北京大学学报(自然科学版) 第 55 卷 第 2 期 2019 年 3 月
Acta Scientiarum Naturalium Universitatis Pekinensis, Vol. 55, No. 2 (Mar. 2019)
doi: 10.13209/j.0479-8023.2019.005

实时嵌入式软件时间抽象状态机的扩展

单锦辉[1] 张路[2,†] 王金波[3] 张涛[3]

1. 华为技术有限公司, 深圳 518129; 2. 北京大学信息科学技术学院, 北京 100871; 3. 中国科学院空间应用工程与技术中心, 北京 100094; † 通信作者, E-mail: zhanglu@sei.pku.edu.cn

摘要 针对时间抽象状态机(TASM)存在的不足, 对TASM进行扩展, 增加数组数据类型、while循环处理规则以及“%”, “&”, “|”, “^”, “>>”和“<<”等运算符, 定义扩展后TASM的语法和语义。采用扩展后的TASM为实际的实时嵌入式软件需求建模, 通过实验, 验证了采用扩展后的TASM为实时嵌入式软件需求建模的有效性。

关键词 需求建模语言; 实时; 嵌入式软件; 形式化定义; 扩展; 时间抽象状态机

Extending Timed Abstract State Machines for Real-Time Embedded Software

SHAN Jinhui[1], ZHANG Lu[2,†], WANG Jinbo[3], ZHANG Tao[3]

1. Huawei Technologies Co. Ltd., Shenzhen 518129; 2. School of Electronics Engineering and Computer Science, Peking University, Beijing 100871; 3. Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences, Beijing 100094; † Corresponding author, E-mail: zhanglu@sei.pku.edu.cn

Abstract According to the deficiency of Timed Abstract State Machine (TASM), TASM is extended with the data type of arrays, a loop rule named “while”, and some operators such as “%”,“&”, “|”, “^”, “>>”, “<<”, etc. The syntax and semantics of the extended TASM are defined. The extended TASM is applied to actual real-time embedded software to validate its effectiveness for requirements modeling.

Key words requirements modeling language; real-time; embedded software; formal definition; extension; timed abstract state machine

嵌入式系统是用来控制或者监视机器、装置、工厂等大规模设备的系统。嵌入式系统往往有实时性要求: 系统运行结果不仅取决于计算结果是否正确, 还取决于产生结果的时间是否满足要求。作为嵌入式系统的神经中枢, 嵌入式软件有举足轻重的作用。

软件需求是关于软件系统的能力、特征或质量因素等的陈述, 这些能力、特征或质量因素是为了让软件系统能对客户有用或有价值[1]。为实时嵌入式软件建立的需求模型可分为3类: 非形式化模型、半形式化模型和形式化模型。

早期采用自然语言描述软件需求规格说明。这种非形式化需求模型的优点在于建模成本较低, 缺点是容易产生歧义, 容易遗漏需求。

半形式化需求模型主要包括数据流图[2]、UML (unified modeling language)图[3]以及其扩展模型MARTE (modeling and analysis of real-time and embedded systems)[4−5]等。MARTE是UML在实时嵌入式领域的扩展, 其优势在于, 图形化需求模型很直观, 易于理解, 缺点是半形式化的需求模型难以支持形式化分析与验证[6]。

形式化需求模型主要包括时间自动机(timed automata, TA)[7]及其扩展模型 Uppaal TA (UTA)[8]、抽象状态机(abstract state machine, ASM)[9]及其扩展模型时间抽象状态机(timed ASM, TASM)[10−13]、Event-B[14−16]、Petri网[17]、面向航天嵌入式软件的

国家自然科学基金(61225007)资助
收稿日期: 2018−03−16; 修回日期: 2019−01−09; 网络出版日期: 2019−01−18

形式化建模语言 SPARDL[18]和扩展的状态迁移矩阵(state transition matrix, STM)[19]等。顾斌等[18]采用区间时序逻辑(interval temporal logic, ITL)[20]描述软件与时间有关的性质。候刚等[19]采用时间计算树逻辑(timed computation tree logic, TCTL)[21]描述软件与时间有关的性质。

ASM 是一种归约软件需求的形式化方法, 其语法小, 语义简单, 易于理解和使用[10]。文献[10–11]提出的 TASM 使用 ASM 的子集, 对 ASM 扩展了时间流逝和资源利用。Ouimet 等[10]采用 TCTL 的一个子集描述 TASM 规约中的安全性和活性。

形式化需求模型的优势在于具有严格的数学规约, 可直接用于形式化分析与验证, 有助于测试用例自动生成[10], 缺点在于很多形式化需求模型需要经过大量数学培训的人员才能使用。包括图灵奖得主 Pnueli 在内的许多计算机科学家都认为, 采用形式化方法对软件进行建模与验证是保证软件质量的重要手段[22]。

虽然目前有多种形式化需求建模语言, 但设计一种易于使用的实时嵌入式软件形式化需求建模语言, 并严格地定义其语法和语义, 是一个具有挑战性的任务。TASM 是一种易于使用的形式化需求建模语言, 基于规则形式为需求建模。TASM 能够支持时间、资源、同步和并发等行为的描述, 还可以表示体系结构分析与设计语言(architecture analysis and design language)子集的转换语义[13]。

但是, TASM 在为嵌入式系统建模时有以下不足之处。

1) 支持的数据类型有限, 不支持数组。

2) 尽管 TASM 各主抽象状态机可循环执行, 但没有提供各动作循环执行的规则[11]。由于嵌入式软件通常都很大, 仅用主抽象状态机为这些软件建模是不现实的, 故当处理情况较多时, 需要在一个抽象状态机内建立几十条乃至上百条规则。

3) 支持的运算符有限, 不支持“%”和实时嵌入式软件常见的二进制位运算符等。

4) 没有形式化地定义抽象状态机的调用和终止语义。

例如, 某嵌入式软件有以下需求: 软件接收数据包后, 如果数据包长度小于所要求的长度(如 144 字节), 应在数据包尾部填充字符, 直到数据包长度满足要求。由于 TASM 不支持数组, 仅使用 TASM 现有的整数类型、实数类型、布尔类型和用户自定义类型, 很难表示数据包。由于 TASM 没有提供各动作循环执行的规则, 故 TASM 也难以用一条规则表示根据接收到的数据包长度不同, 在数据包尾部填充不同个数的填充字符。此外, TASM 需要用两条规则来实现取余运算[10]。

因此, 本文对 TASM 进行扩展, 增加数组数据类型、while 循环规则以及“%”、“&”、“|”、“^”、“>>”、“<<”等运算符, 定义扩展后 TASM 的语法和语义, 并采用扩展后的 TASM 为实际的实时嵌入式软件需求建模。

1 相关工作

1.1 需求建模

UML 在顺序图上增加交互框架(interaction frames)后, 支持循环处理[3]。

除时间值外, UTA 不支持浮点数据类型[10]。Börger 等[9]认为, Petri 网是一类特定的多代理 ASM。ASM 避免使用无关的数学符号, 易于理解和使用, 不需要大量的数学培训[10]。ASM 支持循环处理, 但是没有提供时间流逝和资源利用的描述。

TASM 使用 ASM 的子集, 对 ASM 扩展了时间流逝和资源利用。与其他形式化需求模型相比, TASM 具有以下特点: 1) 采用规则的形式描述软件需求, 克服了非形式化需求方法遗漏软件需求的缺点; 并且便于人们理解 TASM 模型, 增加了 TASM 的易用性; 2) 使用模型检验工具 Uppaal[8]以及仿真分析工具 TASM toolset 对需求的无死锁性、安全性、活性、时间正确性和资源利用正确性等关键性质进行分析与验证[10], 克服了半形式化需求模型难以支持形式化分析与验证的缺点。

但是, TASM 在为嵌入式系统建模时有一些不足之处。本文对 TASM 进行扩展。Zhou 等[12]对 TASM 进行扩展, 增加了事件和观察器结构, 但所扩展的内容与本文不同。

1.2 ASM 及扩展模型建模语言的语法和语义

Börger 等[9]将 ASM 中关于签名 S(一组个数有限的函数名称)的状态 A 定义为一个非空集合 X(即 A 的定义域)加上 S 中函数名称的解释; ASM 中状态 A 的位置是函数名称加上函数参数; 函数值是状态 A 在一个位置的内容。状态 A 的一个修改是一个二元组 (l, v), 其中 l 是 A 的位置, v 是 X 中的一个元素。修改 (l, v) 的含义是状态 A 的位置 l 的内容被修改为值 v。一个修改集是修改的一个集合。ASM 的迁移

规则有skip规则、更新规则、块规则(即并发规则)、条件规则、赋值规则、forall规则、choose规则、顺序规则和调用规则等。文献[9]定义了ASM的语法, 并在修改集的基础上定义了ASM迁移规则的语义。

TASM是对ASM的实时扩展, 保留了ASM的条件语句、赋值语句, 删除了forall, choose, import结构[11]。文献[10]定义了TASM的语法, 用ASM表达TASM的语义。抽象状态机的每个执行步骤对应一条规则执行, 一个执行步骤的修改集定义为该步所有对受控变量的修改之集合, 一个抽象状态机的运行定义为一个修改集序列。

文献[11]利用修改集简要地定义以下语义: 一条规则内各Action之间的层次组装语义, 处于高层的抽象状态机与其所调用的子抽象状态机、函数抽象状态机之间的层次组装语义, 以及多个主抽象状态机之间的并发组装语义。文献[10−11]非形式地说明了TASM中主抽象状态机、子抽象状态机、函数抽象状态机的调用和终止语义。

本文定义了扩展后的TASM的语法和语义。本文所定义的位置和状态与ASM不同。本文将位置定义为将要执行的TASM抽象状态机规则中的动作序号, 或者一个抽象状态机的入口, 将要计算TASM抽象状态机各规则的Condition, 或者一个抽象状态机的终止; 将状态定义为TASM中各变量的取值; 定义TASM的格局为由位置和状态组成的二元组; 在格局之间的迁移关系中定义了TASM规则执行过程中位置和状态的计算方法。

与文献[10−11]相比, 本文一方面形式化地详细定义了一条规则内Actions中各Action的执行语义; 另一方面, 形式化地定义了主抽象状态机、子抽象状态机、函数抽象状态机的调用和终止语义。

1.3 Event-B与TASM建模方法的比较

Event-B方法[14−16]是一种与ASM较为相近的需求建模方法。

Event-B的模型由状态机和环境组成: 1) 状态机包含模型的动态部分: 变量、不变式、定理、变体和事件; 2) 环境包含模型的静态部分: 载体集合、常量、公理和定理。

Event-B中定义的数学模型通过状态以及相应的状态迁移(也即事件表示)。Event-B中事件由守卫和Action组成。Event-B用不变式描述状态变量的性质, 并通过证明的方式来保证状态迁移仍然保持不变式。Event-B的基础是一阶逻辑和类型集合论, 其变量的类型可为内置类型(如布尔类型和整型), 也可以是用户自定义类型。

Event-B采用基于证明的开发方法, 通过状态机精化实现模型的逐步开发建模, 开发过程中从抽象模型逐步引入更多的细节。在抽象模型中已被证明的性质, 在精化模型中仍然能够保持, 进而在后续一系列精化模型中仍然能够保持。

ASM方法基于具有抽象状态修改机制的代数框架。一个抽象状态机由签名、抽象和规则集合组成。ASM包含一种精化关系, 用于系统的增量式设计。

TASM提供对时间和资源的显式支持。文献[10]采用TCTL的一个子集描述TASM规约中的安全性和活性。

2 预备知识

定义1　TASM规约[10−11,13]为一个二元组TASMSPEC = ⟨E, ASMs⟩。

1) 环境E = ⟨EV, TU, ERs⟩。EV表示环境变量, 其类型定义在TU中, 主要包括整数类型、布尔类型、实数类型以及用户自定义类型等。ERs由零个或多个ER组成, 每个ER是形如“rn := rs;”的资源变量的定义, 如处理器、存储器、带宽和功耗等, rn是资源变量, rs表示资源的界限。

2) ASMs为一个或多个抽象状态机, 一个抽象状态机是一个元组ASM = ⟨McnName, MV, CV, IV, Rules⟩。McnName为抽象状态机名称。监控变量MV(monitored variables)为影响抽象状态机执行的环境变量的集合, 监控变量只可读。受控变量CV(controlled variables)为抽象状态机将要更新的环境变量的集合。内部变量IV(internal variables)为抽象状态机内部使用的中间变量集合, 不受环境的影响。Rules由一条或多条规则组成, 一条规则是一个元组Rule=⟨RuleName, T, RRs, r⟩, 其中RuleName是规则的名称。T表示规则执行的持续时间, 可以是一个固定值, 或是一个区间[$t_{\min}$, $t_{\max}$], 或是关键字next(“t:=next;”表示抽象状态机处于等待状态, 直到某个事件发生, 其中t是唯一的时钟变量, 表示全局系统时间), T也可以为空。RRs由零个或多个RR组成, 每个RR是形如“rn:=RS;”的资源利用。如果一个抽象状态机执行过程中所利用的资源数量超出资源的界限, 则执行终止。r是形如“if Condition then Actions”的具体规则, 其中Condition

是条件表达式; Actions 由一个或多个 Action 组成, 每个 Action 称为一个动作, 包括对受控变量的赋值或者空操作“skip;”等; *r* 也可以为“else then Actions”形式的具体规则。一个 ASM 各规则之间是隐式互斥的。

除时间和资源行为外, TASM 还支持并发组合、层次组合以及同步通信等行为的描述。TASM 将抽象状态机分为 3 个类型: 主抽象状态机(main ASM)、子抽象状态机(sub ASM)和函数抽象状态机(function ASM)。

各个主抽象状态机并发执行。主抽象状态机可以包含子抽象状态机和函数抽象状态机。子抽象状态机可进一步包含子抽象状态机和函数抽象状态机。子抽象状态机的调用形式是 SubMcnName( ), SubMcnName 是子抽象状态机名称, 子抽象状态机调用只允许出现在 Action 中, 一个函数抽象状态机不允许更新环境, 必须仅根据其输入变量的值产生输出变量的值。一个函数抽象状态机唯一的副作用是时间和资源利用。函数抽象状态机的调用形式是 FunMcnName(params), 其中 FunMcnName 是函数抽象状态机名称, params 是向函数抽象状态机传递的参数列表。函数抽象状态机调用允许出现在 Condition 和 Action 中。

对于一个主抽象状态机中含有“*t*:=next;”的规则, 在该规则执行开始时, 将 TASMSPEC 的状态进行缓存。当该规则被使能时, 将当前状态与所缓存的状态进行比较。如果两个状态不一致, 则该主抽象状态机继续执行该规则; 否则, 该主抽象状态机继续等待, 直到两个状态不一致[10]。子抽象状态机和函数抽象状态机中禁止使用 next 关键字。各主抽象状态机之间用共享变量进行通信[10]。TASMSPEC 中各抽象状态机隐式互斥访问共享变量。

3 TASM 的扩展

3.1 扩展后的 TASM 的语法

本文使用以下语言 L 定义扩展后 TASM 的抽象语法, 如图 1 所示。其中“Rule::= RuleName {T RRs while Condition Actions }”是本文扩展的语法。

L 的符号集={if, else, then, while, skip, *t*, next, now, Integer, Float, Boolean, True, False, //, :=, (,), [,]}∪VAR∪CON∪FUN∪PRI。其中 VAR 为变量符号集合, VAR 由各抽象状态机的 EV, RN, MV,

TASMSPEC::= E ASMs
E::= EV TU ERs
ERs::= ERs ER | null
ER::= rn := rs;
rs::= [lower, upper]
ASMs::= ASMs ASM | ASM
ASM::= MainASM | SubASM | FunASM
MainASM::= MainMcnName MV CV IV Rules
Rules::= Rules Rule | Rule
Rule::= RuleName { *T* RRs if Condition then Actions }
| RuleName { *T* RRs while Condition Actions }
| RuleName { *T* RRs else then Actions }
T::= *t* := Expression; | *t* := [t_{min}, t_{max}]; | *t* := next; | null
RRs::= RRs RR| null
RR::= rn := RS;
Condition::= Expression
SubASM::= SubMcnName MV CV IV Rules
FunASM::= FunMcnName InputVs OutputV IV Rules
Actions::= Actions Action | Action
Action::= *v* := Expression; | SubMcnName( ); | *v* := FunMcnName(params); | skip;
params::= params, param | param

图 1 扩展后的 TASM 的抽象语法
Fig. 1 Abstract syntax of the extended TASM

CV, IV 和 InputVs 中的变量以及 OutputV 和 t 组成, RN 是资源变量的集合, rn∈RN, v∈CV, 假定 TASMSPEC 中的变量名称唯一, 并令#(VAR)=n; CON 为常量符号集合, lower, upper, t_{min}, t_{max}, RS∈CON; FUN 是包括+, –, *, /, %, &, |, ^, >>, <<, SubMcnName 和 FunMcnName 等在内的函数符号集合; PRI 是包括 and, or, not 等在内的谓词符号集合。VAR, CON, FUN 和 PRI 均是递归可枚举的。

由 L 生成的 TASMSPEC 的规则由以下几种具体规则(在不引起混淆的情况下简称规则)组成: 1) if 规则: if Condition then Actions; 2) while 规则: while Condition Actions; 3) else 规则: else then Actions。Condition 是一个表达式。Actions 由一个或多个动作组成。每个动作为赋值"v := Expression;"、子抽象状态机调用、函数抽象状态机调用或空操作"skip;"。各抽象状态机、规则的名称必须唯一。假定表达式是用 FUN 或 PRI 中的符号以及"("、")"、"["与"]"对 VAR 或 CON 中的符号进行运算组成。将 TASMSPEC 中抽象状态机名称的全体、规则名称的全体分别记为 M^B 和 R^B。

文献[10]第 394 页仅将 TASMVariable 定义为 TASMName, 其中 TASMName 是以大写或小写英文字母开头, 后面是零个或多个字母、数字或下划线。

本文以 TASM 建模工具 TASM toolset 的语法[10]为基础, 开发一个为扩展 TASM 模型建模的工具, 称为 eTASM (extended TASM tool)。在 eTASM 的文法定义中, 将 TASMVariable 的定义扩展为: TASMVariable: TASMName | TASMVariable '[' Expression ']', 从而能够支持数组。

在 eTASM 的文法定义中, 将变量声明 TASMVarDecl 定义为 TASMVarDecl: TASMTypeName TASMVariable ';', 其中 TASMVariable 可为数组类型, TASMVariable 的各数组元素均为数据类型 TASMTypeName, 从而在文法上保证该数组元素的类型相同。

3.2 扩展后的 TASM 的语义

对于给定的 TASM 规约 TASMSPEC, 对其中出现的每个变量 v, 取一个有效的取值集合 D, 称为域, #(D)≥1, 令 $D_\omega = D \cup \{\omega_p\}$, ω_p 是一个不属于 D 的元素, 代表类型 p 的未定义值, 其中 p 是与域 D 关联的类型。例如, $Boolean_\omega$ = Boolean ∪ {ω_b}= True, False, ω_b}。只要不引起混淆, 就用 ω 代替 ω_p 或 ω_b[23]。D_ω 上的值满足通常意义下的相等关系, 即 $\forall d \in D$, $d \neq \omega$; ω 与 ω 相等, 记作 $\omega = \omega$。v 的论域为 D_ω。对 TASMSPEC 中出现的其他符号, 赋予通常意义下的解释。为使 TASMSPEC 有意义, 假定 TASMSPEC 的规则中 Condition 的计算结果属于 $Boolean_\omega$。假定一个抽象状态机内各规则满足一致性要求, 即不会同时有多于一条规则被使能[10,24–25]。当一个抽象状态机中有 else 规则时, 仅使能 Condition 值为 True 的 if 规则或 while 规则; 否则使能 else 规则。

定义 2　$d_i \in D_{i\omega}$ ($i \in I_+$, $i \le n$)为变量 v_i 的值, v_i∈VAR, 则称 $\sigma = \langle d_1, \ldots, d_n \rangle$ 为 TASMSPEC 的一个状态[23]。将 ASMSPEC 所有状态的全体记为 Σ, 即 $\Sigma = D_{1\omega} \times \ldots \times D_{n\omega}$。设 σ_1 和 σ_2 为 Σ 中两个状态, $\sigma_1 = \langle d_1, \ldots, d_n \rangle$, $\sigma_2 = \langle h_1, \ldots, h_n \rangle$, 若 $\forall i \in I_+$, $i \le n$, $d_i = h_i$, 则称 σ_1 与 σ_2 相等, 记为 $\sigma_1 = \sigma_2$; 否则称 σ_1 与 σ_2 不相等, 记为 $\sigma_1 \neq \sigma_2$。

令#(Actions)表示一条规则的 Actions 中动作的数量。对于 $j \in I_+$, $j \le$#(Actions), 令 Action(j)表示 Actions 中第 j 个动作。

对于名称为 M 的抽象状态机, 令 Type(M)表示抽象状态机 M 的类型, 分别用 MAIN, SUB 和 FUNCTION 表示主抽象状态机、子抽象状态机、函数抽象状态机的类型。

对于名称为 M 的子抽象状态机或函数抽象状态机, 令 Main(M)表示该子抽象状态机或函数抽象状态机所处的主抽象状态机的名称。对于名称为 M 的主抽象状态机, 令 Main(M) = M。

定义3　称 $\langle M, R, E \rangle \in M^B \times R^B \cup \{\varepsilon\} \times N \cup \{\perp\}$ 为 TASMSPEC 的主抽象状态机 Main(M)中的一个位置(在不引起混淆的情况下简称为 TASMSPEC 的一个位置)。M 为当前正在运行的抽象状态机名称, R 为该抽象状态机中当前正在执行的规则名称, E 为该规则中正准备执行的动作序号。ε 与任何规则名称不相同, 即 $\forall R \in R^B$, $R \neq \varepsilon$; ε 与 ε 相等, 记为 $\varepsilon = \varepsilon$。特别地, 位置$\langle M, \varepsilon, 0 \rangle$表示处于名称为 M 的抽象状态机的入口, 准备计算该抽象状态机各规则的 Condition; 位置$\langle M, R,$ #(Actions)+1$\rangle$表示刚执行完名称为 M 的抽象状态机中名称为 R 的规则的最后一个动作; 位置$\langle M, \varepsilon, \perp \rangle$表示名称为 M 的抽象状态机终止。将 TASMSPEC 中位置的全体记为 L^B。

假设每个主抽象状态机维护一个调用栈, 用于保存位置信息。对于名称为 M_1 的主抽象状态机和

位置$\langle M_2, R, E\rangle$, 令 Push(M_1, M_2, R, E)、Pop(M_1)分别表示将位置$\langle M_2, R, E\rangle$压入主抽象状态机 M_1 的调用栈、弹出并返回主抽象状态机 M_1 的调用栈顶元素。当调用子抽象状态机、函数抽象状态机时, 通过调用 Push 操作, 将调用处位置压入调用栈。当被调用的子抽象状态机、函数抽象状态机终止后, 通过调用 Pop 操作, 从调用栈获得调用处位置, 并返回调用处继续执行。

本文假设一个主机至多有一条规则出现“t:=next;”。为每个出现“t:=next;”的规则分别增加一个整型变量 state_saved 和状态变量$\sigma' \in \Sigma$, 在系统初始化时, 置 state_saved 初值为 0。当一个主机有多于一条规则出现“t:=next;”时, 为每条出现“t:=next;”的规则分别增加一个整型变量 state_saved 和状态变量$\sigma' \in \Sigma$, 并按照与一个主机至多有一条规则出现“t:=next;”类似的处理方法进行处理。

定义 4 位置集合 L^{B} 上的 ReturnFrom 函数定义为 ReturnFrom (M_1, R_1, E_1) = $\langle M_2, R_2, E_2\rangle$, 当且仅当满足以下条件之一时成立。

1) M_1 中存在一条规则 R_1 {T RRs if Condition then Actions}或 R_1 {T RRs while Condition Actions}或R_1 {T RRs else then Actions}, 使得 E_1 < #(Actions), 并且$\langle M_2, R_2, E_2\rangle = \langle M_1, R_1, E_1 + 1\rangle$。

2) M_1 中存在一条规则 R_1 {T RRs if Condition then Actions}或R_1 {T RRs else then Actions}, 使得 E_1 = #(Actions), 并且

① Type (M_1) = MAIN, 且$\langle M_2, R_2, E_2\rangle = \langle M_1, \varepsilon, 0\rangle$,

② Type (M_1) = SUB 或 Type (M_1) = FUNCTION, 且 $\langle M_2, R_2, E_2\rangle$ = ReturnFrom (Pop(Main(M_1)))。

3) M_1 中存在一条规则 R_1 {T RRs while Condition Actions}, 使得 E_1 = #(Actions), 且

① Condition = True, $\langle M_2, R_2, E_2\rangle = \langle M_1, R_1, 1\rangle$;

② Condition = False, 且

a) Type(M_1) = MAIN, 并且$\langle M_2, R_2, E_2\rangle = \langle M_1, \varepsilon, 0\rangle$,

b) Type(M_1) = SUB 或 Type(M_1) = FUNCTION, 且 $\langle M_2, R_2, E_2\rangle$ = ReturnFrom(Pop(Main(M_1)))。

非形式化地说, 定义 4 定义的 ReturnFrom (M_1, R_1, E_1)函数是在位置$\langle M_1, R_1, E_1\rangle$调用的子抽象状态机或函数抽象状态机一旦终止, 计算返回调用处后的下一个执行位置。定义 4 中第 1 个条件指调用子抽象状态机或函数抽象状态机位置是在一条规则的最后一个 Action 之前, 第 2 个条件指调用子抽象状态机或函数抽象状态机位置是在一条 if 规则或 else 规则的最后一个 Action, 第 3 个条件指调用子抽象状态机或函数抽象状态机位置是在一条 while 规则的最后一个 Action。

定义 5 称$\langle l, \sigma\rangle \in L^{\mathrm{B}} \times \Sigma$为 TASMSPEC 的一个格局[23]。

定义 6 格局集合上的迁移关系[23]⇒定义为$\langle l_1, \sigma_1\rangle \Rightarrow \langle l_2, \sigma_2\rangle$, 当且仅当 $l_1 = \langle M_1, R_1, E_1\rangle$, $l_2 = \langle M_2, R_2, E_2\rangle$, 并且以下条件之一成立。

1) $E_1 = 0$, M_1 中存在一条规则 R_3 {T RRs if Condition then Actions}, 使得 Condition = True, $\langle M_2, R_2, E_2\rangle = \langle M_1, R_3, 1\rangle$; 或 M_1 中存在一条规则 R_4 {T RRs while Condition Actions}, 使得 Condition = True, $\langle M_2, R_2, E_2\rangle = \langle M_1, R_4, 1\rangle$; 或对于 M_1 中任意一条规则 R_3 {T RRs if $\mathrm{Condition}_1$ then Actions}以及 R_4 {T RRs while $\mathrm{Condition}_2$ Actions}, $\mathrm{Condition}_1 \neq$ True, $\mathrm{Condition}_2 \neq$ True, M_1 中存在一条规则 R_5 {T RRs else then Actions}, 使得$\langle M_2, R_2, E_2\rangle = \langle M_1, R_5, 1\rangle$, 并且

① T 为 t:= $\mathrm{Expression}_1$, 且$\sigma_2 = \sigma_1[t / \mathrm{value}_1, \mathrm{RN} \rightarrow \mathrm{RRs}]$; 其中 value_1 是 $\mathrm{Expression}_1$ 的计算结果。$\sigma_1[\mathrm{RN} \rightarrow \mathrm{RRs}]$表示:

a) 如果 RRs 为 null, 那么$\sigma_1[\mathrm{RN} \rightarrow \mathrm{RRs}]$亦为空;

b) 如果 RRs 为 $\mathrm{rn}_1 := \mathrm{RS}_1$; …; $\mathrm{rn}_k := \mathrm{RS}_k$, 且$\forall i \in I_+$, $i \leq k$, rn_i 消耗的资源没有超出其界限, 那么分别将 RS_1, …, RS_k 的计算结果赋给σ_1 中资源变量 rn_1, …, rn_k, σ_1 中其余变量值保持不变;

c) 如果 RRs 为 $\mathrm{rn}_1 := \mathrm{RS}_1$; …; $\mathrm{rn}_k := \mathrm{RS}_k$; 并且$\exists i \in I_+$, $i \leq k$, rn_i 消耗的资源超出其界限, 那么

ⓐ 如果 Type(M_1) = MAIN, 将$\langle M_2, R_2, E_2\rangle$ 修改为 $\langle M_1, \varepsilon, \perp\rangle$,

ⓑ 如果 Type(M_1) =SUB 或 Type(M_1)= FUNCTION, 将$\langle M_2, R_2, E_2\rangle$ 修改为 ReturnFrom (Pop(Main(M_1))) (注: 当 $\mathrm{RS}_i > 0 (i \in I_+, i \leq k)$时, 资源 rn_i 要被消耗 RS_i 个单位; 当 $\mathrm{RS}_i < 0$ 时, 资源 rn_i 要被返还 RS_i 个单位);

② T 为 t:=[$t_{\min}$, $t_{\max}$]; 并且$\sigma_2 = \sigma_1[t/\mathrm{value}_2, \mathrm{RN} \rightarrow \mathrm{RRs}]$。其中 value_2 是区间[$t_{\min}$, $t_{\max}$]中的一个随机整数值;

③ T 为 t:=next;

a) 如果 state_saved 为 0, 那么σ':=σ_1, state_ saved:= 1,

ⓐ 如果$\sigma' = \sigma_1$, 就将$\langle M_2, R_2, E_2\rangle$修改为$\langle M_1, \varepsilon, 0\rangle$, 并且$\sigma_2 = \sigma_1$,

ⓑ 如果$\sigma' \neq \sigma_1$, 令 state_saved:= 0, 且$\sigma_2 = \sigma_1[\mathrm{RN} \rightarrow \mathrm{RRs}]$;

b) 如果 state_saved 为 1, 那么

ⓐ如果$\sigma' = \sigma_1$, 就将$\langle M_2, R_2, E_2\rangle$ 修改为$\langle M_1, \varepsilon, 0\rangle$, 并且$\sigma_2 = \sigma_1$,

ⓑ如果$\sigma' \neq \sigma_1$, 令 state_saved:=0, 并且 $\sigma_2 = \sigma_1$ [RN → RRs];

④ T 为 null 并且$\sigma_2 = \sigma_1$[RN → RRs]。

非形式化地说, 定义 6 中第 1 个条件指处于名称为 M_1 的抽象状态机的入口, M_1 的某一条 if 规则的 Condition 为 True, 或 M_1 的某一条 while 规则的 Condition 为 True, 或 M_1 的所有 if 规则和 while 规则的 Condition 均不为 True, 并且 M_1 存在一条 else 规则的情况。

2) $E_1 = 0$, 对于 M_1 中任意一条规则 R_3 {T RRs if Condition_1 then Actions}和 R_4 {T RRs while Condition_2 Actions}, $\text{Condition}_1 \neq$ True, $\text{Condition}_2 \neq$ True, 并且 M_1 中不存在一条规则 R_5 {T RRs else then Actions}, 使得

① Type (M_1) = MAIN, $\langle M_2, R_2, E_2\rangle = \langle M_1, \varepsilon, \perp\rangle$, 并且$\sigma_2 = \sigma_1$,

② Type(M_1) = SUB 或 Type(M_1) = FUNCTION, $\langle M_2, R_2, E_2\rangle$ = ReturnFrom (Pop(Main(M_1))), 并且$\sigma_2 = \sigma_1$。

非形式化地说, 定义 6 中第 2 个条件指处于名称为 M_1 的抽象状态机的入口, M_1 的所有 if 规则和 while 规则的 Condition 均不为 True, 并且 M_1 不存在一条 else 规则的情况。

3) $E_1 \in I_+$, M_1 中存在一条规则 R_1{T RRs if Condition_1 then Actions}或 R_1 {T RRs while Condition_2 Actions}或 R_1 {T RRs else then Actions}, 使得 $E_1 \leq$ #(Actions), 并且

① Action(E_1)为 v:=Expression_3; $\langle M_2, R_2, E_2\rangle = \langle M_1, R_1, E_1+1\rangle$, 并且$\sigma_2 = \sigma_1[v/\text{value}_3]$, 其中 value_3 是 Expression_3 的计算结果,

② Action(E_1) 为 SubMcnName( ); $\langle M_2, R_2, E_2\rangle$= ⟨SubMcnName, ε, 0⟩, Push(Main(M_1), M_1, R_1, E_1), 且$\sigma_2 = \sigma_1$,

③ Action(E_1)为 v := FunMcnName(params); $\langle M_2, R_2, E_2\rangle$ = ⟨FunMcnName, ε, 0⟩, Push(Main(M_1), M_1, R_1, E_1), 且$\sigma_2 = \sigma_1$,

④ Action(E_1)为 skip; $\langle M_2, R_2, E_2\rangle = \langle M_1, R_1, E_1 + 1\rangle$, 且$\sigma_2 = \sigma_1$。

非形式化地说, 定义 6 第 3 个条件指在执行名称为 M_1 的抽象状态机的一条规则中的一个 Action 时的情况。

4) $E_1 \in I_+$, M_1 中存在一条规则 R_1{T RRs if Condition then Actions}或者 R_1{T RRs else then Actions}, 使得 E_1=#(Actions)+1, 并且 Type(M_1) = MAIN, $\langle M_2, R_2, E_2\rangle = \langle M_1, \varepsilon, 0\rangle$; 或者 Type($M_1$) = SUB 或 Type($M_1$)=FUNCTION, $\langle M_2, R_2, E_2\rangle$ = ReturnFrom(Pop(Main(M_1))); 并且

① Action(E_1–1)为 v:=Expression_3; 且$\sigma_2 = \sigma_1$,

② Action(E_1–1)为 SubMcnName ( ); 且$\sigma_2 = \sigma_1$,

③ Action(E_1–1)为 v:=FunMcnName (params); 且$\sigma_2 = \sigma_1[v/\text{OutputValue}]$。其中$\sigma_1[v/\text{OutputValue}]$表示用调用名称为 FunMcnName 的函数抽象状态机返回时输出变量的值 OutputValue 更新σ_1 中变量 v 的值, σ_1 中其余变量值保持不变,

④ Action(E_1–1)为 skip; 且$\sigma_2 = \sigma_1$。

非形式化地说, 定义 6 第 4 个条件指在执行完名称为 M_1 的抽象状态机的一条 if 规则或 else 规则中的最后一个 Action 后的情况。

5) $E_1 \in I_+$, M_1 中存在一条规则 R_1{T RRs while Condition Actions}, 使得 E_1=#(Actions)+1, 并且 Condition=True, $\langle M_2, R_2, E_2\rangle = \langle M_1, R_1, 1\rangle$; 或 Condition=False, Type(M_1) = MAIN, $\langle M_2, R_2, E_2\rangle$ = $\langle M_1, \varepsilon, 0\rangle$; 或 Condition=False, Type(M_1) =SUB 或 Type(M_1) = FUNCTION, $\langle M_2, R_2, E_2\rangle$ = ReturnFrom (Pop(Main(M_1))), 并且

① Action(E_1–1)为 v:= Expression_3; 且$\sigma_2 = \sigma_1$,

② Action(E_1–1)为 SubMcnName( ); 且$\sigma_2 = \sigma_1$,

③ Action(E_1–1)为 v:= FunMcnName (params); 且$\sigma_2 = \sigma_1[v/\text{OutputValue}]$,

④ Action(E_1–1)为 skip; 并且$\sigma_2 = \sigma_1$。

6) 非形式化地说, 定义 6 第 5 个条件指在执行完名称为 M_1 的抽象状态机的一条 while 规则中的最后一个 Action 后的情况。

$E_1 = \perp$, 即$\langle M_1, R_1, E_1\rangle = \langle M_1, \varepsilon, \perp\rangle$, $\langle M_2, R_2, E_2\rangle = \langle M_1, \varepsilon, \perp\rangle$, 并且$\sigma_2 = \sigma_1$。

非形式化地说, 定义 6 第 6 个条件指名称为 M_1 的抽象状态机终止的情况。

此外, 当计算一条规则的 Condition 或在一条规则的 Action 中调用函数抽象状态机的某个参数中遇到函数抽象状态机调用 FunMcnName (params)时, 主抽象状态机中的位置为⟨FunMcnName, ε, 0⟩。该函数抽象状态机执行过程中满足迁移关系⇒, 并且该函数抽象状态机执行结束时, 用其输出变量的值“替换(substitute)”调用中的 FunMcnName (params)。

4 扩展的TASM模型建模工具开发与实验研究

4.1 扩展的TASM模型建模工具开发

我们以 TASM 建模工具 TASM toolset 的语法[10]为基础, 采用词法分析器/语法分析器自动生成工具 lex/yacc 的 Windows 版本 flex/Bison, 开发了一个为扩展 TASM 模型建模的工具 eTASM。eTASM 可自动识别扩展 TASM 模型, 并进行格式化输出。

文献[10]将 TASMRule 定义为: TASMRule: TASMRuleName ' {' [TASMTimeSpec]{ TASMResourceSpec } TASMRuleDef '}', 其中 TASMRuleName 定义为 TASMName。

对函数抽象状态机[10]的定义:

Listing D.13 Rules of the rotateClockwise function machine

R1: Don't go under 0... {

TASMName 无法识别“R1: Don't go under 0...”。本文在 eTASM 的文法定义中修改了 TASM 的语法, 将 TASMRuleName 的定义改为“TASMRule-Name ':' TASMDescription”, 其中 TASMDescription 定义为字符串。并且, 我们将文献[10]中附录 D, E 和 F 中 TASM 模型的所有规则定义中的“:”与“{”之间的字符串前后都加上“”””。

我们将文献[10]中附录 D, E 和 F 中的 TASM 模型输入 eTASM, 除发现第 472 页和第 507 页两处语法错误外, 其他抽象状态机和规则均被 eTASM 识别。

4.2 实验

我们将 eTASM 用于为实际的实时嵌入式软件需求建模, 来验证扩展后的 TASM 语言为实时嵌入式软件形式化需求建模的有效性。

实验采用实际的实时嵌入式软件: 实验管理单元(experimental handling unit, EHU)主控软件。

EHU 接收平台系统的时间码、数据注入指令和数字量遥测请求等, 解析和执行数据注入指令, 对进行实验的两个设备配电、测控、通信控制与数据管理, 打包数据包并向平台系统发送。为提高可靠性, 系统中每种设备采用双机备份。

EHU 主控软件对外有 1394 接口、RS422 接口、程控指令接口、静态随机访问存储器(Static Random Access Memory, SRAM)接口、A/D 接口、EEPROM 接口和复位接口等。图 2 展示 EHU 主控软件的接口关系。

平台系统周期性地向 EHU 广播时间码, 校准 EHU 的时间, 周期性地向 EHU 发送数字量遥测请求。不定期地向 EHU 发送数据注入指令。EHU 向设备转发数据注入指令, 通过 RS422 接口周期性地向设备广播时间码, 向设备发送数据包请求。设备接收到数据包请求后, 在规定时间内向 EHU 返回一个数据包。EHU 接收设备的数据包, 进行处理, 打包成 1394 等时数据包, 保存在 SRAM 中; 采集自身及设备的模拟信号, 转换成数字量后组包成数字量遥测包; 通过 1394 接口向平台系统返回 1394 等时数据包和数字量遥测包; 通过程控指令接口向设备发送程控指令; 在 EEPROM 保存设备的主备状态; 不断给复位芯片发送喂狗信号(正负交替的脉冲)。如果复位芯片超过一定时间未接收到喂狗信号, 就向 EHU 发送狗叫复位信号(称为看门狗复位)。此外, EHU 主控软件还有两种复位方式: 上电和平台系统发送复位命令。当 EHU 复位时, 按照 EEPROM 保存的结果恢复设备主备状态。

4.3 实验项目抽象状态机的组成

EHU 主控软件的扩展 TASM 模型包含的主抽象状态机包括: 设备 1、设备 2、平台系统、复位芯片、1394 接口处理、RS422 接口处理、复位接口

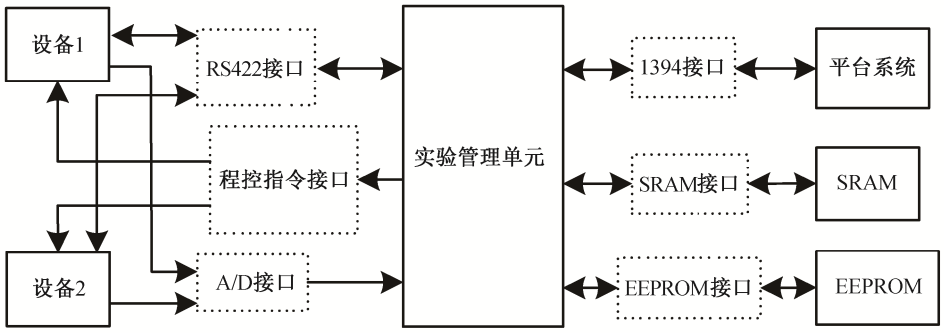

图 2 EHU 主控软件接口关系

Fig. 2 Interface relation of the EHU main control software

处理、事件执行、向设备广播时间码定时器、向设备1发送数据包请求定时器、接收设备1数据包定时器、向设备2发送数据包请求定时器、接收设备2数据包定时器、向EHU广播时间码定时器以及向EHU发送数字量遥测请求定时器。

EHU的功能很复杂，其功能已分解到1394 接口处理、RS422接口处理和事件执行主抽象状态机。EEPROM接口处理功能在事件执行和复位接口处理主抽象状态机中完成，SRAM接口处理功能在 1394接口处理、RS422接口处理主抽象状态机中完成，A/D接口处理功能在1394接口处理主抽象状态机中完成，程控指令接口处理功能在事件执行主抽象状态机中完成。

这些主抽象状态机并发运行。通过在主抽象状态机内识别层次成份，除定时器主抽象状态机外，其他主抽象状态机进一步由子抽象状态机和函数抽象状态机组成。由于空间所限，这里没有列出EHU主控软件和CCU主控软件的子抽象状态机和函数抽象状态机。

4.4 while 规则和数组数据类型

EHU接收设备1发送的数据包正确长度为144字节。如果EHU主控软件接收到设备1数据包长度小于144字节，应在数据包尾部填充字符0xBB，直到数据包长度满足要求。数据包填充字符子机的规则如图3所示。

图3中规则R1使用while规则，并且将数据包存放在一个数组中。

由于TASM不支持数组，故TASM利用现有的整数类型、实数类型、布尔类型和用户自定义类型很难表示长度为144字节的数据包。由于TASM没有提供各动作循环执行的规则，对于以上需求，TASM需要用一百多条规则，分别根据接收到数据包的不同长度，在数据包尾部填充不同个数的填充字符。利用数组和while规则，扩展后的TASM仅用两条规则就可表示上述需求。

4.5 资源消耗和“%”操作符

EHU将1394等时数据包保存在SRAM中，每个1394等时数据包大小为160字节。图4是设备1数据包存储子机中的一条规则。

该嵌入式软件将SRAM的容量作为一种资源，每向SRAM中存入一个1394等时数据包，就消耗160字节的SRAM。上述规则中使用“%”操作符。

由于TASM不支持“%”操作符，在文献[10]中，TASM需要专门定义一个函数抽象状态机，并用两

```
R1: "Number of fill characters for Equipment1data packet greater than 0" {
while NO_Fill_Char_Equip1_Data_Packet > 0
   Equip1_Data_Packet [144－NO_Fill_Char_Equip1_Data_Packet + 1]:= 0xBB;
   NO_Fill_Char_Equip1_Data_Packet:= NO_Fill_Char_Equip1_Data_Packet －1;
}
R2: "Number of fill characters for Equipment1 data packet not greater than 0" {
else then
   skip;
}
```

图 3　while 规则和数组数据类型示例片断

Fig. 3　Snippets of sample rule named while and the data type of arrays

```
R1: "SRAM not overflow after saving one Equipment1 isochronous packet" {
SRAMUsed:= 160;
if (Communication_Status_Equip1 = 1) and (SRAM_Overflow_Status = 0) and
    ((SRAM_Write_Ptr + 1) % SRAM_Length!= SRAM_Read_Ptr) then
      Data_length_to_be_copied:= 160;
      Copying_data_packet_of_equipment1_received_from_RS422_currently_to_SRAM ( );
      SRAM_Write_Ptr:= (SRAM_Write_Ptr + 1) % SRAM_Length;
}
```

图 4　if 规则和资源消耗示例片断

Fig. 4　Snippets of the rule named if and resource consumption

条规则来实现取余运算, 不直观, 容易发生错误。

4.6 时间流逝和定时器

Zhou 等[26]给出用 TASM 为单个定时器建模的方法。利用定时器, 可以为 EHU 主控软件的功能性需求“EHU 每 1 分钟±1 秒向设备广播一次时间码”建立扩展后的 TASM 模型。“向设备广播时间码定时器主机”的规则片断如图 5 所示。

系统初始化时, 定时器 BRDCST_time_code_equip_Timer 和变量 BRDCST_time_code_equip_Timeout_Flag 的值均为 0。因此图 5 中规则 R2 被连续使能, 该定时器开始累加。当该计时器值不小于 59000 时, 规则 R1 被使能, 将变量 BRDCST_time_code_equip_Timeout_Flag 的值置为 1, 将该定时器清零。此后规则 R3 一直被使能, 保持“向设备广播时间码定时器主机”“活着”。直到系统状态发生改变, 例如变量 BRDCST_time_code_equip_Timeout_Flag 被 RS422 接口处理主机的一个子机的一条规则置为 0。然后该定时器被清零, 图 5 中规则 R2 重新被连续使能, 定时器重新开始计时。

4.7 为非功能性需求建模

EHU 主控软件有性能需求“EHU 向设备广播时间码的周期为 1 分钟±1 秒”。采用扩展后的 TASM 为该性能需求的建模方法如下。

1) 在环境中定义自定义数据类型“Error_Cycle:={none, invalid_cycle};”, invalid_cycle 表示发生向设备广播时间码周期错误。

2) 每次向设备广播时间码时, 用变量分别记录本次向设备广播时间码时间和上次向设备广播时间码时间。

3) 在向设备广播时间码的子机中增加以下规则:

if (Current_Time_BRDCST_Time_Equip – Last_Time_ BRDCST_Time_Equip < 59 * 1000) or

(Current_Time_ BRDCST_Time_Equip – Last_Time_ BRDCST_Time_Equip > 61 * 1000) then

Error_Time_Code_Equip_Cycle:= invalid_cycle。

变量 Error_Time_Code_Equip_Cycle 的类型为 Error_Cycle。

4) 用时序逻辑公式表示该性能需求为 AG (Error_Time_Code_Equip_Cycle != invalid_cycle)。该时态逻辑公式表示变量 Error_Time_Code_Equip_Cycle 永远不会被赋值 invalid_cycle, 即扩展后的 TASM 模型满足该性能需求, 此需求也是一种安全性质。

4.8 为另一个实时嵌入式软件需求建模

我们还将 eTASM 用于为另一个实际的实时嵌入式软件——通讯与控制单元(Communication and Control Unit, CCU)主控软件需求建模。

CCU 主要完成 CCU 与载荷数据处理系统(Payload Data Handling Unit, PDHU)之间的 1553B 总线

```
R1: "broadcasting time code for equipment timer timeout" {
if (BRDCST_time_code_equip_timer >= 59*1000) and
      ( BRDCST_time_code_equip_Timeout_Flag = 0) then
      BRDCST_time_code_equip_Timeout_Flag:= 1;
      BRDCST_time_code_equip_Timer:= 0;
}
R2: "broadcasting time code for equipment timer not timeout" {
t:= 1;
if (BRDCST_time_code_equip_Timer < 59*1000) and
      (BRDCST_time_code_equip_Timeout_Flag = 0) then
      BRDCST_time_code_equip_Timer:= BRDCST_time_code_equip_Timer + 1;
}
R3: "broadcasting time code for equipment timer reset" {
t:= next;
else then
      BRDCST_time_code_equip_Timer:= 0;
}
```

图 5 “向设备广播时间码定时器主机”的规则片断

Fig. 5 Snippets of the rules in the main machine named “broadcasting time code for equipment timer”

通讯、CCU与两个外设之间的RS422通讯、CCU对外设1进行遥测、遥控和温控、CCU对定标机构进行定标、CCU解析和执行数据注入指令以及打包数据包并向PDHU发送等功能。CCU主控软件对外有1553B接口、RS422接口、遥测接口、遥控接口、温控接口、定标接口和复位接口等。

研究结果表明，eTASM可以有效地为CCU主控软件需求建模[27]。

4.9 讨论

Shan等[27]提出一种基于扩展TASM的实时嵌入式软件需求建模方法。该方法包括以下步骤：1) 识别并发成份，建立主抽象状态机；2) 在一个并发成份内识别层次成份，建立子抽象状态机和函数抽象状态机；3) 功能性需求规约；4) 非功能性需求规约；5) 根据需要对以上4个步骤进行迭代。第2步根据层次关系，采用分而治之、逐步求精的思想，将功能复杂的主抽象状态机和子抽象状态机逐步划分为子抽象状态机和函数抽象状态机。4.3节中模型的多个主抽象状态机体现了嵌入式系统的并发性。

从图3~5的规则片断可以看出，TASM支持资源消耗和时间流逝。TASM采用规则形式描述需求，便于人们理解TASM模型，增加了TASM 的易用性，并且便于列举出所感兴趣的全部情况，防止遗漏软件需求。

本文对TASM扩展了数组，扩展后的TASM可以描述数据包等内容。本文还对TASM扩展了while规则，扩展后的TASM可以在一条规则内描述需要重复处理的动作。另外，嵌入式软件常常需要进行取模运算、二进制位逻辑运算和移位操作，TASM难以便利地表达这种需求，因此本文扩展了“%”、“&”、“|”、“^”、“>>”、“<<”等运算符，扩展后的TASM可以很便利地表达这种需求。

通过将eTASM用于为实际的实时嵌入式软件需求建模，验证扩展后的TASM是一种易于使用的、有效的实时嵌入式软件形式化需求建模语言。

5 结束语

软件需求在实时嵌入式软件中发挥重要作用，设计一种易于使用的实时嵌入式软件形式化需求建模语言，并严格定义其语法和语义是一个具有挑战性的任务。TASM是一种易于使用的实时嵌入式软件形式化需求建模语言[10−11]，能够支持时间、资源、同步、并发等行为的描述，但是在为嵌入式系统建模时有一定的不足。

本文对TASM进行扩展，增加了数组数据类型、while循环规则以及“%”、“&”、“|”、“^”、“>>”、“<<”等运算符，并定义扩展后TASM的语法和语义。采用扩展后的TASM为实际的实时嵌入式软件需求建模，初步验证了扩展后的TASM为实时嵌入式软件需求建模的有效性。

未来的工作中，我们将采用扩展后的TASM为更多的实时嵌入式软件需求建模，以验证扩展后的TASM的需求建模能力。测试需求描述了具体的测试内容。将来我们会根据扩展后的TASM模型，自动生成测试需求和测试用例。软件需求往往会不断地演化，自动地验证需求模型是否完整、一致，将简化和缩短需求分析过程[25]。将来我们还会对扩展TASM模型的完整性和一致性进行分析和验证。

参考文献

[1] 金芝，刘璘，金英，编著. 软件需求工程：原理和方法. 北京：科学出版社, 2008

[2] Hull E, Jackson K, Dick J, et al. Requirements engineering. 3rd Ed. New York: Springer-Verlag, 2011

[3] Fowler M. UML distilled: a brief guide to the standard object modeling language. 3rd Ed. Boston: Addison Wesley Longman, 2003

[4] Object Management Group. A UML profile for MARTE: modeling and analysis of real-time and embedded systems, Version 1.1 [R/OL]. (2011–06) [2018–01–03]. http://www.omg.org/spec/MARTE/1.1

[5] Selic B, Gérard S. Modeling and analysis of real-time and embedded systems with UML and MARTE: developing cyber-physical systems. Burlington: Elsevier, 2014

[6] 杨志斌，皮磊，胡凯，等. 复杂嵌入式实时系统体系结构设计与分析语言：AADL. 软件学报，2010, 21(5): 899–915

[7] Bengtsson J, Yi W. Timed automata: semantics, algorithms and tools // Desel J, Reisig W, Rozenberg G. Lectures on concurrency and petri nets: advances in petri nets. New York: Springer-Verlag, 2004: 87–124

[8] Behrmann G, David A, Larsen K G. A tutorial on Uppaal // Bernardo M, Corradini F. Proceedings of the formal methods for the design of real-time systems. Bertinora: Springer-Verlag, 2004: 200–236

[9] Börger, E, Stärk R. Abstract state machines: a method for high-level system design and analysis. New York: Springer-Verlag, 2003

[10] Ouimet M. A formal framework for specification-based embedded real-time system engineering [D]. Cambridge: Massachusetts Institute of Technology, 2008

[11] Ouimet M, Lundqvist K. The timed abstract state machine language: abstract state machines for real-time system engineering. Journal of Universal Computer Science, 2008, 14(12): 2007–2033

[12] Zhou J, Lu Y, Lundqvist K. A TASM-based requirements validation approach for safety-critical embedded systems // Proceedings of the 19th International Conference on Reliable Software Technologies. Paris: Springer-Verlag, 2014: 43–57

[13] Yang Z, Hu K, Ma D, et al. From AADL to timed abstract state machines: a verified model transformation. The Journal of Systems and Software, 2014, 93: 42–68

[14] Cansell D, Méry D. The Event-B modelling method: concepts and case studies // Bjørner D, Henson M C. logics of specification language. New York: Springer-Verlag, 2008: 47–152

[15] Abrial J R. Modeling in Event-B: system and software engineering. Cambridge: Cambridge University Press, 2010

[16] 苏雯. 基于 Event-B 的混合系统形式化: 理论与实践[D]. 上海: 华东师范大学, 2013

[17] Best E, Devillers R, Koutny M. Petri net algebra. New York: Springer-Verlag, 2001

[18] 顾斌, 董云卫, 王政. 面向航天嵌入式软件的形式化建模方法. 软件学报, 2015, 26(2): 321–331

[19] 侯刚, 周宽久, 常军旺, 等. 基于时间 STM 的软件形式化建模与验证方法. 软件学报, 2015, 26(2): 223–238

[20] Dutertre B. Complete proof systems for first order interval temporal logic // Proceedings of the Tenth Annual IEEE Symposium on Logic in Computer Science. San Deigo, 1995: 36–43

[21] Alur R, Courcoubetis C, Dill D. Model-checking for real-time systems // Proceedings of the 5th Symposium on Logic in Computer Science. Philadelphia, 1990: 414–425

[22] Pnueli A. Verification engineering: a future profession // Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing. New York, 1997: 7

[23] Loeckx J, Sieber K, Stansifer R D. The foundations of program verification. 2nd ed. Chichester: John Wiley & Sons, 1987

[24] Heimdahl M P E, Leveson N G. Completeness and consistency in hierarchical state-based requirements. IEEE Transactions on Software Engineering, 1996, 22(6): 363–377

[25] Ouimet M, Lundqvist K. Automated verification of completeness and consistency of abstract state machine specifications using a SAT solver. Electronic Notes in Theoretical Computer Science, 2007, 190(2): 85–97

[26] Zhou J, Lu Y, Lundqvist K. The observer-based technique for requirements validation in embedded real-time systems // Proceedings of the 2014 IEEE 1st International Workshop on Requirements Engineering and Testing. Karlskrona, 2014: 47–54

[27] Shan J H, Zhao H Y, Wang J B, et al. An extended TASM-based requirements modeling approach for real-time embedded software: an industrial case study // Zhang L, Xu C. Software engineering and methodology for emerging domains. Singapore: Springer, 2016: 19–34