(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 111611029 A
(43)申请公布日 2020.09.01

(21)申请号 201910136529.9

(22)申请日 2019.02.25

(71)申请人 中国科学院沈阳自动化研究所
地址 110016 辽宁省沈阳市沈河区南塔街114号

(72)发明人 曾鹏 万广喜 宋纯贺 于海斌

(74)专利代理机构 沈阳科苑专利商标代理有限公司 21002
代理人 许宗富

(51)Int.Cl.
G06F 9/448(2018.01)
G05B 19/418(2006.01)

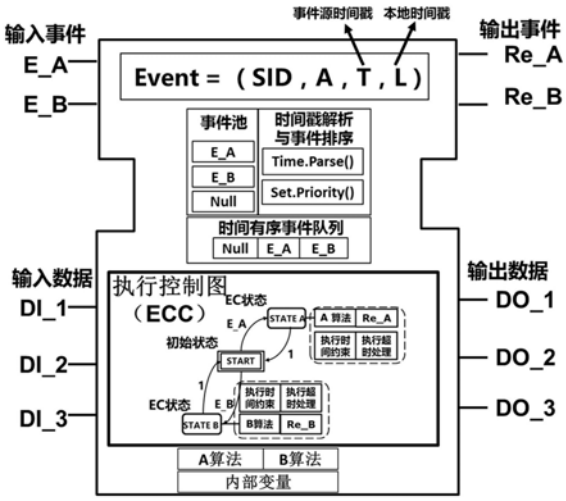
权利要求书1页 说明书4页 附图3页

(54)发明名称

一种计算时序确定的IEC 61499功能块建模方法

(57)摘要

本发明涉及一种保证计算时序确定性的IEC 61499功能块建模方法,包括:重新定义功能块的事件接口,用于将事件定义为包含时间戳的事件结构体;增加功能块对到达事件的时序管理模块,用于解析到达的本地事件,将事件按照时序排列,保证事件队列中事件按照确定的时序被响应;增加算法执行管理模块,用于定义算法执行时间约束、超时响应处理算法接口,对ECC执行响应算法进行管理、监测程序执行状态。本发明通过对IEC 61499基础功能块的扩展保证分布式系统协同计算的确定性,该功能块结构可以在异步事件高并发的分布式系统中保证事件执行时序的一致性和事件响应算法执行的确定性。



CN 111611029 A

1.一种计算时序确定的IEC 61499功能块建模方法,其特征在于包括:

定义功能块的事件接口,将事件定义为包含时间戳的事件结构体;

增加功能块对到达事件的时序管理模块,包括解析到达本地功能块的事件,将事件按照产生的时序排列,使得事件队列中的事件按照时序被响应;

增加算法执行管理模块,包括定义算法执行时间约束、超时响应处理算法接口,对执行控制图ECC执行响应算法进行管理、监测程序执行状态。

2.按照权利要求1所述的一种保证计算时序确定性的IEC 61499功能块建模方法,其特征在于,所述事件结构体包括事件的标识、事件的状态、事件的产生时间和事件到达本地功能块的时间。

3.按照权利要求1或2所述的一种保证计算时序确定性的IEC 61499功能块建模方法,其特征在于,所述解析到达本地功能块的事件,将事件按照产生的时序排列,包括:

将事件暂存到事件池中;

从本地功能块接收到第一个事件起,等待最大估计网络时延δt;

对事件池中的事件进行解析,获取时间戳;

根据时间戳设置事件优先级,生成时间有序事件队列,时间戳越早的事件执行优先级越高。

4.按照权利要求3所述的一种保证计算时序确定性的IEC 61499功能块建模方法,其特征在于,所述最大估计网络时延$\delta t \geq \max$(事件E_A到达本地功能块的时间-事件E_A的产生时间,事件E_B到达本地功能块的时间-事件E_B的产生时间)。

5.按照权利要求3所述的一种保证计算时序确定性的IEC 61499功能块建模方法,其特征在于,所述对事件池中的事件进行解析,获取时间戳是利用Time.Pase()函数实现的。

6.按照权利要求3所述的一种保证计算时序确定性的IEC 61499功能块建模方法,其特征在于,所述根据时间戳设置事件优先级是利用Set.Priority()函数实现的。

7.按照权利要求1所述的一种保证计算时序确定性的IEC 61499功能块建模方法,其特征在于,所述算法执行时间约束为用户预设的程序最坏情况执行时间t。

8.按照权利要求1或7所述的一种保证计算时序确定性的IEC 61499功能块建模方法,其特征在于,所述超时响应处理算法接口用于当程序执行超时时,输出错误信息事件给用户。

9.按照权利要求7所述的一种保证计算时序确定性的IEC 61499功能块建模方法,其特征在于,所述对ECC执行响应算法进行管理、监测程序执行状态,包括:

当响应事件时,算法执行管理线程开启,保存程序开始执行时的时间t_0;

在程序运行过程中,周期性的获取当前时间t_τ,直到程序执行结束,同时判断$t_\tau - t_0 < t$是否成立,如果成立则继续执行程序,否则触发超时响应处理算法接口,输出错误信息事件。

一种计算时序确定的IEC 61499功能块建模方法

技术领域

[0001] 本发明涉及工业自动化和计算机控制技术领域,特别涉及工业分布式系统编程标准IEC 61499以及功能块编程语言,具体的说是一种计算时序确定的IEC 61499功能块建模方法。

背景技术

[0002] 随着柔性制造的发展,通过各制造单元的协同计算实现制造系统对动态需求的响应已经成为趋势。IEC 61499标准针对工业过程测量和控制系统(IPMCS)提出以功能块为核心内容,强调通过功能块组合完成协同操作的分布式软件编程方法。IEC 61499功能块通过事件驱动的方式,并利用执行控制图(ECC)响应系统中的各类事件。分布式系统运行过程中,大量设备上计算任务的高并发导致同一时隙上可能会存在多个事件,而这些事件在时序上有先后次序,由于分布式系统中时钟不同步加上网络时延的动态变化导致执行次序并不是按照任务产生的时序执行,从而产生系统的不确定性。另外,对事件的响应程序的执行过程中可能会出现错误,程序的执行陷入无限循环或者未知状态,导致无输出结果或者错误的结果等不确定性情况的发生。IEC 61499功能块仅强调事件驱动的快速响应和异步灵活的优点,对事件时序的正确性响应以及响应算法的确定性执行缺少严格的定义,难以保证分布式系统协同计算的确定性。

发明内容

[0003] 针对上述功能块的不足,本发明的目的是通过对IEC 61499基础功能块的扩展保证分布式系统协同计算的确定性。该功能块结构可以在异步事件高并发的分布式系统中保证事件执行时序的一致性和事件响应算法执行的确定性。

[0004] IEC61499功能块分为基础功能块、复合功能块和服务接口功能块三种,本发明主要针对基础功能块进行扩展。本发明提出的一种保证分布式协同计算确定性的IEC 61499基础功能块结构通过下述技术方案实现。

[0005] 一种计算时序确定的IEC 61499功能块建模方法,包括:

[0006] 定义功能块的事件接口,将事件定义为包含时间戳的事件结构体;

[0007] 增加功能块对到达事件的时序管理模块,包括解析到达本地功能块的事件,将事件按照产生的时序排列,使得事件队列中的事件按照时序被响应;

[0008] 增加算法执行管理模块,包括定义算法执行时间约束、超时响应处理算法接口,对执行控制图ECC执行响应算法进行管理、监测程序执行状态。

[0009] 所述事件结构体包括事件的标识、事件的状态、事件的产生时间和事件到达本地功能块的时间。

[0010] 所述解析到达本地功能块的事件,将事件按照产生的时序排列,包括:

[0011] 将事件暂存到事件池中;

[0012] 从本地功能块接收到第一个事件起,等待最大估计网络时延δt;

[0013] 对事件池中的事件进行解析，获取时间戳；

[0014] 根据时间戳设置事件优先级，生成时间有序事件队列，时间戳越早的事件执行优先级越高。

[0015] 所述根据时间戳设置事件优先级包括按照事件的产生时间设置优先级，若事件的产生时间相同则按照事件到达本地功能块的时间设置优先级。

[0016] 所述最大估计网络时延$\delta t \geq \max$(事件E_A到达本地功能块的时间-事件E_A的产生时间，事件E_B到达本地功能块的时间-事件E_B的产生时间)。

[0017] 所述对事件池中的事件进行解析，获取时间戳是利用Time.Pase()函数实现的。

[0018] 所述根据时间戳设置事件优先级是利用Set.Priority()函数实现的。

[0019] 所述算法执行时间约束为用户预设的程序最坏情况执行时间t。

[0020] 所述超时响应处理算法接口用于当程序执行超时时，输出错误信息事件给用户。

[0021] 所述对ECC执行响应算法进行管理、监测程序执行状态，包括：

[0022] 当响应事件时，算法执行管理线程开启，保存程序开始执行时的时间t_0；

[0023] 在程序运行过程中，周期性的获取当前时间t_τ，直到程序执行结束，同时判断$t_\tau - t_0 < t$是否成立，如果成立则继续执行程序，否则触发超时响应处理算法接口，输出错误信息事件。

[0024] 本发明具有以下有益效果及优点：

[0025] 解决了由于分布式系统计算任务高并发带来的系统在事件响应顺序和响应时间不确定问题，进一步丰富了IEC61499功能块事件驱动执行的机制，推动IEC61499功能块在分布式系统中的进一步应用。

附图说明

[0026] 图1为IEC61499基本功能块模型；

[0027] 图2为IEC61499扩展功能块模型；

[0028] 图3为实施例功能块网络；

[0029] 图4为ECC扩展算法执行管理模块。

具体实施方式

[0030] 下面结合附图及实施例对本发明做进一步的详细说明。

[0031] 本发明提出一种保证分布式协同计算时序确定的功能块结构的应用场景如图3所示。功能块FB1，FB2和FB3是运行在分布式系统中不同的计算平台上。功能块FB1和FB2分别用于产生事件FB1.E_O和FB2.E_O，并输出数据FB1.D_O和FB2.D_O。功能块FB3实现简单的算术运算，当响应E_A事件时，计算FB3.D_O＝FB3.D_I1+FB3.D_I2，并输出事件CNF；当响应E_B事件时，计算FB3.D_O＝FB3.D_I1-FB3.D_I2并输出事件CNF。

[0032] 假设系统期望先获取FB1.D_O+FB2.D_O的值，后获取FB1.D_O-FB2.D_O的值，即表示为(FB1+FB2,FB1-FB2).在没有对IEC61499功能块模型扩展之前，由于网路时延和时间不同步的原因，FB3先接收到E_B事件后接收到E_A事件，导致最终执行结果为(FB1-FB2,FB1+FB2)。显然(FB1+FB2,FB1-FB2)≠(FB1-FB2,FB1+FB2)。即事件的响应时序导致响应执行结果的不确定性。

[0033] FB3响应E_A,执行加法程序,但是程序执行可能会出现错误,加法算法的执行陷入无限循环或者未知状态,导致无输出结果或者错误的结果等不确定性情况的发生。

[0034] 为了保证上述问题的确定性,本发明对IEC61499功能块模型进行扩展,具体模型如图2所示。功能块的外部接口保持不变,如图1所示。主要接口包括:事件输入接口用于接收外部事件;数据输入接口用于接收外部数据;事件输出接口用于输出内部产生的事件;数据输出接口用于输出内部产生的数据。

[0035] 扩展部分包括三个部分:

[0036] 1.事件定义的扩展。

[0037] 将事件定义为包含时间戳的事件结构体,具体实施为:

```
struct  EVENT  //事件结构体
{
    int      SID;              //定义事件标识
bool    ActiveState;      //定义事件激活状态
TIME    Time _born;       //定义事件产生时间戳
TIME    LocalTime _arrive;//定义事件到达时间戳
}
```

[0039] SID是整型变量,表示事件的ID号,对事件进行标识;

[0040] ActiveState是布尔变量,表示事件的状态,0表示事件处于沉默状态,1表示事件处于激活状态;

[0041] Time_born是时间变量,表示功能块外部事件产生时的时间;

[0042] LocalTime_arrive是时间变量,表示功能块接收到外部事件时的时间;

[0043] 实施例中,FB1,FB2,FB3的时间戳分别为:

[0044] FB1.E_O.Time_born=FB1.systemclock();//FB1通过调用系统时间,获取事件产生时间戳;

[0045] FB2.E_O.Time_born=FB2.systemclock();//FB2通过调用系统时间,获取事件产生时间戳;

[0046] FB3.CNF.Time_born=FB3.systemclock();//FB3通过调用系统时间,获取事件产生时间戳;

[0047] 当事件FB1.E_O到达FB3时,

[0048] FB3.E_A.LocalTime_arrive=FB3.Systemclock();//FB3通过调用系统时间,获取时间到达时间戳;

[0049] FB3.E_A.Time_born=FB1.E_O.Time_born;//FB3通过FB1.E_O事件,获取产生时间戳;

[0050] 同理,当事件FB2.E_O到达时

[0051] FB3.E_B.LocalTime_arrive=FB3.Systemclock();//FB3通过调用系统时间,获取时间到达时间戳;

[0052] FB3.E_B.Time_born＝FB2.E_0.Time_born。//FB3通过FB2.E_0事件，获取产生时间戳；

[0053] 通过定义包含时间戳的事件结构体，可以明确确定事件发生的时间。

[0054] 2.增加事件队列管理。

[0055] 功能块在接收到事件之后，并不立刻执行而是将事件暂存到事件池中。从接收到第一个事件起，等待最大估计网络时延δt之后，对事件池中的事件通过Time.Pase()函数进行时间戳解析，并利用Set.Priority()函数设置事件优先级时间戳越早的事件执行优先级越高。

[0056] 其中，δt是网络时延最大估计。

[0057] δt≥max(FB3.E_A.LocalTime_arrive-FB3.E_A.Time_born,FB3.E_B.LocalTime_arrive-FB3.E_B.Time_born)

[0058] 同时要求功能块运行系统平台之间运行时间同步协议，保证时钟同步，事件队列管理模块才能正确工作。

[0059] 系统期望结果(FB1+FB2,FB1-FB2)，则FB1.E_0.Time_born<FB2.E_0.Time_born。FB3对事件响应的顺序根据事件发生的时序响应，响应间隔δt能够保证不会丢失事件，因此最终响应结果是确定的系统期望结果。

[0060] 3.增加算法执行管理模块。

[0061] 功能块控制框图在响应执行上述事件队列中的事件的同时会开启一个对该响应算法执行的监测线程，实时监测程序的执行状态，当响应程序执行超时时会产生相应的超时处理事件。例如图4所示，用户可以显示的定义加法ADD的程序最坏情况执行时间(Worst-Case Execution Time)t和超时处理(输出Error事件)。

[0062] FB3响应E_A，执行加法程序的同时，算法执行管理线程开启，并保存程序开始执行时的时间t_0＝ADD.start()，在程序运行过程中周期性的获取当前时间t_τ＝ADD.run()，直到程序执行结束，同时判断$t_\tau-t_0<t$是否成立，如果成立则加法程序继续执行，否则触发超时处理算法，输出错误信息。因此保证了程序执行的确定性。

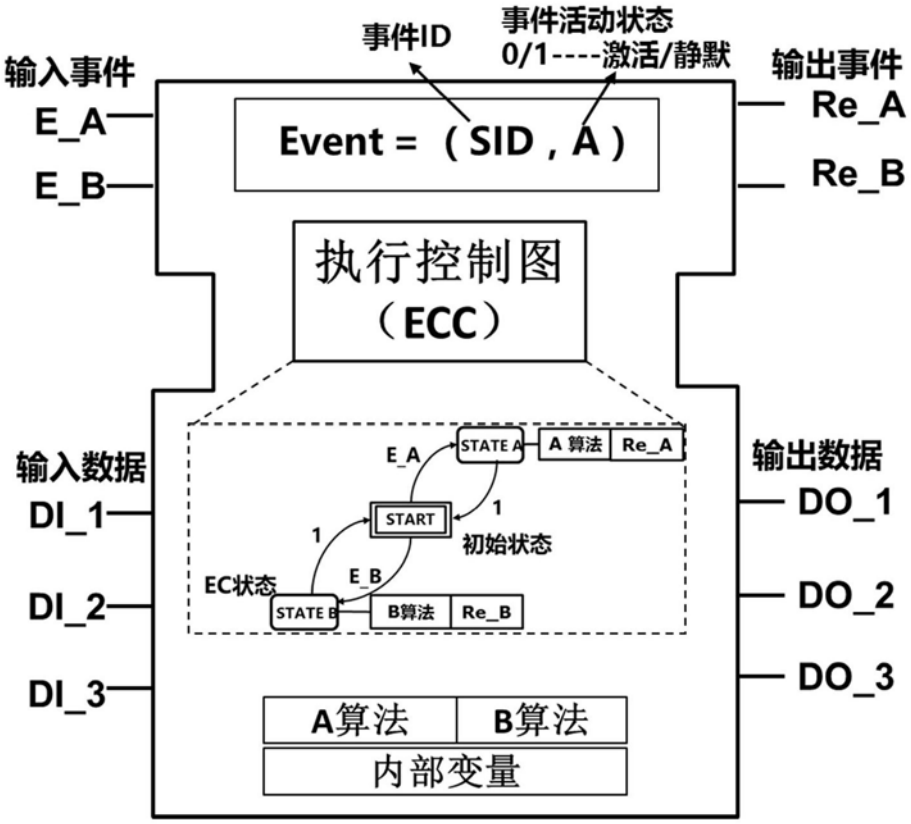

事件ID
事件活动状态
0/1----激活/静默
输入事件
E_A
E_B
Event = （SID，A）
输出事件
Re_A
Re_B
执行控制图
（ECC）
输入数据
DI_1
DI_2
DI_3
E_A
STATE A
A 算法
Re_A
START
1
1
初始状态
E_B
EC状态
STATE B
B算法
Re_B
输出数据
DO_1
DO_2
DO_3
A算法
B算法
内部变量

图1

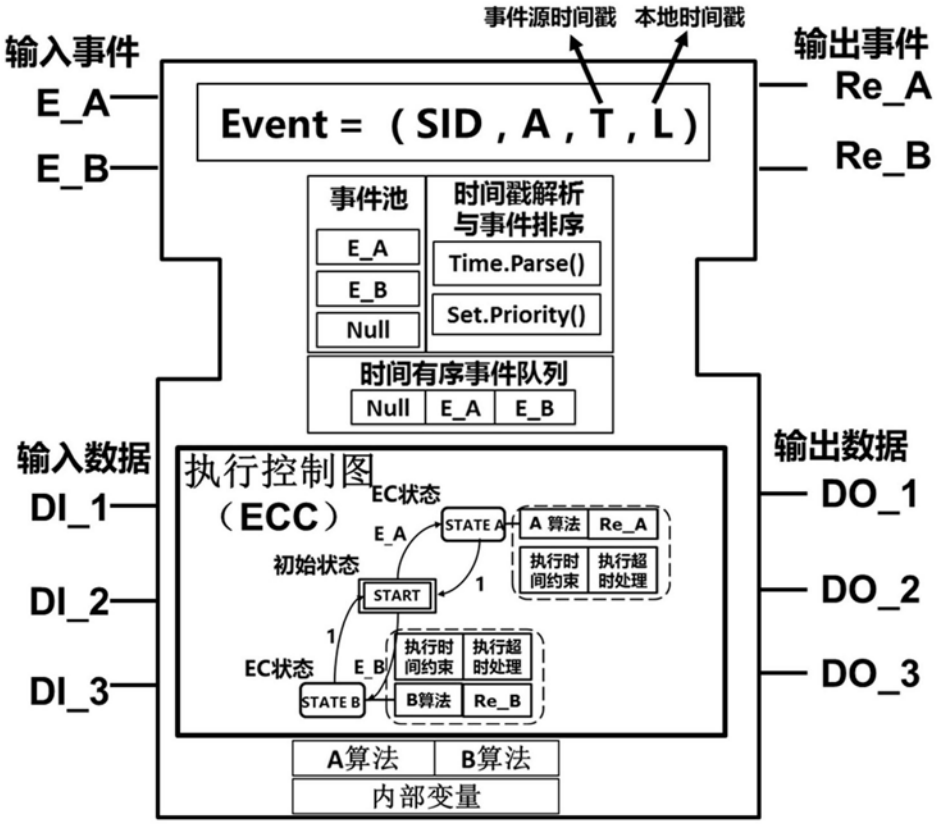

事件源时间戳
本地时间戳
输入事件
E_A
E_B
Event = （SID，A，T，L）
输出事件
Re_A
Re_B
事件池
E_A
E_B
Null
时间戳解析与事件排序
Time.Parse()
Set.Priority()
时间有序事件队列
Null
E_A
E_B
输入数据
DI_1
DI_2
DI_3
执行控制图（ECC）
EC状态
E_A
STATE A
A 算法
Re_A
执行时间约束
执行超时处理
初始状态
START
1
1
EC状态
E_B
STATE B
B算法
Re_B
输出数据
DO_1
DO_2
DO_3
A算法
B算法
内部变量

图2

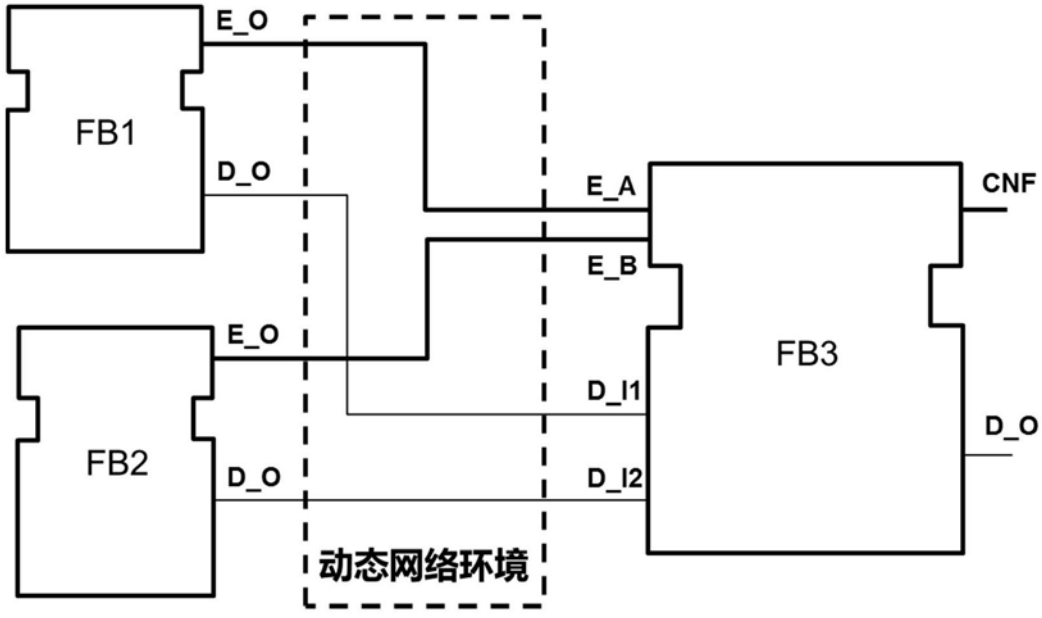

FB1
E_O
D_O
FB2
E_O
D_O
E_A
E_B
D_I1
D_I2
FB3
CNF
D_O
动态网络环境

图3

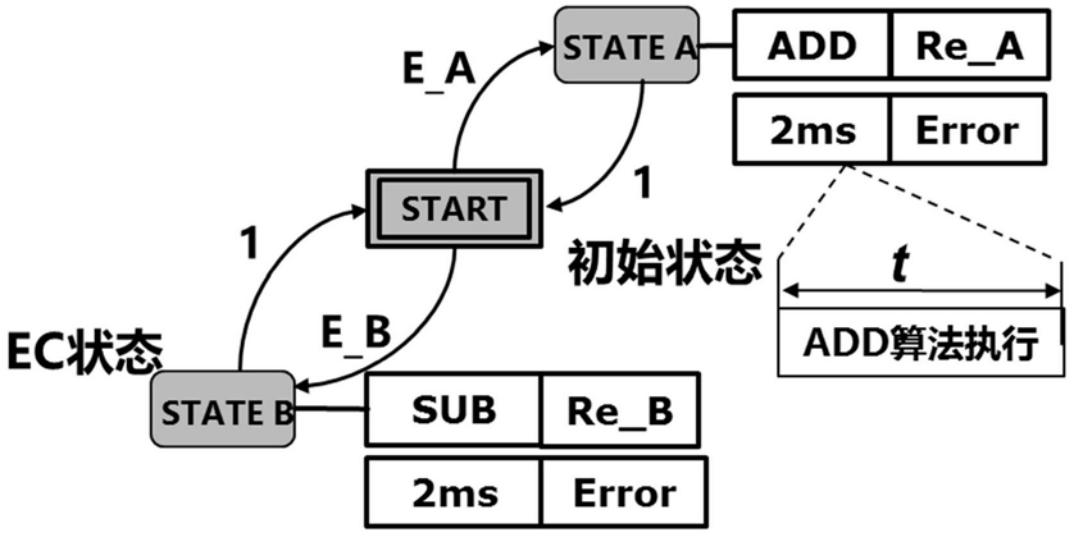

E_A
STATE A
ADD
Re_A
2ms
Error
START
1
1
初始状态
t
ADD算法执行
EC状态
E_B
STATE B
SUB
Re_B
2ms
Error

图4