* <= 1)

6.2. Task Scheduling Level Verification

At the task scheduling level, we were concerned that the following properties should be verified:

- *Property* 6: At any given point in time, CPU is used mostly by just one task.

The property could be described as $\forall\Box(T_1.Running + T_2.Running + \ldots + T_n.Running <= 1)$

- *Property* 7: For a specific task set and partition time budget configuration, the given task set is schedulable.

$\forall\Box(T_1.Error + T_2.Error + \ldots + T_n.Error = 0)$.

If schedule requirement can be satisfied during its execution, all the tasks will not move into the *Error* location.

- *Property* 8: If a partition scheduled with a priority preemptive policy is busy, the task with the highest priority must be the currently running task.

($\neg$ProcessScheduler1.Pended) *imply* (curThdPri >= getPri (head1()))

Parts of the other properties to be verified are as follows:

$\exists\Diamond$*Ti.Running*. There is a transition sequence from the initial state that eventually schedules *Ti* to execute at least once.

$T_i.Ready \rightarrow T_i.Running$. If the schedule request has been dispatched and task T_i begins to wait for scheduling, T_i could be scheduled eventually.

Ti.Running→Ti.WaitDispatch. If task Ti is scheduled to execute, it can be terminated successfully.

Ti.WaitDispatch→Ti.Ready. Periodic tasks should always be repeatedly activated to wait for scheduling.

$T_i.Ready \rightarrow (T_i.Running \wedge t<=s)$. When task T_i becomes ready it should be scheduled to execute within an *s* time unit. In the logic expression, *t* is a clock variable that is set to zero when the task changes to the ready state.

7. Analysis and Verification of The Schedule Model

7.1. Verification Data and Process

We used UPPAAL [6] as the verification tool. We chose a schedule case that contained 2 partitions. Each partition contained 10 schedule tasks. The scheduling policy of partition 1 was priority-based preemptive scheduling, while partition 2 was round robin scheduling. The scheduling system is presented in Table 1.

Table 1. Partition set.

Name	Budget	Scheduler policy	Tasks
Partition 1	1,000	Static priority preemptive scheduling	{task1,task2,...task10}
Partition 2	1,000	Round robin scheduling	{task11,task12,...task20}

The specifications, parameters, and time constraints of individual tasks of the experiment are presented in Table 2. The task data came from the schedule data of the Herschel-Planck [13] satellite system.

By inputting the ARINC 653 scheduling model into UPPAAL, a priced automata network was generated. The time shifting of the model evolution could be studied with a model simulator by observing the clock variation or by observing time points at which trace events happened. Furthermore, we simulated the evolution of the scheduling model by a simulator. The simulation process showed the current location, interactions, the network evolution, and the clock value of every automaton in the network. By step simulation we traced the evolution of the model, observed the time shift of the clock, analyzed the errors in automaton design, and checked whether the working mode met the expectations of the designer. The simulation process is shown in Figure 7.

Table 2. Task set.

taskid	Initialization time	Period	Execution time	Priority	Partition
1	0	10,000	13	1	P1
2	0	250,000	70	2	P1
3	62,500	125,000	70	3	P1
4	0	250,000	20	4	P1
5	200,000	250,000	100	5	P1
6	0	15,625	70	6	P1
7	0	20,000	70	7	P1
8	0	39,000	70	8	P1
9	0	250,000	70	9	P1
10	0	15,625	150	10	P1
11	0	125,000	400	-	P2
12	0	250,000	170	-	P2
13	20,000	25,000	5,000	-	P2
14	20,000	250,000	720	-	P2
15	20,000	250,000	230,220	-	P2
16	62,500	125,000	500	-	P2
17	62,500	250,000	6,000	-	P2
18	200,000	250,000	6,000	-	P2
19	200,000	250,000	3,000	-	P2
20	0	1,000,000	1,100	-	P2

7.2. Verification Result

The verification results were as follows:

At the partition scheduling level, *property* 1, *property* 2, *property* 3, *property* 4, and *property* 5 were satisfied. This also showed that at the partition scheduling level the design of the scheduling model was consistent with our expectations. At the task scheduling level, *property* 6 and *property* 8 were satisfied. For the specific task set, *property* 7 was satisfied. This meant that for the given task set the scheduling system was schedulable. In the experiment, if we had changed the task set or the task parameters, the task set may have changed to non-schedulable. For a non-schedulable case, we can analyze the reason that it is non-schedulable by tracking the system evolution.

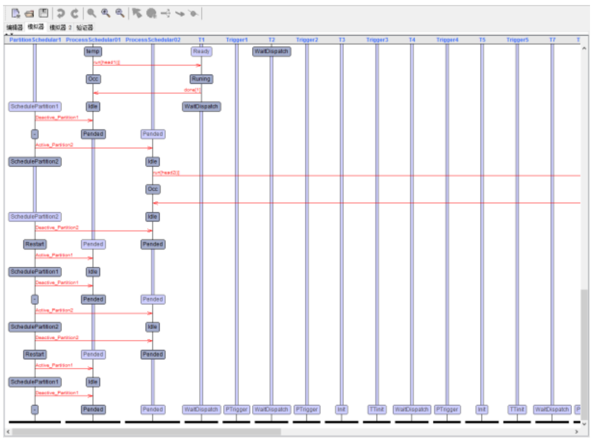


Figure 7. The step simulation of the ARINC 653 scheduling model.

Model checking uses an exhaustive method to analyze every possible state in the system evolution. Therefore, it provides us a way to precisely analyze the running states and time properties for each entity in the ARINC 653 partition scheduling system. As an example, we can input this logic expression into the verification system: E<>T_0.Running and T_0.t <= 0. The property determines whether there exists a possible running trace that gives T_0 execution immediately without waiting.

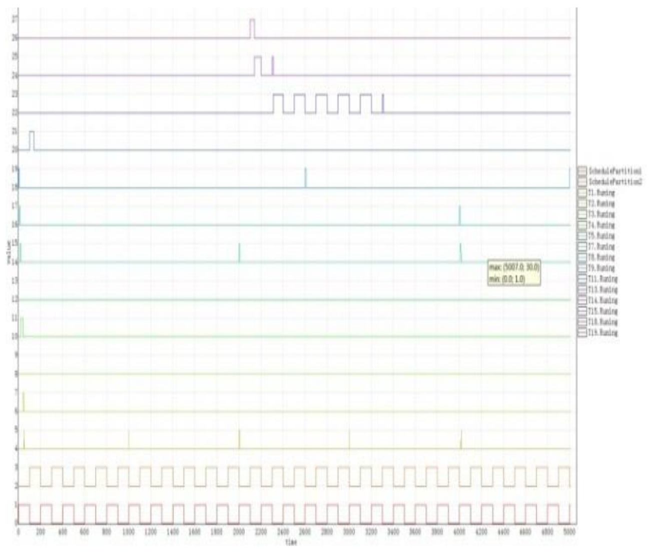


Figure 8. The simulation of the ARINC 653 scheduling model.

The simulation results of the experimental data of this paper are shown in Figure 8. The simulation makes it possible for us to visually observe the process of scheduling, the dispatching of a time slice, the CPU scheduling of each task, and the running state of each entity in the system. In addition, the simulation provides a visual method to precisely analyze the scheduling and execution states of the scheduling system.

Figure 9 shows the comparison of time required to deem periodic schedulable task sets in paper [20] and our method. It can be seen the method we presented has a better time performance. Figure 10 shows the time performance of our method. In the beginning, with the number of tasks increases, the verification time increase slowly. When the number of tasks verified is greater than 60, time tends to grow faster. When the number of verification tasks is 100, the required verification time is 31.38s. When the number of tasks to be verified is greater than 100, the time tends to grow more and more faster with the increase of the number of tasks.

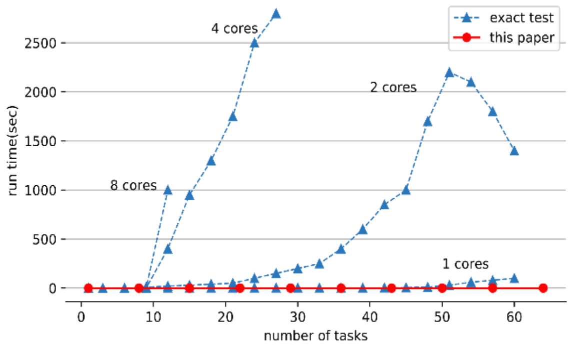


Figure 9. Average runtime of the exact test [20] vs. our test for schedulable task sets.

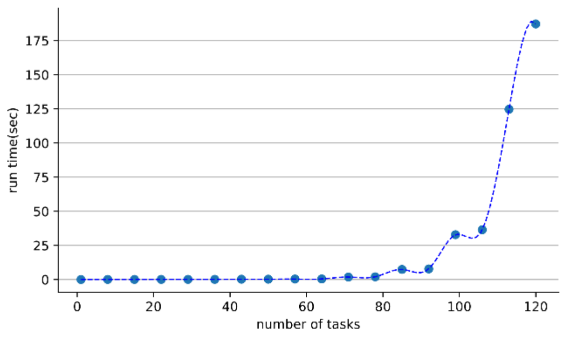


Figure 10. Average runtime of verification.

8. Conclusions

In this paper we discussed how to use a formal method and verification tools to analyze and verify the ARINC 653 hierarchical scheduling system. Based on priced TA theory, an ARINC 653 scheduling system, including partitions, tasks, and a hierarchical scheduler was modeled as a priced TA network. The schedulability of the system was described as a set of temporal logic expressions, and schedulability was analyzed and verified by model checking. In addition, we discussed the analysis of a specific ARINC 653 partition scheduling case. The method proposed solved the modelling of preemptive scheduling by introducing priced TA. Our research shows that it is workable to use verification tools such UPPAAL to analyze the schedulaibity of tasks set in an ARINC 653 hierarchical scheduling system. Unlike traditional scheduling analysis methods, the formal method we propose exhaustively analyzes the schedulability of a system, resulting in a more precise analysis.

In future distributed multicore Integrated Modular Avionics systems (IMA) [18], the scheduling mechanism and task model will be changed, so future work for us is to study the analysis and verification of the scheduling process in that distributed multicore environment.

Acknowledgements

This research was supported by The National Key Research and Development Program of China (No: 2017YFB1001902).

References

[1] Alur R., Courcoubetis C., Halbwachs N., Henzinger T., Ho P., Nicollin N., Olivero A., Sifakis J., and Yovine S., "The Algorithmic Analysis of Hybrid Systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3-34, 1995.

[2] Alur R. and Dill D., "A Theory of Timed Automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183-235, 1994.

[3] Amnell T., Fersman E., Mokrushin L., Petterssou P., and Yi W., "TIMES B-A Tool for Modelling and Implementation of Embedded Systems," *in Proceedings of Tools and Algorithms for the Construction and Analysis of Systems*, Grenoble, pp. 460-464, 2002.

[4] Asberg M., Pettersson P., and Nolte T., "Modelling, Verification and Synthesis of Two-Tier Hierarchical Fixed-Priority Preemptive Scheduling," *in Proceedings of 23rd Euromicro Conference on Real-Time Systems*, Porto, pp. 172-181, 2011.

[5] Aziz M. and Shah S., "Evolutionary Testing for Timing Analysis of Parallel Embedded Software," *The International Arab Journal of Information Technology*, vol. 16, no. 3, pp. 415-423, 2019.

[6] Behrmann G., David A., and Larsen K., *A Tutorial on Uppaal*, Springer, 2004.

[7] Burns A., *Advances in Real-Time Systems*, Upper Prentice-Hall, 1994.

[8] Committee A., "Avionics Application Software Standard Interface Part 1-Required Services," *ARINC Specification 653P1-2, Technical Standard*, Aeronautical Radio Inc., Annapolis, Maryland, 2006.

[9] Daum M., Dörrenbächer J., and Wolff B., "Proving Fairness and Implementation Correctness of A Microkernel Scheduler," *Journal of Automated Reasoning*, vol. 42, no. 2-4, pp. 349-388, 2009.

[10] Davis R. and Burns A., "Hierarchical Fixed Priority Pre-Emptive Scheduling," *in Proceedings of 26th IEEE International Symposium on Real-Time Systems*, Miami, pp. 388-398, 2005.

[11] Glaubius R., Tidwell T., Smart W., and Gill C., "Scheduling Design and Verification for Open Soft Real-Time Systems," *in Proceedings of IEEE International Symposium on Real-Time Systems*, Barcelona, pp. 505-514, 2008.

[12] Liu Q., Gui S., Luo L., and Li Y., "Schedulability Verification of AADL Model Based on UPPAAL," *Journal of Computer Applications Computer Applications*, vol. 29, no. 7, pp. 1820-1824, 2009.

[13] Mikučionis M., Larsen K., Rasmussen J., and Nielsen B., *Schedulability Analysis Using Uppaal: Herschel-Planck Case Study*, Springer, 2010.

[14] Nasri M., Nelissen G., and Brandenburg B., "Response-Time Analysis of Limited-Preemptive Parallel DAG Tasks under Global Scheduling," *in Proceedings of Euromicro Conference on Real-Time Systems*, Dagstuhl, pp. 1-23, 2019.

[15] Regehr J., Reid A., Webb K., Parker M., and Lepreau J., "Evolving Real-Time Systems Using Hierarchical Scheduling and Concurrency Analysis," *in Proceedings of 24th IEEE International Symposium on Real-Time Systems*, Cancun, pp. 25-36, 2003.

[16] Serrano M., Melani A., Bertogna M., and Quinones E., "Response-Time Analysis of DAG Tasks under Fixed Priority Scheduling with Limited Preemptions," *in Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, Dresden, pp. 1066-1071, 2016.

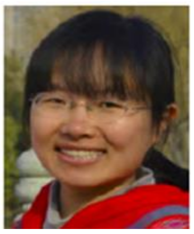
[17] Sokolsky O. and Chernoguzov A., "Analysis of AADL Models Using Real-Time Calculus with Applications to Wireless Architectures," Technical Report No.MS-CIS-08-25, University of Pennsylvania, 2008.

[18] Wang T. and Qingfan G., "Research on Distributed Integrated Modular Avionics System Architecture Design and Implementation," *in Proceedings of 32nd IEEE/AIAA International Conference on Digital Avionics Systems*, East Syracuse, pp. 1-53, 2013.

[19] Waszniowski L. and Hanzálek Z., "Formal Verification of Multitasking Applications Based on Timed Automata Model," *Real-Time Systems*, vol. 38, no. 1, pp. 39-65, 2008.

[20] Yalcinkaya B., Nasri M., and Brandenburg B., "An Exact Schedulability Test for Non-Preemptive Self-Suspending Real-Time Tasks," *in Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Florence, pp. 1-8, 2019.

Ning Fu is an assistant research fellow at the School of Computer, Northwestern Polytechnical University, Xi'an. He received his Ph.D. (2010) degree in computer science and technology from Northwestern Polytechnical University. His main research interests focus on modelling and verification embedded system by formal methods.



Lijun Shan is a lecturer at the School of Computer, Northwestern Polytechnical University, Xi'an. She received her Ph.D. degree in computer science and technology from National University of Defense Technology. Her main research interests include Model-Driven Development Method and Formal Method.

Chenglie Du is a professor, Ph.D. supervisor at the School of Computer, Northwestern Polytechnical University. His main research interests include Cyber-Physical system modelling & analysis and trustworthy software architecture.

Zhiqiang Liu is an associate professor at the School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an. His current research interests focus on modelling and verification embedded system by formal methods.

Han Peng is a PhD candidate at the School of Computer, Northwestern Polytechnical University, Xi'an. His current research interests focus on modelling and verification Cyber-Physical system by formal methods.