# A Deterministic Analysis Method of Embedded System Based on Event-driven

X.C. Shi[1], Y.A. Zhu[1], X.Y. Zhang[1], L. Li[2], Z. L. Qi[1], J. H. Dou[2]
[1]School of Computer Science, Northwestern Polytechnical University, Xi'an, China
[2]School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an, China
(shixianchen@mail.nwpu.edu.cn, zhuya@nwpu.edu.cn, lilian@nwpu.edu.cn,
zhangxiangyu2017@126.com, qizonglong@mail.nwpu.edu.cn, jhdou@mail.nwpu.edu.cn)

*Abstract* - **This paper elaborates on the deterministic analysis method for event-driven embedded system to reduce the impact of uncertain factors at the design stage. By abstracting each unit of the embedded system, the framework of the embedded system is built. Margin analysis of time resource and space resource is carried out to provide accurate reference for the designers and realize reasonable planning of resources, task priority and task scheduling.**

*Keywords* - **determinism, margin analysis, schedulability**

## I. INTRODUCTION

As an abstract notion which usually refers to high level requirements, determinism is described in DO-297 as "The ability to produce a predictable outcome generally based on the preceding operations, the outcome occurs in a specified period of time with some degree of repeatability". Generally, a system is deterministic if its behavior is ruled by a set of identified laws which are compatible with certification objectives[1].

To ensure the outcome of the software execution prediction, three requirements should be met. First, ensure the system resources to prevent the abnormal exit of task due to insufficient resources. Second, the task schematization should be taken into consideration, including task priority, task dependency and task scheduling strategy[2,3], etc. Thirdly, interrupt should be taken into account. Interrupt can be divided into periodic interrupt and aperiodic interrupt, which have different influences on embedded software. Whether the interrupt be reasonably abstracted is critical for deterministic analysis[4].

Based on the above problems, tasks of the embedded system are abstracted, which increases the attribute of resource requirement and carries out the margin analysis to ensure that the system resources of the embedded software can meet the margin requirements. Establish a task scheduling mechanism with dependencies, and require tasks to meet a certain time margin to increase the reliability of task execution; for interrupts, establish periodic and aperiodic interrupts models, especially for aperiodic interrupts , we assume that the interrupts meet a certain distribution function in the simulation cycle, and then use the selection method to generate a random number that meets the specified distribution to achieve the simulation of aperiodic interrupts. Finally, the framework is built, based on which the deterministic analysis of the task execution process is carried out, and the we can get detailed results of system resource usage during task execution.

## II. EMBEDDED SYSTEM MODELING

In this section, based on the original interrupt, event and scheduling strategy model, we optimize the model to support margin analysis [5,6,7].

### *A. Interrupts*

In embedded systems, interrupts are generally divided into two types: periodic interrupts and aperiodic interrupts. Periodic interrupts mean that there is a fixed time interval between arrival time of the two interrupts. For aperiodic interrupts, trigger time of interrupts is defined to satisfy a specific distribution function within certain interval[8,9].

1) Model of periodic interrupts: We use Ip=< id, type, priority, off, period, resptime, exetime, pubevent, funcunit> to denote periodic interrupt where id is the unique identifier of a periodic interrupt; type is the type of interrupt, priority is the interrupt priority; the higher the value; the higher the priority; off is the time offset between period and start time; period is the time interval between two interrupts; resptime is the interrupt response time that elapses between the interrupt signal coming in at the interrupt controller and the execution of the first statement in the corresponding interrupt handler; exetime is the interrupt execution time; pubevent is the event generated by interrupt, which will trigger corresponding function unit; funcunit is the function unit, to achieve specific functional requirements; interrupt service program is also part of the function unit[10].

2) Model of aperiodic interrupts: Aperiodic interrupt is defined as interrupt which follows specific distribution within a time interval. Acceptance-Rejection Method (ARM)[11,12] is used to generate random number for simulating the occurrence time of aperiodic interrupt. Aperiodic interrupts are characterized by a tuple $I_{ap}$=<id, type, priority, off, F(x), timeinterval, resptime, exetime, pubevent, funcunit >.

Aperiodic interrupt is different from periodic interrupt from the following aspects:

- *F(x)*: distribution function of aperiodic interrupt.
- *timeinterval*: value range of *x*.

The ARM is an algorithm of generating random samples from an arbitrary probability distribution. Let x

be a random variable with probability density function f(x), then set a<b and P{a<X<b}, the procedure is:

- step 1:λf(x) < 1, x∈ (a, b);
- step 2: Generate $r_1$, $r_2$ with uniform distribution, y = a + ( b – 1 ) $r_i$;
- step 3:if $r_2 \leqq \lambda f(x)$, x = y, otherwise, reject $r_1$, $r_2$, go back to step 2,repeating until we obtain a value of y in step 3.

For example aperiodic interrupt occurrence time T ~ N( μ, $\sigma^2$ ), μ=50, σ=10, and its probability density function is:

$$f(x)=\frac{1}{\sqrt{2\pi}\times 10}\exp(-\frac{(x-50)^2}{2\times 10^2})$$

The random number obtained by ARM method is shown in Fig. 1.

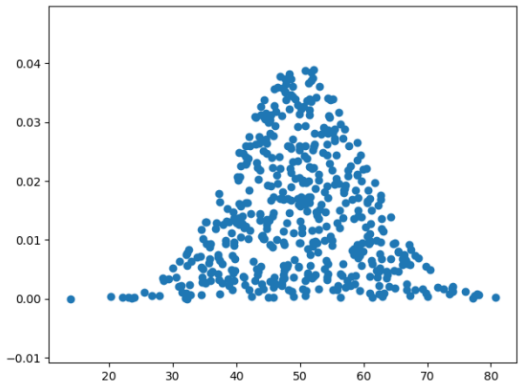


Fig. 1. Generation of random numbers of normal distribution：Using Accept-Reject method

## B. Events

Event of the embedded system, which is based on event-driven mechanism, is the system function executed by signal triggering function unit. In addition, event will also trigger the state transition of function unit state machine. Interrupts can generate events, and functional units can also generate events during operation. Event is characterized by a tuple E = <id, size, event pool> where id is the event identifier; size is the memory used by the event; event pool is divided into three types: large, medium and small. The event is put into appropriate event pool according to its size.

## C. Function unit

The event-driven embedded system distributes event to the function unit, which can complete specific system tasks. State machine is a common method of event driven system analysis[13], so this paper uses state machine as a function unit to achieve the following three concepts related to hierarchical state machine:

- Event: used to trigger state machine behavior in a specific state, including state machine behavior triggering and transition;
- State: the system satisfies certain conditions in its life cycle, performs certain behaviors or waits for certain events to occur;
- Transition: migration from one state node to another.

Taken state nesting in embedded system, hierarchical state machine (HSM) is also a description of function units. [14].

The hierarchical state machine introduces the concept of behavior inheritance, and the behavior of the parent state can be reused by the child state. Therefore, the child state does not need to care about the treatment of public behavior, but to define the behavior different from the parent state, which effectively reduces the complexity of the state machine. The simulation framework based on hierarchical state machine should include event queue and event distribution mechanism in addition to functional units. Therefore, each function unit maintains an event queue. When the functional unit receives a new event, the event will be inserted into the event queue and trigger the functional unit in turn.

Function units are characterized by a tuple FU=<id, priority, statestable, subsevent, pubevent, exectime, WCET, queuevolume> where id is the unique identifer of function unit; priority is the priority of functional units; satestable is the status of functional units and transition rules of status; subsevent is the events subscribed by function unit; pubevent is the events published by functional unit; exetime is the execution time of function unit; WCET is the worst-case execution time of the functional unit; usually consistent with the deadline; queuevolume is the Capacity of function unit's queue.

## D. Scheduling strategy

This paper analyzes preemptive scheduling and non-preemptive scheduling in event-driven embedded system.

In the non-preemptive scheduling, the high priority task can not preempt the low priority task. Once the task starts to run, it would voluntary give up the processor resources, otherwise it will continue to execute until the task is completed and the processor resources are released. When there is an interrupt, the interrupt may trigger the high priority task and fall in the ready state[15]. When the interrupt service is completed, it still returns to the original interrupted task, and the high priority task will not be executed until the task is completed. The non-preemptive scheduling is shown in Fig. 2.

In preemptive scheduling, when there are high priority tasks in the system, the running low priority tasks in the system will be suspended and high priority tasks will be executed, and the low priority tasks will continue to be executed after the high priority tasks are completed. If the interrupt service program triggers the high priority task and switch to the ready state, when the interrupt is completed, the interrupted task is still suspended, the high priority task entering the ready state is executed first, and

the low priority task will continue to be executed only after the high priority task is executed. The preemptive scheduling mode is shown in Fig. 3.

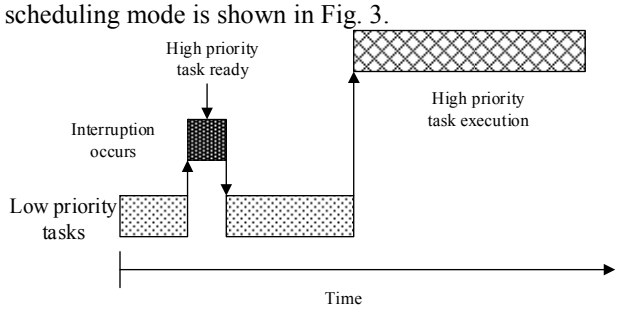


Fig. 2. Non-Preemptive scheduling

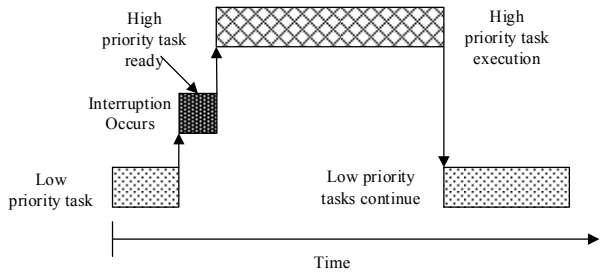


Fig. 3. Preemptive scheduling

## III. SIMULATION FRAMEWORK

The simulation framework model combines the above models and provides the following functions:

1) Events distribution management: Event distribution function. When an event is generated by a function unit or interrupt, it will be distributed to function unit that subscribes to the event;

2) Event pool: When an event occurs, the event will be filled into the appropriate event pool according to the amount of memory size;

3) Space margin analysis: The space margin analysis is to analyze the usage of event pool and event queue in the simulation process, indicating the ratio of residual resources to system resources under the maximum usage of a resource. The margin of event pool and event queue is defined as follows:

$$PSM = (1 - \frac{PD_i}{PV_i}) \times 100\% \quad (1)$$

$$QSM = (1 - \frac{QD_i}{QV_i}) \times 100\% \quad (2)$$

Where:PSM is the Event pool space margin, $PD_i$ is the Used depth of event pool; i represents three event pools; large, medium and small; $PV_i$ is Event pool capacity; QSM is the Event queue space margin; $QD_i$ is the depth that the event queue has been used; j represents Function unit number to which event queue belongs; $QV_i$ is the Capacity of event queue.

4) Time margin analysis: On the one hand, time margin analysis indicates task schedulability, on the other hand, it quantitatively analyzes the time usage of the system as a whole and each functional unit. Time margin represents the ratio of the time when the system is idle to the total time. The time margin of a single task is defined as follows:

$$TM_{FU} = (1 - \frac{RT_{FU}}{WCET_{FU}}) \times 100\% \quad (3)$$

Where: $TM_{FU}$ is the Time margin of functional unit, $RT_{FU}$ is the Response time of functional unit, $WCET_{FU}$ is the worst-case execution time of function unit.

As shown in Fig. 4, the function unit FU is starts from $T_1$, is interrupted by other high priority functional units at $T_2$ and $T_4$, continues to execute at $T_3$ and $T_5$, and completes at $T_6$. The response time of the function unit is from the beginning of the time $T_1$ to the completion of the time $T_6$, execution time is the sum of $<T_1, T_2>$、$<T_3, T_4>$ and $<T_5, T_6>$, $<T_1, T_7>$ is the WCET of function unit. Therefore, the time margin of this function unit is $TM_{FU} = (1 - \frac{< T_1, T_6 >}{T_1, T_7}) \times 100\%$.

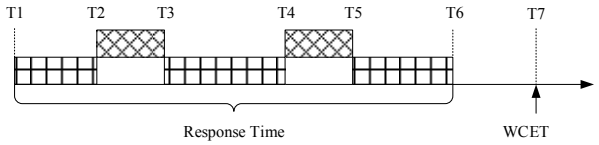


Fig. 4. Function unit execution process

The system time margin is formally represented as

$$TM_{SYS} = (1 - \frac{\sum_{i=1}^{n} (T_{ISR\_R} + T_{ISR\_E})_i + \sum_{i=1}^{m} T_i^{CS} + \sum_{i=1}^{k} T_i^{FU\_E}}{T_{SYS}}) \times 100\% \quad (4)$$

where $TM_{SYS}$ is the System time margin, n is the number of interrupts, m is the context switching times, k is the number of function unit execution, $T_{ISR\_R}$ is the interrupt response time, $T_{ISR\_E}$ represent interrupt execution time, $T_i^{CS}$ represent context switching time, $T_i^{FU}$ is the function unit execution time, $T_{SYS}$ is the total system time.

## IV. EXPERIMENT

This paper takes an aero-engine test bed control system as the test object, simulates the aero-engine start-up process, and analyzes the certainty of the engine test bed control system through the simulation framework model to verify the rationality of the engine control system resource allocation and mission planning.

The aero-engine startup process is shown in Fig. 6.

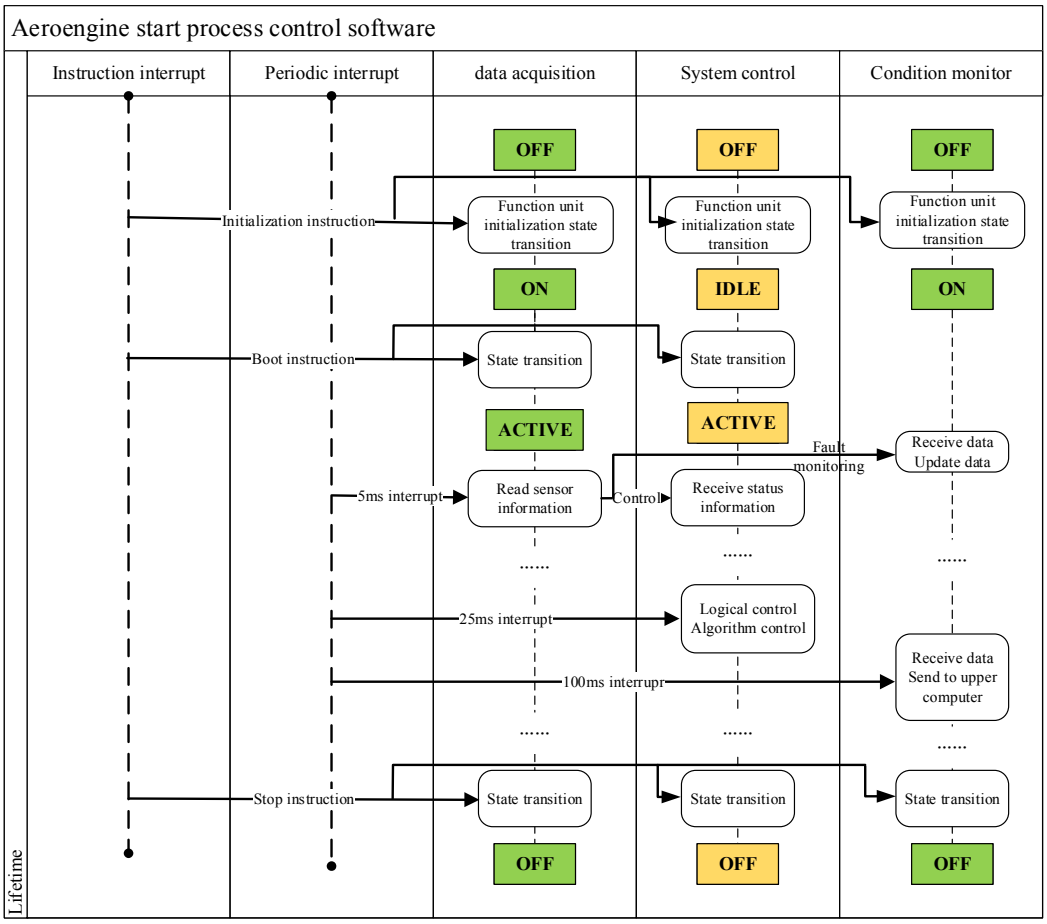


Fig. 5. Aeroengine startup process

*A. Models*

According to the principle of H. gomma, the engine control system is divided into three active objects, which are data acquisition function unit, engine control function unit and status monitoring unit. The model configuration of the three active units is shown in Table I.

TABLE I DEFINITION OF FUNCTIONAL UNIT

| **Id** | **DAQ** | **System Control** | **Status Monitor** |
|---|---|---|---|
| Priority | 3 | 2 | 1 |
| States | OFF/ON/ ACTIVE | OFF/IDLE/ ACTIVE | OFF/ON |
| Subsevent | Initial/Start/Stop/5TimeOut | Initial/Start/Stop/25TimeOut/System Control | Initial/Stop/100TimeOut/Fault Monitor |
| Pubevent | System Control/Fault Monitor | None | None |
| Exectime | 1/1/1/2 | 1/1/1/8/2 | 1/1/10/1 |
| WCET | 3/3/3/4 | 3/3/3/20/4 | 3/3/80/3 |
| Queue volume | 100 | 100 | 100 |

There are six interrupts, three periodic interrupts and three aperiodic interrupts in the starting process of aeroengine. The interrupt model is shown in Table II.

TABLE II INTERRUPT MODELING

| Id | 5TimeOut | 25TimeOut | 100 TimeOut | Initial | Start | Stop |
|---|---|---|---|---|---|---|
| Type | P | P | P | AP | AP | AP |
| Priority | 3 | 2 | 1 | 6 | 5 | 4 |
| Off | 10 | 101 | 10 | 0 | 4 | 1010 |
| Resptime | 0 | 0 | 0 | 0 | 0 | 0 |
| Exetime | 0 | 0 | 0 | 0 | 0 | 0 |
| F(x) | - | - | - | N(0,1) | N(0,1) | N(0,1) |
| Time interval | - | - | - | [0,4] | [4,10] | [1010,1020] |
| Pubevent | 5TimeOut | 25TimeOut | 100 TimeOut | Initial | Start | Stop |
| Function unit | DAQ | System Control | Status Monitor | DAQ/System Control/ Status Monitor | DAQ/System control | DAQ//System control /Status Monitor |

In the startup process, the system drives the functional unit through eight events, among which initial, start and stop are events generated by non periodic interrupt, 5timeout, 25timeout and 100timeout are events generated by periodic interrupt, system control and fault monitor are events generated by functional unit, and the event model is shown in Table II.

TABLE III EVENT MODEL

| id | Initial | Start | Stop | 5TimeOut | 25TimeOut | 100TimeOut | System Control | Fault Monitor |
|---|---|---|---|---|---|---|---|---|
| size | 15 | 15 | 15 | 25 | 25 | 25 | 25 | 25 |
| Event pool | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 |

Fig.6 shows the relationship between all functional units and events.

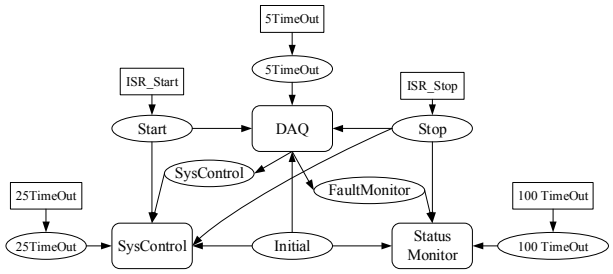


Fig. 6. Event relationship

## B. Results

According to the defined engine control system model, the time margin and system space margin analysis results of each functional unit can be obtained by simulation.

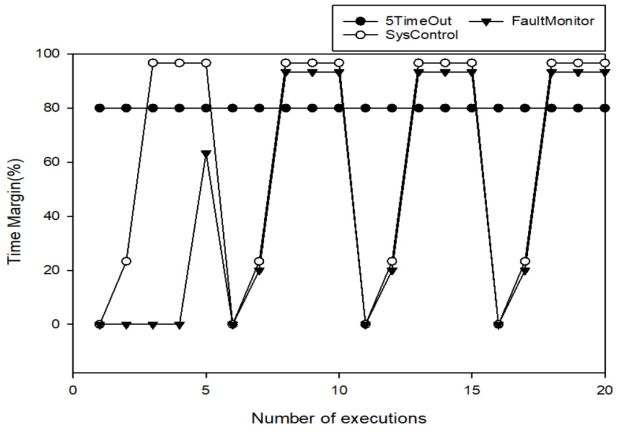


Fig. 7. Data acquisition time margin analysis results

From Fig. 7, it can be seen that in the simulation process, in the data collection function unit, the number of occurrence of each event is 20, and the time margin of event 5timeout is 80%, which remains unchanged; the time margin of event system control changes from 0 to 97.8%; the time margin of event fault monitor changes from 0 to 60%.

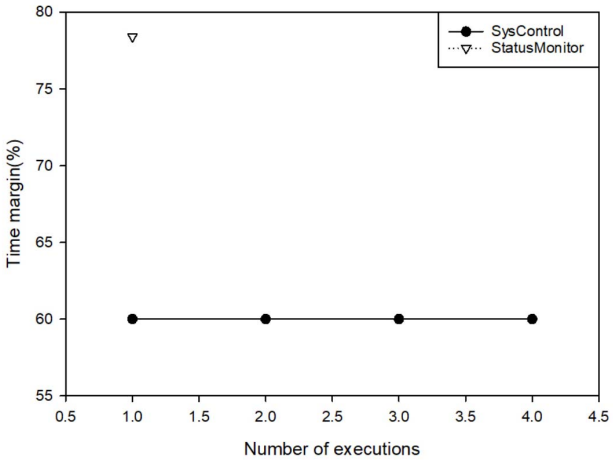


Fig. 8. Analysis results of system control time margin

It can be seen from Fig. 8 that the control time margin of the time system has been kept at 60%, with 4 times of occurrence, while the condition monitoring event only occurs once, with a time margin of 78.4%.

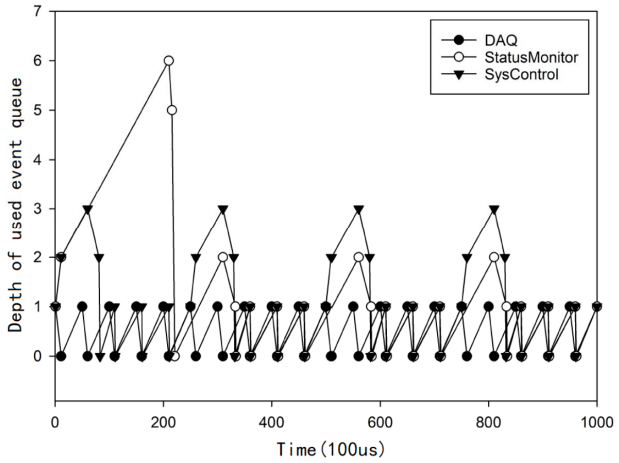


Fig. 9. Depth of used event queue

It can be seen from Fig. 9 that with the change of time, the queue usage of the functional unit has also been changing, among which the maximum queue usage depth of the status monitoring is 6, the maximum queue usage depth of the status monitoring is 2, and the maximum queue usage depth of the system control is 4. According to the calculation formula of the space margin, the minimum space margin of the queue is 94%, 98% and 96%.

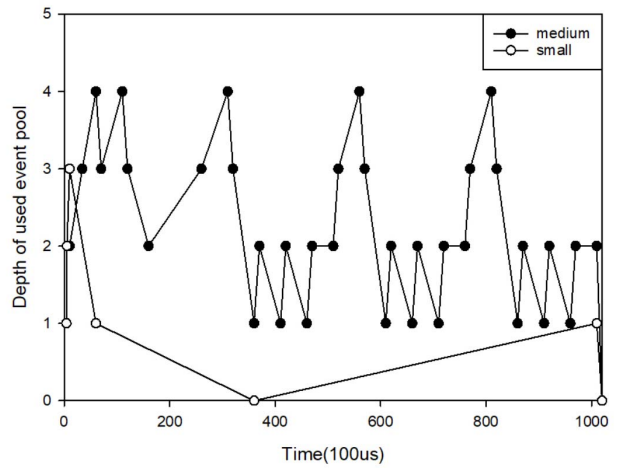


Fig. 10. Depth of used event pool

It can be seen from Fig. 10 that the large event pool is not used, and the maximum use depth of the medium event pool is 4, so the minimum space margin is 96, and the maximum use depth of the small event pool is 3, so the minimum space margin is 97%.

## V. CONCLUSION

In this paper, we presented a novel simulation framework for embedded system deterministic analysis. Based on the interrupt model, event model, function unit model, scheduling strategy model and the final simulation framework model, we can set the configuration of embedded system, and then get the time margin of each event in the system through simulation, and give the use of event pool and event queue, and finally get the space margin results. Even if the interrupt and task of the system can meet the schedulability, in order to ensure the reliability of the execution process of the system, if it does not meet the preset margin index, it is still determined that the system does not meet the certainty requirements.

From modeling and simulating the embedded system, we can get the performance verification of different operation scenarios under the preemptive and non-preemptive scheduling modes, help the designers to carry out the embedded software architecture simulation in the early stage of software development, and quantitatively analyze and evaluate the time and space performance of the software system architecture, It can provide effective support for the estimation of resource consumption, hardware selection and platform adaptation, improve the design efficiency of embedded software and ensure the software reliability.

## ACKNOWLEDGMENT

The authors acknowledge financial support from the ZFPR Foundation of China (Project No. 31511070301).

## REFERENCES

[1] RTCA DO-297, Integrated Modular Avionics(IMA) Development Guidance and Certification Considerations.

[2] Umarani Srikanth, G., Shanthi, A. P., Uma Maheswari, V. & Siromoney, A. “A survey on real time task scheduling”, European Journal of Scientific Research,pp: 33–41 2012.

[3] Grolleau E . “Introduction to Real□Time Scheduling”, Real□Time Systems Scheduling 1. John Wiley & Sons, Ltd, 2014.

[4] Girbal, S. , Jean, X. , Rhun, J. L. , Perez, D. G. , & Gatti, M. “Deterministic platform software for hard real-time systems using multi-core COTS”, IEEE/AIAA Digital Avionics Systems Conference IEEE, 2015.

[5] Stephen A. Edwards and Olivier Tardieu. “SHIM: a deterministic model for heterogeneous embedded systems”. In Proceedings of the 5th ACM international conference on Embedded software (EMSOFT’05). Association for Computing Machinery, New York, NY, USA, pp: 264–272.2005

[6] Moallemi, M. “Real-Time and Embedded Systems Development based on Discrete Event Modeling and Simulation”, Carleton University,2011.

[7] Niyonkuru D , Wainer G A . “Discrete-Event Modeling and Simulation for Embedded Systems”, Computing in ence & engineering, pp:52-63, 2015.

[8] Mignogna, Alessro & Ferrante, Orlo & Carloni, Marco & Ferrari, Alberto. “A Fully Configurable RTOS Model for Large Scale Distributed Embedded Systems Simulations based on SystemC”,2011.

[9] Brylow, D, & Palsberg, J. “Deadline analysis of interrupt-driven software”, IEEE Transactions on Software Engineering, pp: 634-655, 2004.

[10] Resmerita, Stefan & Poelzleitner, Anton & Lukesch, Stefan. “Modeling and Simulation of Software Execution Time in Embedded Systems” presented at the 10th Annual Computing and Communication Workshop and Conference pp: 0888-0894, 2020.

[11] Gass, Saul I, Fu M.C, “Acceptance-Rejection Method”, Encyclopedia of Operations Research and Management Science. Springer, Boston, MA, 2013
[12] Method, A. “Acceptance-rejection method”, encyclopedia of operations research & management science, 2004.
[13] Opeyemi O. Adesina, Timothy C. Lethbridge, Stéphane S. Somé, Vahdat Abdelzad, & Alvine Boaye Belle. “Improving formal analysis of state machines with particular emphasis on and-cross transitions”, Computer Languages, Systems & Structures, pp: 544-585, 2018.
[14] Fragal, Vanderson Hafemann, A. Simao, and M. R. Mousavi. "Hierarchical featured state machines." Science of Computer Programming, 2018.
[15] Dongping Huang, and H. Sarjoughian. “Software and Simulation Modeling for Real-Time Software-Intensive Systems”, IEEE International Symposium on Distributed Simulation & Real-time Applications IEEE, 2004.