实时嵌入式系统的竞态条件及其分析方法研究

陈 艳[1] 徐晓峰[2] 李晓潮[1,3] 郭东辉[1,2,3]
[1](厦门大学信息科学与技术学院 福建厦门 361005)
[2](厦门大学物理系 福建厦门 361005)
[3](福建省集成电路设计工程技术研究中心 福建厦门 361005)
(dhguo@xmu.edu.cn)

Race Condition and Its Analysis Approach of Real-time Embedded Systems

Chen Yan[1], Xu Xiaofeng[2], Li Xiaochao[1,3], and Guo Donghui[1,2,3]
[1](*School of Information Science and Technology, Xiamen University, Xiamen, Fujian* 361005)
[2](*Department of Physics, Xiamen University, Xiamen, Fujian* 361005)
[3](*Fujian IC R&D Engineering Center, Xiamen, Fujian* 361005)

Abstract In real-time embedded systems, due to race conditions, synchronization order of operations to the shared variables or shared resource during the multiple tasks may be different from one execution to another. This may cause abnormal behaviors of systems. In order to detect the possible race conditions and analyze the impacts of race conditions effectively in real-time embedded systems, a formal model of execution sequences and operation events are presented according to the timing behaviors and execution characteristics of real-time embedded systems. Their characteristics are also discussed. Based on the execution sequence model, race conditions including message races and semaphore races in real-time embedded systems are described formally and precisely. And then, a new race set is presented to describe and store race conditions in systems. It includes the information of happened-before relations and race synchronization relations among the operation events which have races. With the race set generated, a race condition graph is constructed to visualize the race conditions. It is also used to predict the potential race synchronization relations of systems. The case study shows that the approach proposed can be used to analyze and predict efficiently the potential race synchronization relations as well as the different execution situations and results of real-time embedded systems.

Key words real-time embedded system; execution sequence; race condition; race set; race condition graph

摘 要 竞态条件使得多个任务对共享资源进行操作的先后顺序在不同的执行情况下发生改变,从而可能引起系统异常.为了分析实时嵌入式系统可能出现的竞态条件及其所带来的影响,根据目标系统的执行特征,建立系统的执行序列模型,对相关的竞态条件进行精确描述,并在此基础上提出一种竞态集来存储和分析系统的竞态条件,然后利用获取到的竞态集,构建系统的竞态条件图来预测系统潜在的竞态同步关系.实验分析表明该方法能够有效地分析和预测目标系统各种可能的竞态同步关系及其所带来的不同执行情况和结果.

收稿日期: 2009-04-08; 修回日期: 2009-11-30
基金项目: 国家自然科学基金项目(60753001); 教育部新世纪人才计划基金项目

关键词 实时嵌入式系统; 执行序列; 竞态条件; 竞态集; 竞态条件图

中图法分类号 TP316

0 引 言

由于实时嵌入式系统在任务调度及任务间通信同步以及外界环境影响等方面都存在固有的不确定性因素, 所以可能引起竞态条件(race condition)[1]的出现, 即多个任务对共享资源进行操作(如获取信号量、释放信号量、发送消息、接收消息等)的先后顺序在不同的执行情况下发生变化, 从而可能造成系统出错、死锁甚至崩溃等后果. 因此, 只有有效地分析系统可能出现的竞态条件及其所带来的影响, 才能保障系统的安全运行[2].

目前已有的用于竞态条件分析与检测的方法可分为静态和动态 2 种. 其中静态法[3-5]是指通过直接分析系统程序的源代码, 获取程序中所有可能的执行路径, 从中分析出系统的竞态条件. 该方法的优点是没有运行时间开销, 且能够分析出系统所有可能的竞态条件, 但是由于静态分析技术无法精确考虑系统运行时的任务切换及任务间通信同步等情况, 所以得到的很多竞态条件在系统实际运行中都可能无法出现, 而且如果目标程序可达路径的数量很大, 则可能引起状态空间爆炸问题[5]. 为了克服静态法的缺点, 人们又分别提出在线(on-the-fly)和线下(post-mortem)2 种动态方法来分析系统的竞态条件. 其中在线分析方法[6-9]是指在系统程序运行过程中存储部分的执行信息, 并对该信息进行在线分析, 判断是否存在竞态条件; 而线下分析方法[10-11]则是在程序运行期间追踪记录程序的执行信息, 并在线下对保存下来的执行信息进行分析, 得出可能存在的竞态条件. 动态法不存在状态空间爆炸问题, 虽然具有一定的运行时间或空间开销, 但是由于是对系统实际执行情况进行分析, 因此与静态法相比, 分析得到的竞态条件更加精确[12].

目前, 不管是在线分析方法还是线下分析方法, 竞态条件的动态分析方法一般都采用时空图(space-time diagram)[13-14](如图 1 所示)把系统执行信息描绘出来, 再通过分析任务间通信同步关系来获取竞态条件. 然而, 由于时空图只能描述系统执行事件之间的发生先后关系(happened-before relation)[13], 不能用于竞态条件的存储、计算和后续分析, 因此学者们还提出各种图形来辅助分析系统的竞态条件. 如文献[15]中提到一种图形可以用来描述数据库应用程序中的竞态条件, 该图用节点表示系统执行信息中的每个执行事件, 用边表示事件之间的发生先后关系和竞态关系, 但是由于该图包含执行信息中的所有事件及其同步关系, 因此其空间开销较大. 文献[16]中提到一种叫作 POEG 的图形可用来辅助分析系统的竞态条件, 与文献[15]类似, 这种图形也是把执行信息中的所有事件用节点表示出来, 不同的是该图形中的边只描述系统执行事件之间的发生先后关系. 文献[17]提出一种消息序列图(UML/MSC)来描述消息传递事件之间的偏序场景(partial order scenario), 并可对其进行竞态条件分析, 但是这种图形无法描述系统其他类型的执行事件, 缺乏通用性.

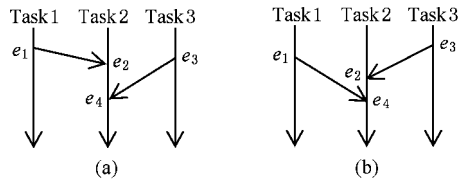


Fig. 1 An example of message race. (a) Execution sequence Q and (b) Execution sequence Q'.

图 1 消息竞态实例. (a) 执行序列 Q; (b) 执行序列 Q'

针对上述竞态条件动态分析方法的不足之处, 本文提出竞态集和竞态条件图来辅助分析实时嵌入式系统的竞态条件. 首先根据实时嵌入式系统的执行特征, 建立系统的执行序列模型, 对相关的竞态条件进行精确描述, 并在此基础上提出一种竞态集来存储和分析系统的竞态条件, 然后利用获取到的竞态集, 构建系统的竞态条件图来预测系统潜在的竞态同步关系. 与文献[15-17]中的图形相比, 本文的竞态条件图有如下几个优点: 1. 只包含与系统竞态条件直接相关的执行事件, 因此具有较小的空间开销; 2. 不仅可以描述实时嵌入式系统执行事件之间的发生先后关系和竞态同步关系, 而且可以用来预测系统潜在的竞态同步关系及其所带来的不同执行情况和结果; 3. 适用于各种类型的执行事件, 具有通用性.

1 执行序列与竞态条件的相关概念

实时嵌入式系统的竞态条件主要包括数据竞态

(data race)[1]、消息竞态(message race)[10,14]和信号量竞态(semaphore race)[7,11-12]等,其中数据竞态一般可以通过在系统中增加正确的同步机制来解决[1],因此,本文主要讨论实时嵌入式系统的消息竞态和信号量竞态.为此,本文首先根据实时嵌入式系统的执行特征,给出系统执行序列的数学模型定义,并以该数学模型定义的性质来描述实时嵌入式系统的竞态条件.

1.1　实时嵌入式系统的执行序列

实时嵌入式系统一般是多任务的并发系统,其系统内核通过各种调度算法[18](如基于优先级的可抢占式调度算法、基于时间轮换的调度算法等)实现任务管理和资源分配,而且系统任务之间通过信号量、消息队列等方式进行同步通信.因此,实时嵌入式系统的执行过程可以看作是各种通信同步执行事件的集合.下面先定义嵌入式系统的执行事件,再具体描述执行序列及其性质.

定义 **1.** 执行事件.在实时嵌入式系统中,任务间通信同步的执行事件 e 可表示为 $e=<y, p, t, i, b>$,其中:

1. y 表示事件类型,包括获取信号量(TS)、释放信号量(GS)、发送消息(SM)、接收消息(RM)等;
2. $p\in P$ 表示执行事件所属的任务;
3. t 表示执行事件的发生时间;
4. i 表示执行事件在任务 p 中的序号;
5. $b\in B$ 表示执行事件的操作对象,且有:若 $y=(TS\vee GS)$,则 b 表示信号量,用 $b=$ sem 表示;若 $y=(SM\vee RM)$,则 b 表示消息队列,用 $b=$ msq 表示.

设 e_i, e_j 为 2 个执行事件,若 $p_i!=p_j\wedge b_i=b_j=$ msq$\wedge y_i=$ SM$\wedge y_j=$ RM 且 e_j 收到 e_i 发送的消息,则称 e_i, e_j 共享一个消息队列 b_i,用符号$<e_i, e_j>$来表示.

定义 **2.** 执行序列.实时嵌入式系统的任意一次执行过程称为一个执行序列,可表示为 $Q=<P, I, B, E>$,其中:

1. $P=\{p_i\mid i=1, 2, \cdots, n\}$表示实时嵌入式系统的任务集合,其中 n 表示任务总数;
2. I 表示实时嵌入式系统的执行输入条件;
3. $B=\{b_i\mid i=1, 2, \cdots, m\}$表示实时嵌入系统中的操作对象集合(本文主要指信号量和消息队列),其中 m 表示操作对象总数;
4. $E=\{e_i\mid i=1, 2, \cdots, k\}$表示本次执行 Q 中所有执行事件的集合,其中 k 表示执行事件总数.

实时嵌入式系统的执行序列有如下 2 个性质:

性质 **1.** 在执行序列 Q 中,集合 E 的各个元素之间(即各个执行事件之间),满足一种发生先后关系(happened-before relation)[13].这种关系是一种偏序关系,可用符号"$\xrightarrow{hb}$"来表示.即设 $\forall e_i, e_j\in E$,则有:

1. 若 $p_i=p_j\wedge i<j$,则 $e_i\xrightarrow{hb}e_j$;
2. 若$<e_i, e_j>$成立,则 $e_i\xrightarrow{hb}e_j$;
3. 若 $e_i\xrightarrow{hb}e_j\wedge e_j\xrightarrow{hb}e_k$,则 $e_i\xrightarrow{hb}e_k$.

性质 **2.** 在执行序列 Q 中,集合 E 的各个元素(即各个执行事件)在发生时间上是一种全序关系.即设 $\forall e_i, e_j\in E$,若 $e_i\xrightarrow{hb}e_j$,则有 e_i 先于 e_j 发生,即 $t_i<t_j$.

1.2　实时嵌入式系统的竞态条件

基于执行事件和执行序列的相关定义与性质,可以具体来描述实时嵌入式系统的消息竞态和信号量竞态.

定义 **3.** 消息竞态.令 Q 和 Q' 是任一实时嵌入式系统的 2 个执行序列.若在 Q 中存在 2 个消息对$<e_1, e_2>$和$<e_3, e_4>$,而在 Q' 中消息对$<e_3, e_2>$成立,则称执行事件 e_1 和 e_3 之间是一个消息竞态.

如图 1 所示是一个消息竞态的实例.其中图 1(a)和图 1(b)分别用时空图展现了某个实时嵌入式系统的 2 个执行序列 Q 和 Q'.在 Q 中,Task2 先收到 Task1 发送的消息(即存在消息对$<e_1, e_2>$),后收到 Task3 发送的消息(即存在消息对$<e_3, e_4>$);而在 Q' 中,Task2 先收到了 Task3 发送的消息(即消息对$<e_3, e_2>$成立),后收到 Task1 发送的消息(即消息对$<e_1, e_4>$成立).因此发送消息事件 e_1 和 e_3 相对于接收消息事件 e_2 来说,是一个消息竞态.

定义 **4.** 信号量竞态.令 Q 和 Q' 是任一实时嵌入式系统的 2 个执行序列.若在 Q 中执行事件 e_1 先获取到某个信号量,e_2 后获取到相同的信号量,而在 Q' 中 e_2 先获取到该信号量,e_1 后获取到相同的信号量,则称执行事件 e_1 和 e_2 之间是一个信号量竞态.

如图 2 所示是一个信号量竞态的实例.其中图 2(a)和图 2(b)分别是某个实时嵌入式系统的 2 个执行序列 Q 和 Q',e_1, e_2, e_3 和 e_4 为执行事件,且这些事件的类型为 $y_1=y_2=$ TS$\wedge y_3=y_4=$ GS.由图 2 可知,在 Q 中,Task1 先获取到信号量 S,Task2 后获取到该信号量;而在 Q' 中,Task2 先获取到信号

量 S, Task1 后获取到该信号量. 因此执行事件 e_1 和 e_2 之间是一个信号量竞态.

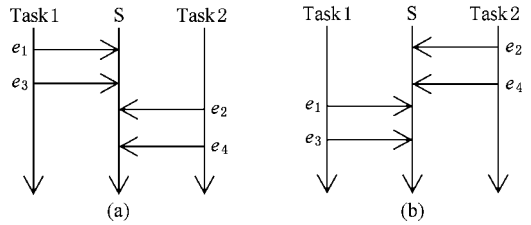


Fig. 2 An example of semaphore race. (a) Execution sequence Q and (b) Execution sequence Q'.

图 2 信号量竞态实例. (a) 执行序列 Q; (b) 执行序列 Q'

此外, 可用"$\xrightarrow{rd}$"表示执行事件之间的竞态同步关系, 即若执行事件 e_1 与 e_2 之间有竞态关系, 且在该次执行序列 Q 中, e_1 发生在 e_2 之前(即 $t_1 < t_2$), 则 $e_1 \xrightarrow{rd} e_2$. 因此, 竞态同步关系也是一种偏序关系. 由文献[3]可知存在竞态条件的 2 个执行事件 e_1 和 e_2 满足 $(9)(e_1 \xrightarrow{hb} e_2) \wedge (9)(e_2 \xrightarrow{hb} e_1)$ 为真, 即 e_1 和 e_2 是并行关系(也可表示为 $e_1 \parallel e_2$). 因此, 可推出如下结论:

结论 **1.** $\forall Q = <P, I, B, E> \wedge \{e_1, e_2, e_3, e_4\} \in E$, 若满足以下条件:

1. $b_1 = b_2 = b_3 = b_4 = msq$;
2. $y_1 = y_3 = SM \wedge p_1 != p_3$;
3. $y_2 = y_4 = RM \wedge p_2 = p_4$;
4. $<e_1, e_2> \wedge <e_3, e_4>$ 为真;
5. $e_2 \parallel e_3$;

则 e_1 和 e_3 之间是一个消息竞态, 且若 $t_1 < t_3$, 则 $e_1 \xrightarrow{rd} e_3$, 若 $t_3 < t_1$, 则 $e_3 \xrightarrow{rd} e_1$.

结论 **2.** $\forall Q = <P, I, B, E> \wedge \{e_1, e_2\} \in E$, 若满足以下条件:

1. $b_1 = b_2 = sem$;
2. $y_1 = y_2 = TS \wedge p_1 != p_2$;
3. $e_1 \parallel e_2$;

则 e_1 和 e_2 之间是一个信号量竞态, 且若 $t_1 < t_2$, 则 $e_1 \xrightarrow{rd} e_2$, 若 $t_2 < t_1$, 则 $e_2 \xrightarrow{rd} e_1$.

2 竞态条件分析方法

实时嵌入式系统通常包含许多竞态条件, 首先需要定义一种竞态集来存储和分析系统的竞态条件.

定义 **5.** 竞态集. 令 $Q = <P, I, B, E>$ 是实时嵌入式软件的执行序列, 则所有与竞态条件直接相关的执行事件及其时序关系所组成的集合, 称为竞态集, 可表示为 $R = <E_r, \xrightarrow{hb}, \xrightarrow{rd}>$, 其中:

1. $E_r \subseteq E$ 是与竞态条件直接相关的执行事件集合;
2. $\xrightarrow{hb}$ 是定义在 E_r 上的发生先后关系;
3. $\xrightarrow{rd}$ 是定义在 E_r 上的竞态同步关系.

不管是消息竞态还是信号量竞态, 它们都是存在于具有相同操作对象的执行事件之间. 因此, 在对系统执行序列 Q 进行竞态条件分析时, 可以根据系统中不同的操作对象, 对执行事件集合 E 进行归类划分. 这里把执行序列 Q 根据不同操作对象归类划分出来的子序列集合记为 $Q_c = \{q_k \mid k = 1, 2, \cdots, m\}$, m 为操作对象个数}, 其中, 子序列 $q_k \in Q_c$ 可表示为 $q_k = <b_k, E_k>$, $b_k \in B$ 为第 k 个操作对象, E_k 为针对该操作对象的所有执行事件的集合. 然后, 就可以根据 q_k 中每个执行事件的类型和同步关系来分析系统的消息竞态和信号量竞态, 具体如算法 1 所示. 根据定义 3 可知消息竞态发生在接收消息的时刻, 因此在分析消息竞态时, 算法只要在接收消息事件出现的时候作判断: 首先找到接收消息事件 e 所对应的发送消息事件 e_s, 然后在 e 所属任务中查找每个发生在 e 之前的接收消息事件 e', 如果 $e' \parallel e_s$, 那么 e_s 与 e' 所对应的发送消息事件 e'_s 之间是一个消息竞态, 且 $e'_s \xrightarrow{rd} e_s$; 根据定义 4 可知信号量竞态发生在获取信号量的时刻, 因此分析信号量竞态时, 算法只要在获取信号量事件出现的时候作判断: 在 q_k 中查找每个发生在 e 之前的获取信号量事件 e', 如果 $e' \parallel e$, 那么 e' 与 e 之间是一个信号量竞态, 且 $e' \xrightarrow{rd} e$.

算法 **1.** 竞态条件分析算法.

输入: $Q = <P, I, B, E>$;

输出: $R = <E_r, \xrightarrow{hb}, \xrightarrow{rd}>$

Process:

① Let $R = \{\}$ be an empty race set;
② Get $Q_c = \{q_k \mid k = 1, 2, \cdots, m\}$ from Q;
③ For $q_k = <b_k, E_k> \in Q_c$
④ For $e_i \in E_k$
⑤ If $y_i == RM$ then
⑥ e_s = the sending event of e_i;
⑦ For prior RM event e' in the same task
⑧ If $e' \parallel e_s$ then
⑨ e'_s = the sending event of e';
⑩ Add $e'_s \xrightarrow{rd} e_s$ into R;
⑪ End if

⑫　　End for
⑬　End if
⑭　If y_i = = TS *then*
⑮　For prior TS event e'
⑯　If $e' \| e_i$ then
⑰　Add $e' \xrightarrow{rd} e_i$ into R;
⑱　End if
⑲　End for
⑳　End if
㉑ End for
㉒End for
End Process.

我们对算法 1 所示的竞态条件分析算法进行时间复杂度分析. 在该算法中, 语句③和语句④的 2 个"For"循环遍历了执行序列 Q 中的每个执行事件. 对于每个接收消息事件, 语句⑦的"For"循环遍历了每个发生在该事件之前且与该事件具有相同任务属性和相同操作对象的接收消息事件; 而对于每个获取信号量事件, 语句⑮的"For"循环则遍历了每个发生在该事件之前且获取相同信号量的执行事件. 设 C_i 表示算法中语句 i 的复杂度, 则语句⑥和语句⑨的时间复杂度相同, 且为 $C_6 = C_9 = O(|E_k|)$, 其中 $|E_k|$ 表示子序列 q_k 中执行事件的总数, 且 $|E_k| \leqslant |E|$. 为了将 $e'_s \xrightarrow{rd} e_s$ 加到竞态集 R 中, 语句⑩需要判断 R 中是否已经存在执行事件 e'_s 和 e_s, 如果它们已经存在, 则只要添加竞态同步关系即可, 因此语句⑩时间复杂度 $C_{10} = O(|E_r|)$, 其中 $|E_r|$ 表示竞态集 R 中执行事件的总数; 同理, $C_{17} = O(|E_r|)$. 因此, 从语句⑤到语句⑳的时间复杂度为 $O(|E_k|) \cdot O(|E_r|)$, 而整个算法的时间复杂度 C 为:

$$C = O\left(\sum_{k=1}^{m}(|E_k| \cdot |E_k| \cdot |E_r|)\right) = O(|E|^2 \cdot |E_r|).$$

在算法 1 中, 若执行序列 Q 中不含竞态条件, 则其时间复杂度变为 $C = O(|E|^2)$.

3　竞态条件图

根据分析获得的系统竞态集, 就可以构建系统的竞态条件图. 竞态条件图可以说是竞态集的可视化表示. 如: 令 $R = \langle E_r, \xrightarrow{hb}, \xrightarrow{rd}\rangle$ 是执行序列 Q 的竞态集, 则 R 可表示为 $G = \langle V, A\rangle$ 其中 V 表示节点集合, 其元素即某个节点 $v \in V$ 表示集合 E_r 中的每个执行事件; A 表示边的集合, 其元素 $a \in A$ 表示 2 个执行事件之间的时序关系, 即 $\xrightarrow{hb}$ 或 $\xrightarrow{rd}$, 分别用符号实箭头和虚箭头表示.

即如图 3 所示是一个竞态条件图实例 (其中 $e(p, n)$ 表示任务 p 中的第 n 个执行事件), 它可表示出以下信息:

$$e(x,i) \xrightarrow{hb} e(x,i+1);\ e(y,j) \xrightarrow{hb} e(y,j+1);$$
$$e(x,i) \xrightarrow{rd} e(y,j+1);\ e(x,i+1) \xrightarrow{rd} e(y,j).$$

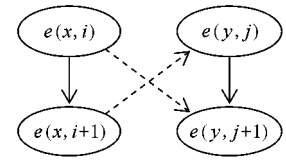


Fig. 3　Race condition graph.
图 3　竞态条件图实例

利用竞态条件图来描述竞态集可以更有效地分析和预测系统潜在的竞态同步关系. 设由执行序列 $Q = \langle P, I, B, E\rangle$ 得到的竞态集 $R = \langle E_r, \xrightarrow{hb}, \xrightarrow{rd}\rangle$, 其竞态条件图为 $G = \langle V, A\rangle$ 若 $\forall e_i, e_j \in E_r \wedge e_i \xrightarrow{rd} e_j$, 则可能存在 $Q' = \langle P, I, B, E'\rangle$ 使得 $e_j \xrightarrow{rd} e_i$ 成立, 即在竞态条件图 G 上表现为竞态同步关系的边其方向发生改变, 从而得到其他形式的竞态条件图. 而这些其他形式的竞态条件图则反映了系统潜在的竞态同步关系. 如图 4 给出了图 3 中的竞态条件图的所有潜在竞态同步关系.

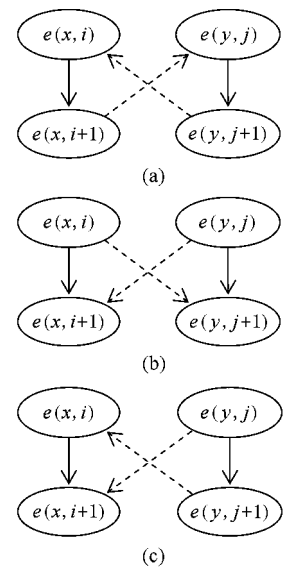


Fig. 4　Other changes of a race condition graph.
图 4　竞态条件图的不同变化形式

但是在实时嵌入式系统中, 并不是所有的可变化获得的竞态条件图都是可行的, 如图 4(a) 所示的

竞态条件图在实际运行中不可能发生. 这是由于该图中存在一个全局环, 即: $e(x,i) \rightarrow e(x,i+1) \rightarrow e(y,j) \rightarrow e(y,j+1) \rightarrow e(x,i)$. 即存在如下定理:

定理 **1.** 设 $G' = <V, A'>$为一个由 G 变化获得的竞态条件图, 若在 G' 中存在一个全局环, 则该竞态条件图在实际系统执行中将不可能发生.

证明. 可用反证法进行证明, 即: 假设 $G' = <V, A'>$在实际执行当中发生并且存在一个全局环, 不妨设该环为 $V_1 \rightarrow V_2 \rightarrow \cdots \rightarrow V_n \rightarrow V_1$. 根据竞态条件图的定义, 其相应的竞态集 $R' = <E_r, \xrightarrow{hb}, \xrightarrow{rd}>$中, 存在 n 个事件 $e_1, e_2, \cdots, e_n \in E_r$, 满足 $e_1 \rightarrow e_2 \rightarrow \cdots \rightarrow e_n \rightarrow e_1$(这里$\rightarrow$可以是$\xrightarrow{hb}$或$\xrightarrow{rd}$), 再根据发生先后关系与竞态同步关系的定义, 可知在该执行当中, 这 n 个事件发生时间之间的关系为 $t_1 < t_2 < \cdots < t_n < t_1$, 这显然矛盾. 因此 G' 在实际执行中将不可能发生. 证毕.

从竞态条件图的变化情况来看, 若一个竞态条件图 $G= <V, A>$中, 竞态同步关系的边的数量为 n, 则可变化的竞态条件图总数为 $2^n - 1$ 个, 即系统有 $2^n - 1$ 个可能的竞态同步关系. 根据上述定理, 可以剔除一些实际不可行的系统竞态同步关系, 因此, 在对系统进行测试时, 无需遍历所有竞态条件图的变化情况.

4 实验与性能分析

我们采用基于宿主机/目标机架构的实现方案(如图 5 所示)对上述的竞态条件分析方法进行编程实验. 其中目标机端由 Xilinx ML505 开发板以及 uC/OS II 实时嵌入式操作系统组成, 宿主机端由 PC 机(AMD64 位双核 CPU 和 1 GB 内存)和 Windows XP 操作系统组成. 目标机与宿主机之间采用以太网和串口进行通信. 当目标机端的实时嵌入式系统开始运行时, 通过追踪组件[13]获取系统的执行信息, 接着由追踪控制程序传回到宿主机端生成执行序列, 然后进行竞态条件分析并生成竞态条件图.

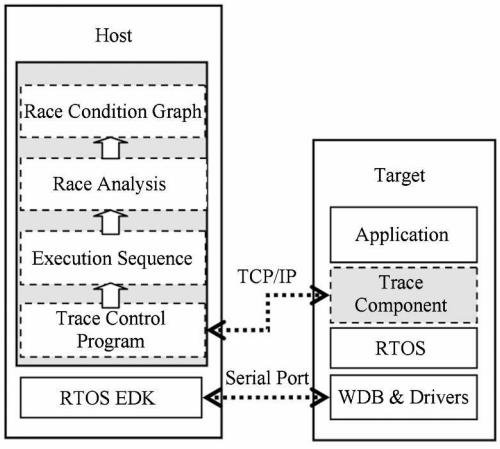


Fig. 5 Implementation scheme based on host/target architecture.

图 5 基于宿主机/目标机架构的实现方案

这里以“哲学家”问题[19]为例子来说明本方法的工作过程. 本例子共有 4 个哲学家, 对应于 4 个任务: T_1, T_2, T_3 和 T_4; 共有 4 根筷子, 对应于 4 个二进制信号量: S_1, S_2, S_3 和 S_4. 每位哲学家在拿每根筷子之前都有个随机的延时, 用完餐后先后把两根筷子放下. 设在某次系统执行当中, 4 个哲学家的用餐先后顺序为 T_4, T_1, T_2 和 T_3. 根据 4 个不同的信号量, 对执行序列进行划分得出 4 个子序列, 分别如图 6 所示, 其中 $TS(x, y)$ 表示第 x 个任务中序号为 y 的获取信号量事件, 而 $GS(x, y)$ 则表示第 x 个任务中序号为 y 的释放信号量事件.

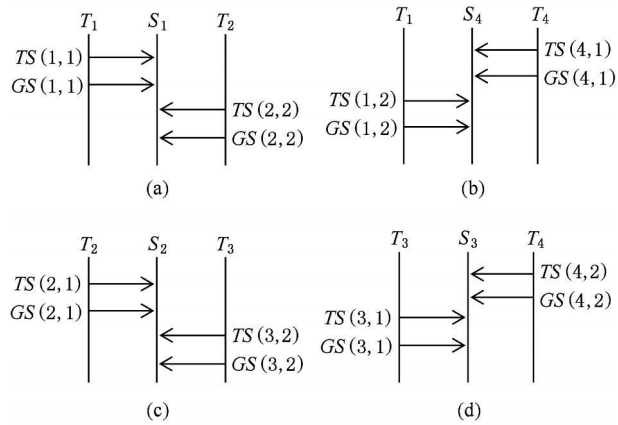


Fig. 6 Subr sequences of dining philosopher program.

图 6 “哲学家”例子的子序列图

采用第 2 节所描述的竞态条件分析算法对该执行序列进行分析, 获得系统的竞态集如下所示:

$$R = \{TS(1,1) \xrightarrow{hb} TS(1,2); TS(2,1) \xrightarrow{hb} TS(2,2);$$
$$TS(3,1) \xrightarrow{hb} TS(3,2); TS(4,1) \xrightarrow{hb} TS(4,2);$$
$$TS(1,1) \xrightarrow{rd} TS(2,2); TS(4,1) \xrightarrow{rd} TS(1,2);$$
$$TS(2,1) \xrightarrow{rd} TS(3,2); TS(4,2) \xrightarrow{rd} TS(3,1)\}.$$

根据得到的竞态集, 可构建出系统的竞态条件图, 如图 7 所示:

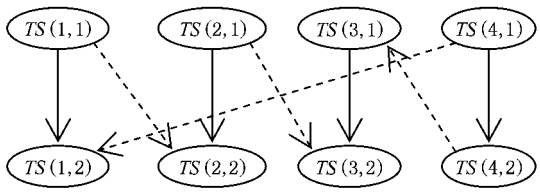


Fig. 7　Race condition graph of dining philosopher program.

图 7　“哲学家”例子的竞态条件图

在该例子中,可变化的竞态条件图总数有 $2^4-1=15$ 个(详见附录 A 所示),它们反映了系统所有可能的竞态同步关系. 其中图 A1(g)存在全局环,图 A1(h)会造成系统死锁,剩下的竞态同步关系都是可行的,因此该例子总共有 13 中潜在的竞态同步关系,而在这 13 种竞态同步关系下,程序执行可分别得到不同的结果,即可能造成 4 位哲学家用餐的先后顺序不同,如表 1 所示. 因此在对该例子进行测试的时候,需要充分考虑到各种可能的竞态同步关系所带来的不同的执行情况.

Table 1　Result of Potential Race Synchronization Relations of the Dining Philosopher Program

表 1　“哲学家”例子中所有潜在的竞态同步关系及其结果说明

No.	Feasibility	Ordering of Dining
1	✓	$T_4, \{T_3, T_1\}, T_2$
2	✓	$\{T_2, T_4\}, \{T_1, T_3\}$
3	✓	T_1, T_2, T_4, T_3
4	✓	T_4, T_3, T_2, T_1
5	✓	T_1, T_4, T_3, T_2
6	✓	T_2, T_1, T_4, T_3
7	×	Include global circle
8	×	Cause deadlock
9	✓	T_3, T_4, T_1, T_2
10	✓	T_2, T_3, T_4, T_1
11	✓	T_1, T_2, T_3, T_4
12	✓	T_3, T_4, T_2, T_1
13	✓	$\{T_1, T_3\}, \{T_2, T_4\}$
14	✓	$T_2, \{T_1, T_3\}, T_4$
15	✓	T_3, T_2, T_1, T_4

此外,我们还将包含生成竞态条件图的分析方法与未包含生成竞态条件图的分析方法在时间和空间开销方面进行比较,结果如图 8 所示. 虽然前者的时间和空间开销比后者的大,但是随着“哲学家”总数和竞态条件数量的增大,前者的时间和空间开销的增长率逐渐降低,如图 9 所示. 同时,我们还将本文的竞态条件图与文献[15]中所描述的竞态条件图形表示方法进行比较,结果如图 10 所示. 随着“哲学家”数量的增大,2 种图形的节点以及边的总数都逐渐上升(在本例子中,节点总数与边的总数相等),但

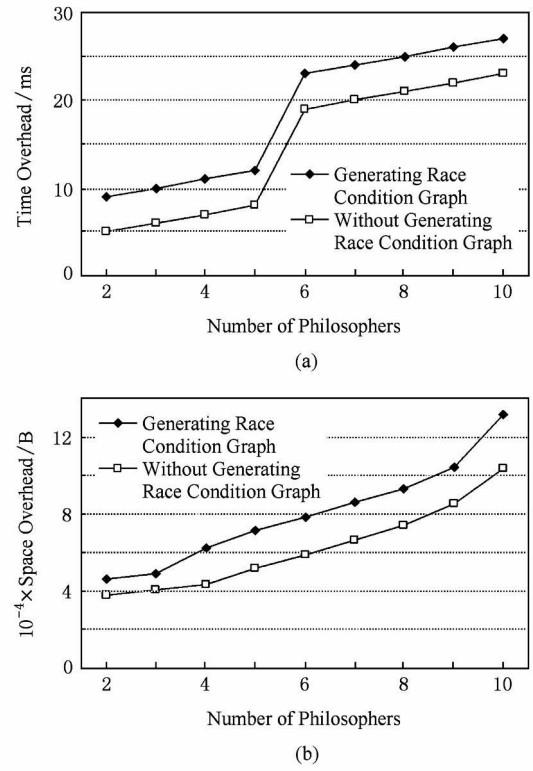


Fig. 8　Time and space overhead of the approach. (a) Time overhead and (b) Space overhead.

图 8　方法的时间和空间开销. (a) 时间开销; (b) 空间开销

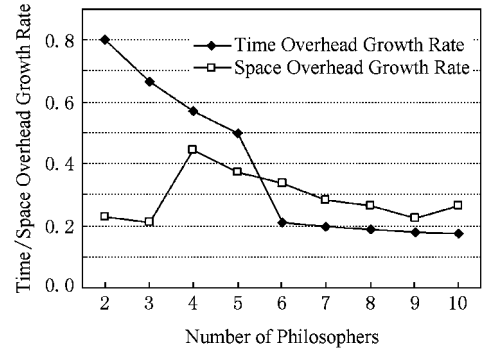


Fig. 9　Growth rate of time and space overhead of the analysis approach.

图 9　本分析方法时间和空间开销的增长率

是本文的竞态条件图明显比较节省空间开销, 这是由于文献[15]中的图形表示方法包含系统所有的执行事件及其相关信息, 而本文的竞态条件图则只需包含与竞态条件直接相关的执行事件及其同步关系信息.

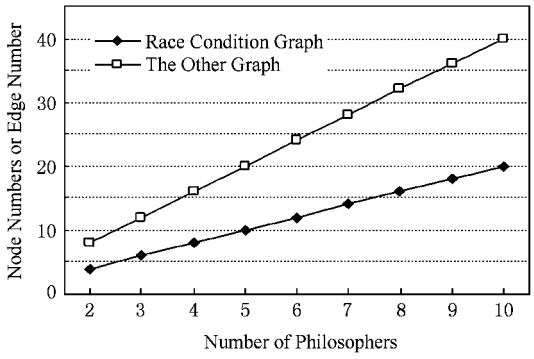


Fig. 10 Comparison of space overhead with the other approach.

图 10 两种方法空间开销的比较

5 结 论

本文所提出的竞态集和竞态条件图不仅适用于分析实时嵌入式系统的消息竞态和信号量竞态, 而且也同样适用于分析其他系统的竞态条件. 由于本文描述的竞态同步关系是一种偏序关系, 因此竞态集是一种特殊的偏序集, 而竞态条件图也是一种特殊的偏序图. 虽然本文所描述的动态方法不能保证分析出目标系统所有可能的竞态条件(因为分析所有可能的竞态条件是 NP-Hard 问题[11]), 但是可以根据所构建出的竞态条件图来预测和分析系统潜在的竞态同步关系, 从而可以有效地分析系统可能存在的其他执行情况, 协助用户对系统进行测试.

参 考 文 献

[1] Netzer R H B, Miller B P. What are race conditions? Some issues and formalizations [J]. ACM Letters on Programming Languages and Systems, 1992, 1(1): 74-88

[2] Keralapura R, Chuah C N. Race conditions in coexisting overlay networks [J]. IEEE/ACM Trans on Networking, 2008, 16(1): 1-14

[3] Engler D, Ashcraft K. RacerX: Effective, static detection of race conditions and deadlocks [C] //Proc of the 19th ACM Symp on Operating Systems Principles (SOSP). New York: ACM, 2003: 237-252

[4] Boyapati C, Lee R, Rinard M. Ownership types for safe programming: preventing data races and deadlocks [C] // Proc of the 17th ACM OOPSLA'02. New York: ACM, 2002: 211-230

[5] Blanc N, Kroening D. Race analysis for SystemC using model checking [C] //Proc of IEEE/ACM Int Conf on Computer-Aided Design. New York: ACM, 2008: 356-363

[6] Yu Y, Rodeheffer T, Chen W. RaceTrack: Efficient detection of data race conditions via adaptive tracking [C] // Proc of the 12th ACM Symp on Operating Systems Principles. New York: ACM, 2005: 221-234

[7] Klein P N, Lu H I, Netzer R H B. Detecting race conditions in parallel programs that use semaphores [J]. Algorithmic, 2003, 35(4): 321-345

[8] Pozniansky E, Schuster A. Efficient on-the-fly data race detection in multithreaded C++ programs [C] //Proc of PPoPP'03. New York: ACM, 2003: 179-190

[9] Park M Y, Chung S H. Detection of first races for debugging message-passing programs [C] //Proc of the 8th IEEE Int Conf on Computer and Information Technology. Piscataway, NJ: IEEE, 2008: 261-266

[10] Tai K C. Race analysis of traces of asynchronous message-passing programs [C] //Proc of ICDCS'97. Piscataway, NJ: IEEE, 1997: 261-268

[11] Lei Y, Carver R H. Reachability testing of concurrent programs [J]. IEEE Trans on Software Engineering, 2006, 32(6): 382-403

[12] Chen Y, Guo D H. Race condition analysis and testing of concurrent programs [J]. Int Journal of Information and Electronics, 2008, 1(1): 12-18

[13] Lamport L. Time, clocks, and the ordering of events in a distributed system [J]. Communications of the ACM, 1978, 21(7): 558-565

[14] Park M Y, Hai N C T, Jun Y K, et al. Visualization of message races in MPI parallel programs [C] // Proc of the 7th IEEE Int Conf on Computer and Information Technology. Piscataway, NJ: IEEE, 2007: 316-321

[15] Hwang G H, Chang S J, Chu H D. Technology for testing nondeterministic client/server database applications [J]. IEEE Trans on Software Engineering, 2004, 30(1): 59-77

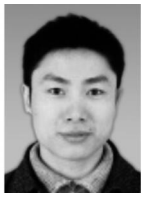
[16] Schaeli B, Gerlach S, Hersch R D. Decomposing partial order execution graphs to improve message race detection [C] // Proc of IEEE Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2007: 1-8

[17] Mitchell B. Resolving race conditions in asynchronous partial order scenarios [J]. IEEE Trans on Software Engineering, 2005, 31(9): 767-784

[18] Li Renfa, Liu Yan, Xu Cheng. A survey of task scheduling research progress on multiprocessor system-on-chip [J]. Journal of Computer Research and Development, 2008, 45(9): 1620-1629 (in Chinese)

(李仁发, 刘彦, 徐成. 多处理器片上系统任务调度研究进展评述[J]. 计算机研究与发展, 2008, 45(9): 1620- 1629)

[19] Chen Y, Lee Y H, Wong W E, et al. A race condition graph for concurrent program behavior [C] // Proc of the 3rd Int Conf on Intelligent System and Knowledge Engineering. Piscataway, NJ: IEEE, 2008: 662- 667

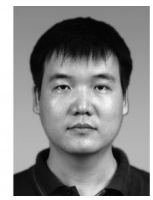


Chen Yan, born in 1981. PhD candidate of the School of Information Science and Technology, Xiamen University. Student member of China Computer Federation. His main research interests include real-time system analysis and testing.

陈　艳, 1981 年生, 博士研究生, 中国计算机学会学生会员, 主要研究方向为实时嵌入式软件的分析与测试.

Xu Xiaofeng, born in 1983. PhD candidate of the Physics Department, Xiamen University. Student member of China Computer Federation. His main research interests include software debugging and testing.

徐晓峰, 1983 年生, 博士研究生, 中国计算机学会学生会员, 主要研究方向为软件测试.



Li Xiaochao, born in 1970. Received his PhD degree from the Physics Department of Xiamen University in 2004. He is now an associate professor in the Electronic Engineering Department of Xiamen University. His main research interests include embedded systems, integrated circuit and multimedia.

李晓潮, 1970 年生, 博士, 副教授, 主要研究方向为嵌入式系统、集成电路、扩谱通信和多媒体.

Guo Donghui, born in 1967. Received his PhD degree in semiconductor device & device physics from the Physics Department of Xiamen University in 1994. He is now professor and PhD supervisor of the School of Information Science & Technology of Xiamen University. He is also a member of China Computer Federation. Recently his main research interests include artificial intelligence, network communication, and software-hardware co-design and so on.

郭东辉, 1967 年生, 1994 年获厦门大学半导体物理与器件物理专业博士学位, 厦门大学教授, 博士生导师, 中国计算机学会会员, 主要研究方向为人工智能、网络通信、软硬件协同设计等.

Research Background

The non-determinacy of task scheduling and the uncertainty of external environment cause race conditions in real-time embedded systems. Race conditions make synchronization order of operations to the shared variables or shared resource during multiple tasks be different from one execution to another. This may cause abnormal behaviors of systems, such as deadlock, timing errors, etc. Many static and dynamic approaches have been proposed to analyze and detect race conditions in parallel and concurrent programs. These approaches usually use space-time diagrams to describe system executions and analyze race conditions. However, space-time diagram is not formal enough to store, visualize and compute race conditions. In this paper, a kind of race set and race condition graph are presented to assist users in analyzing race conditions in real-time embedded systems. The race set includes the information of happened-before relations and race synchronization relations among the operation events which have races. The race condition graph is constructed to visualize the race conditions. It is also used to predict the potential race synchronization relations of systems. The approach proposed can be used to predict the potential execution situations and results of real-time embedded systems and other kinds of concurrent programs efficiently. Our work is supported by the National Natural Science Foundation of China (No. 60753001) and the Financial Support Program from the Chinese Ministry of Education for New Century Talented Persons.

附录 A “哲学家”例子竞态条件图的所有变化情况

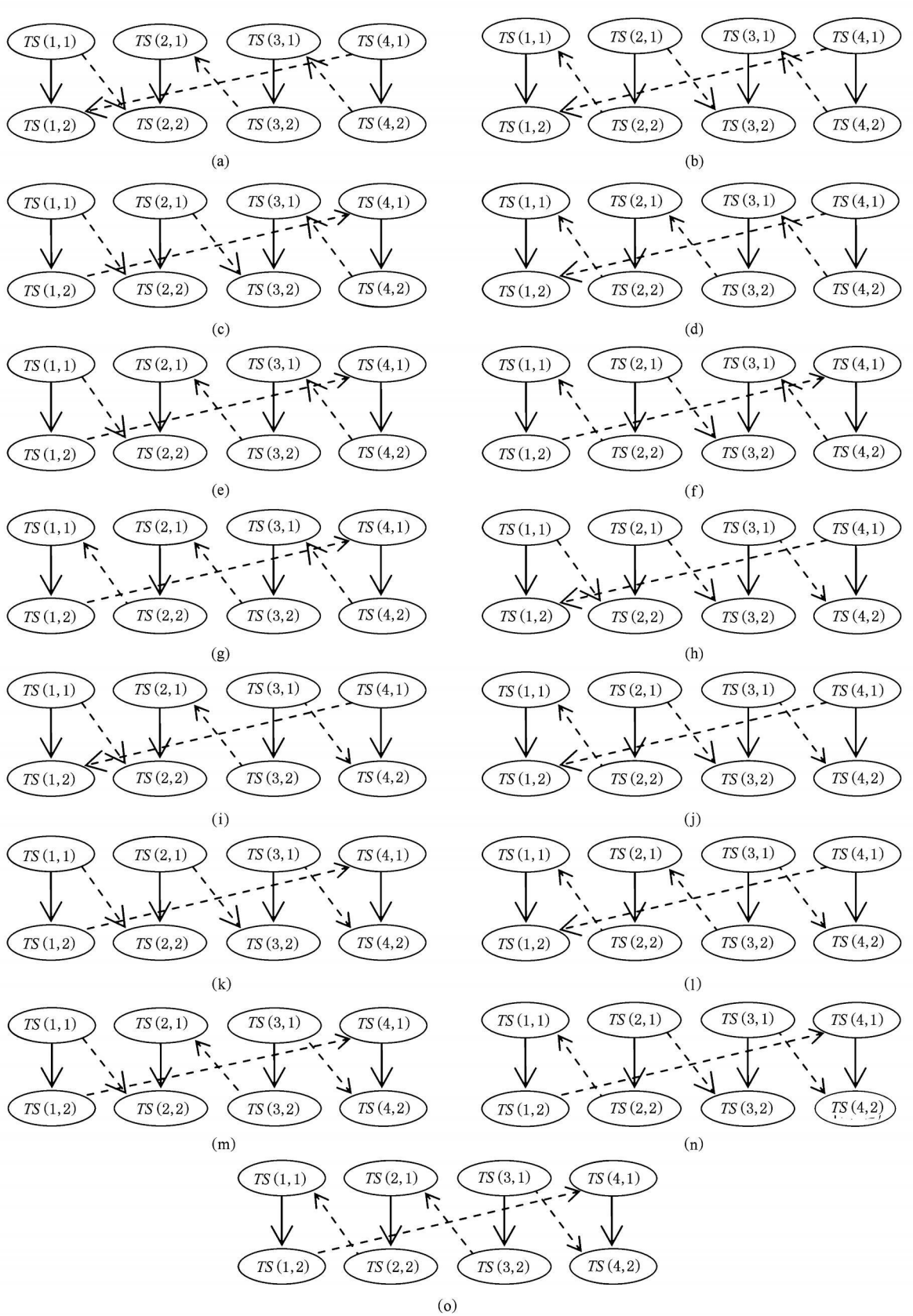


Fig. A1 Race condition graphs of dining philosopher program.

图 A1 “哲学家”例子竞态条件图的所有变化情况