基于 SPIN 的 G 语言系统模型的验证

薛 艳[1] 武淑红[1] 王耀力[2]
(太原理工大学计算机科学与技术学院 山西 晋中 030600)[1]
(太原理工大学信息工程学院 山西 晋中 030600)[2]

摘 要 对于大型系统,为确保其运行的可靠性、稳定性及高效性,需要从两个方面对系统进行验证:业务模型和系统模型。目前,对业务模型的验证可通过 BPMN 来完成;对系统模型的验证可通过 SPIN(Simple Promela Interpreter)工具执行。G 语言是由 NI 公司创建的一种图形化程序框图语言,还未被加入 ANSI 标准,因此,文中第一步工作是提取 G 语言的形式、规则、文法等语言特性。由于 SPIN 对 G 语言不提供直接的支持,因此第二步工作是完成 G2Promela 的映射。在 G2Promela 的工作中,主要是基于编译器的框架,以 Scanner-Parser-Optimizer-Generator(SPOG 框架)为主线,根据第一步的预处理工作,按方法函数、指针、关键字、变量等分类创建 G2Promela 的映射规则,最终实现 G2Promela 的转换,完成对 G 语言系统模型的验证。该方法的提出弥补了 G 语言系统模型验证方面的空白,从而更深入地确保了 G 语言程序的性能。

关键词 系统模型,G 语言,SPIN,G2Promela,SPOG
中图法分类号 TP399 **文献标识码** A

Verification of G Language System Model Based on SPIN

XUE Yan[1] WU Shu-hong[1] WANG Yao-li[2]
(Department of Computer Science and Technology, Taiyuan University of Technology, Jinzhong, Shanxi 030600, China)[1]
(Department of Information Engineering, Taiyuan University of Technology, Jinzhong, Shanxi 030600, China)[2]

Abstract For large systems, in order to ensure the reliability, stability and efficiency of its operation, it is necessary to verify the system from two aspects, the business model and the system model. At present, the validation of the business model can be done through BPMN. For the system model validation, SPIN tool is selected. G language created by the NI company is a graphical block diagram language and has not yet joined the ANSI standard. Therefore, the first step is to extract the G language form, rules, grammar and other language features. SPIN does not provide direct support for the G language, so the second step is to complete the G2Promela mapping. In the work of G2Promela, mainly taking the framework of the compiler to Scanner- Parser- Optimizer- Generator (SPOG framework) as the main line, according to the first step of the pre-processing work, G2Promela mapping rules is classified and created through the method function, pointer, keywords, variables to realize the G language system model validation. The proposed method complements the gaps in the G language system model validation, thus further ensuring the performance of the G language program.

Keywords System model, G language, SPIN, G2Promela, SPOG

一般情况下,软件系统可以概括为两个模型:业务模型和系统模型。系统模型是指以某种形式依据形式化理论构建的实现具体功能的模型,可以从本质上体现系统的属性、流程及详细的技术设计;而业务模型只是对目标对象框架的描述,侧重于目标对象的业务流程及操作。因此,对软件产品的测试与验证可以从两个方面进行:1)业务模型(BP)的测试和分析;2)系统模型的检测。其中,系统模型的验证是指对系统模型中的形式化理论、算法的逻辑结构及流程实现的代码进行系统用例测试,验证其是否存在逻辑缺陷、死锁及活锁问题;而业务模型的验证是对系统的业务流程进行用例驱动测试,检测其是否存在流程设计上的不合理或定量属性不兼容的错误,以及是否存在活锁、死锁的状态。上述两种方式一个针对业务流程,另一个针对程序及其流程,可以完成对软件系统的整体测试与验证。

目前,传统的测试方法还未能完全提供自动化模型测试与验证,为此,我们课题组针对业务流程与程序流程两方面开展了初步研究:1)在业务流程模型的自动化检测与验证方面,使用 BPMN 流程建模工具建立 BP 模型,利用优化验证算法构造了统一建模平台——Eclipse 平台,并生成可执行 Java 代码来完成模型验证,可检测出业务流程是否存在死锁或活锁

本文受山西省自然科学基金资助。

薛 艳(1991-),女,硕士生,主要研究方向为编译器相关技术及应用、嵌入式开发、AI,E-mail:617989449@qq.com;**武淑红**(1969-),女,博士,副教授,主要研究方向为计算机应用、音视频编码算法、金融信息系统设计以及 SoC 与嵌入式系统软、硬件技术研究;**王耀力**(1965-),男,博士,副教授,主要研究方向为信息系统设计、人机视觉分析与处理、嵌入式系统电路设计理论。

等；2)对系统模型即代码及其流程的检测进行研究，实现系统模型的验证。业务模型的验证已经基本完成，本文将对程序系统模型进行验证，并针对 G 语言提出验证的技术方法。

1　系统模型的验证

SPIN (Simple Promela Interpreter)可对系统模型进行检测，确保系统中不存在死锁、活锁及 LTL 逻辑错误。SPIN 是一种用来分析和验证并发系统逻辑是否一致的辅助验证器，主要针对软件进行检测，而不是验证硬件是否能够高效运行。从 1980 年至今，SPIN 不断结合更先进的理论方法对其进行扩充和完善，主要涉及的理论支撑有：基于 Partial-order reduction 基础上的静态归纳技术 STREM、内嵌算法、软件验证思想、Statement Merging 技术、Property-base Slicing 技术及深度优先搜索算法等。目前，SPIN 已经成为开源的工具。

SPIN 首先从系统模型规格的描述开始，只有检测出其不存在语法错误才进行系统的交互模拟，直到确定系统设计达到了事先要求的行为。最后，SPIN 会生成一个 on-the-fly 的优化验证程序，并由检测器进行编译后再采用。如果在验证中出现了违反正确性的任何一个反例，就必须退回到交互模拟状态再次进行仔细检测，以找到不支持的原因。

把 Promela 作为系统的归约语言，它只会给有关的进程行为建模并采用 SPIN 验证，不关心与进程交互无关的其他元素。Promela 语言包括信息通道、进程和变量。其中进程用来记录系统行为，而信息通道与变量用来描述进程执行时的环境。以进程为单位，用进程异步组合方式进行建模，而对进程的同步则需要额外进行声明。SPIN 检测器的基本数据结构主要包括：状态矢量、Seen state set 和 Depth-first stack。

SPIN 一般使用 on-the-fly 机制建立自动机模型，SPIN 检测器能为每一个进程模板提供一个 Bochi 自动机，然后利用计算自动机异步交错积来获知并发系统的整体行为，具体方法是：1)最先通过 LTL 公示描述的系统性质选取反 Bochi 的自动机 A；2)根据计算机系统的每一个进程所转移的子系统数量的乘积，计算系统的整体行为，以搭建 Bochi 自动机 P；3)计算自动机 A 与 P 的乘积；4)依次类推得到最后的自动机，验证它是否能接收到语言，如果接收的语言为空，则满足系统定义的属性要求，反之则不满足。SPIN 验证的具体流程如图 1 所示。

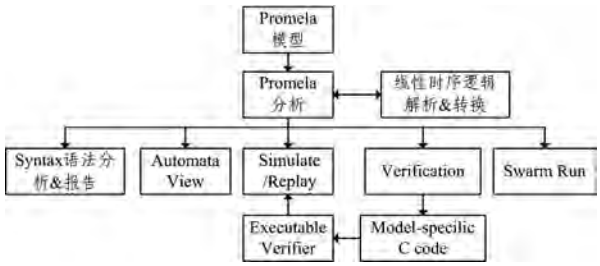

图 1　SPIN 的验证流程

2　G 语言现状

G 语言是一种程序化框图编程语言，是由美国 NI 公司开发设计的。相比于其他高级语言，G 语言是基于数据流驱动的而不是基于过程驱动的；在使用时，基本上不编写程序代码，而是使用流程图进行原理的研究、设计、测试并实现仪器系统，容易掌握且可视化。但是，它也存在一些缺陷：首先，G 语言还未列入 ANSI 标准，且由 NI 公司掌握，因此没有通用的验证工具，无法从程序层面进行模型验证；其次，G 语言的框图排列并没有形成一定的规范，使得程序框图间的连线杂乱无章，图形化界面查找错误太困难，程序难以阅读和维护；然后，利用 G 语言设计实现的产品与人类生活密切相关，且涉及医疗、核能、航空、航天等关键领域，例如 Sepios 水下机器人、DAO 机器人、完全自主型仿人机器人等，必须确保万无一失；最后，本课题组已经在 NI 平台上使用 G 语言编写程序多年，如餐厅服务机器人项目，它需要与人交流，最低要求是不能因为程序原因而使机器人失控而伤人，故需要进行系统软件包的关键应用升级。因此，利用本文提出的 G 语言模型验证技术，不仅可以弥补 G 模型验证工具的空白，还可以保障产品的可靠性、稳定性。

SPIN 并不支持 G 语言直接进行系统模型的验证，因此需要将 G 语言转化为 Promela 模型完成模型验证。目前，除了 G 语言的开发环境 Labview 外，并没有其他的环境可以读取并执行该程序，因此利用 Labview 提供的非开源生成器可生成 G 语言的文本代码，以便其他工具进行读入处理。在 Labview 开发环境中，根据 G 语言开发环境提供的接口将其转为文本语言，其框架如表 1 所列。

表 1　G 文本代码框架

方法内容	功能
全局变量声明及初始化；	VI Front diagram，Block diagram 中所有涉及的框图接口
void _TEXT_SECTION * * _CleanupLSRs	清理未初始化的左移位寄存器，防止内存泄漏
void _TEXT_SECTION * * _AddSubVIInstanceData	添加子 VI 实例化数据到全局列表中
void _TEXT_SECTION * * _AddVIGlobalConstants	分配 VI 常数
void _TEXT_SECTION * * _InitVIConstantList	VI 常数初始化函数
extern eRunStatus * * _Run ()	该 VI 主要的功能
eRunStatus * * _Start()	该方法作为 VI 的一个主要入口
void VIName()	调用库函数的接口

3　G2Promela 的设计与实现

3.1　G 语言预处理

目前，G 语言在智能机器人领域的应用已经非常广泛，但是 G 语言还未列入 ANSI 规范，因此还没有统一的形式、语法规范。经过大量的 G 语言编程总结，G 语言有以下规则：首先，G 语言的关键字有：int8，int16，int32，int64，float32，float64，uInt8，uInt16，uInt32，uInt64，TextPtr，VoidHand，PDAArrPtr，_DATA_SECTION，return，ArrDimSize，struct，void，_TEXT_SECTION，DataType，floatDataType，doubleDataType，stringDataType，static，eRunStatus，extern，ArrayDataType，false，true，new，for，if，while，case，switch 等；其次，G 语言的文法格式为：

start：module [module]*
　Start：Modules
　Modules：Module Modules | $
　Module：TypeSet

```
    | TypeSingles
    | FunctionDecl
    | FunctionImpl
TypeSet:struct Name '{' VarDecls '}' _DATA_SECTION Name ';'
  | struct Name '{' VarDecls '}' ';'
TypeSingles:TypeSingle
FunctionDecl:TypeNames Name '(' Parameter_list ')' ';'
FunctionImpl:TypeNames Name '(' Parameter_list ')' '{' Sequence '}'
Name:Alpha Iden
VarDecls:VarDecls VarDecl | $
……
```

然后，由于 G 语言程序的信息传递是由数据驱动而非过程驱动的，而且它的结构有前面板和程序框图面板两部分，即同一变量分布在不同位置却指向同一内存空间，因此所有的变量都会声明指向该变量的指针从而进行值的传递和引用；最后，G 语言程序中的方法分为两种：1）主框架中的方法，如 * * _CleanupLSRs()，* * _Run()，* * _InitVIConstantList()等；2）被调用的方法，但是没有具体的引用实现，需要将其封装为静态库函数进行动态调用或者每次进行动态编程生成。

Promela 是一种过程建模语言，其用于验证并行系统的逻辑，它的关键字有 active，assert，atomic，bit，bool，break，byte，chan，d_step，D_proctype，do，else，empty，enabled，fi，full，goto，hidden，if，inline，init，int，len，mtype，empty，never，nfull，od，of，pc_value，printf，priority，prototype，provided，run，short，skip，timeout，typedef，unless，unsigned，xr，xs。通过对 G 语言的预处理，形成与 Promela 的映射表，如表 2 所列。

表 2 G 语言与 Promela 模型的映射表

G 语言	Promela 模型
G 源代码主框架方法	active proctype
公共执行模块	init{} 一直处于活动状态的进程
if. 条件选择	if…fi
while 或者 do…while 循环	do…od
for 循环	for(参数不同)
else 分支	else
原子序列模块	atomic {sequence}
int8，int16，int32，boolean，PDAArrPtr，ArrDimSize	bit，bool，byte，short，int，unsigned
struct 变量	typedef 或者 c_decl { }
数组 array 或保留	通道 chan
float32，float64	c_decl[…] {…}
指针变量	c_decl[…]{…}
switch…case	if…fi
源程序中被调方法，如 PDAFltUnop()	inline(宏的程式化版本)或 c_code[…]{…}
局部变量和全局变量直接使用	局部‘Pproctypename－＞’；全局‘now.’
参数传递机制：值传递和引用传递	通道 chan 传递
方法直接调用	inline 声明的直接调用；proctype 声明的用 run 调用

3.2 模型验证框架

基于 SPIN 进行模型验证时，必须以 Promela 作为输入才可进行随机或迭代模拟来验证模型的正确性，因此必须对 G 语言程序完成系统建模。Promela 程序由进程、消息通道和变量组成。进程是表示分布式系统的并发实体的全局对象，消息通道和变量可以在一个进程中全局或本地声明，进程指定行为，通道和全局变量定义进程运行的环境。从整体结构来看，分为两个部分：处理阶段和生成阶段。处理阶段主要通过 Scanner 和 Parser 进行词法和语法分析，并创建符号表；生成阶段工作和转换器类似，在部分语义分析的基础上，完成从 G 语言中提取 Promela 模型的系统转换。模型验证的基本框架如图 2 所示。

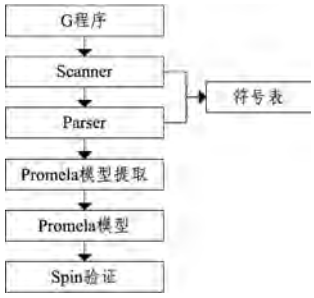

图 2 模型验证框架

本文的主体工作涉及到编译器的部分设计，主要有以下特色：1）对 G 语言的规范化处理，从实践中获取其规则及语言形式，有依据性；2）文中使用有限自动机(FAT)算法实现对源代码词法的分析和预处理，依据状态节点的起始和终止的特殊字符，初步生成符号表，并分类处理；3）采用下推自动机(PDA)进行语法规则的分析和处理，通过分析综合回溯和左递归问题，选择空间占用小、向前一位搜索的 LALR 方法来完成；4）在 PROMELA 模型中，将通过消息通道的通信定义为同步或异步，使用 SPIN 模型检查器来分析 PROMELA 模型，以验证模型化系统是否产生了所需的行为。在模拟和验证期间，SPIN 检查是否存在死锁、未指定的接收和不可执行代码。另外，进行 G2Promela 模型结构设计时，对比了所有环节的选择，从而做出了最优的选择，使得整体性能优良、执行稳定且可靠。

3.3 模型提取关键技术

相比于常用的 C，C＋＋及 Java，Promela 语言的语法结构非常简单，内部不涉及指针，for，if 及 while 3 种经典结构的表示形式多样化。但是它的细节要求特别多，如对方法的定义、方法的调用、局部变量和全局变量的使用、块语句的表示等，都有着自己独特的规定。针对这些特点，提出采用精巧的结构或者语义等价转化的方式实现。另外，针对不同的影响因素及不同的使用环境采用不同的转换方法。文中通过对 G 语言主框架及语言特点的分类说明了模型提取的关键技术方法。

1）关键字

在 Promela 模型中，仅有 bool，break，do，else，int，false，true 及 if 与 G 语言中的关键字相同，还有一些如 bit，byte，short，mtype，chan 及 unsigned 与 G 语言中的 int8，int16，enum8，PDAArrPtr 及 VoidHand 类似。对于 float32，float64 则直接用 int 类型表示；struct 及完全没有声明的类型用 typedef 来重新定义，剩余的根据实际含义具体对应。部分关键字的转换可以直接实现，少部分需要利用 Promela 提供的嵌套 C 代码的方法需进行折衷处理，如图 3 所示。

```
struct _xy_heap {
int16 n_y;
int16 n_x;
}_DATA_SECTION __xy_heap;
Boolean bRunToFinish=true;
int32 nReady=1;
```

G 语言

```
typedef _xy_heap {
int n_y_1;
int n_x_2;
}
Bool bRunToFinish_3=true;
int nReady_4=1;
```

Promela 模型

图 3　关键字模型处理

2)变量

G 语言中的变量主要涉及 static 变量和指针变量两种，且都为全局形式。static 变量可以被几个线程使用，修改变量可能会影响多个线程的行为，导致控制流图中的更改。例如，两个或多个线程可以在同一时刻通过限定访问表达式测试变量的值来进行控制决策。基于上述原因，我们选择将方法外的变量建模为 PROMELA 中的全局变量。为了避免源代码变量之间的名称冲突，根据以下约定命名 proctypeName_VariableName，proctypeName 为进程的名字。另外，由于原语中定义的变量不能够被 promela 本体中的程序调用，而 promela 本体中定义的变量可以被原语中定义的变量调用，因此需要以上述规则为基准具体情况具体对待。

一般情况下，还可以将变量分为局部变量和全局变量。在 Promela 中，局部变量的调用通过‘P＋proctypename－＞varName’实现，而全局变量的调用通过‘now. varName ’实现。这种规则和大多数语言对变量的调用都不同，一旦逆向执行在 Syntax check 阶段就会有错误报告，模型验证的下一步工作将无法执行，具体实例如图 4 所示。

```
static int32 n=0;void xy_Run()
{
  int32 a;
  a= n;
}
```

G 语言

```
int n=0;
active proctype p_xy_Run ( ) {
pid id4 ;
int p_xy_Run_a;
c_code{Pp_xy_Run－＞p_xy_Run_a=now. n;}
}
```

Promela 模型

图 4　变量模型处理

3)方法函数

在 G 语言中，函数主要分为两种类型：1)主框架中的 7 个函数，这些方法由 G 语言程序功能的复杂性决定是否具有具体的实现；2)源程序中被调用的函数，这类方法的主要特点是没有具体实现细节，需要动态编程实现。在对进程进行建模时，Spin 将新创建的进程保留在堆栈结构中，它不允许进程退出，直到其生成的所有进程终止。由于本文的目的是对 G 源程序的系统模型进行验证，因此在转化过程中应最大程度地保持程序的原型。为实现这一目标，在主框架中拥有具体功能实现语句的方法会被转换为 active proctype 过程类型，若没有则不进行转换，而第二种方法会以内联的方式转换。函数内联技术包括插入被调用函数的代码，而不是使用函数调用运行时机制，这是一种优化技术，可以避免基于运行时栈的函数调用机制所产生的开销。具体实例如图 5 所示。

```
void _TEXT_SECTION xy_AddVIGlobalConstants(void)
{ }
void _TEXT_SECTION xy_InitVIConstantList(void) { }
extern eRunStatus xy_Run()
{
Boolean bRunToFinish=true;
int32 nReady=1;
AddDoublexy_InitVIConstantList();
{
PDAFltBinop(heap－＞n_x,doubleDataType,heap－＞n_y,doubleDataType,opPlus,heap－＞n_Add_x_y,doubleDataType);
}
}
```

G 语言

通道形式

```
chan ret=[10] of {int ,int ,byte}
chan res=[10] of {int}
inline PDAFltBinop(){
int a; int b; byte c; int d; ret? x,y,op; ……. res ! z;
}
active proctype p_AddDoublexy_Run ( ){
int a_3; int b_4; byte c_5='＋'; int d_6;
ret ! a_3,b_4,c_5; PDAFltBinop(); res ? D_6;
}
```

Promela 模型

```
active proctype xy_AddVIGlobalConstants ( ){
pid id2 ;
}
active proctype p_xy_InitVIConstantList ( ){
pid id3 ;
}
inline PDAFltBinop(a,b,c,d)
{
  if
    ::(c ==‘＋’)－＞ d=a+b;
    ::(c ==‘－’) －＞ d=a－b;
  fi
}
active proctype p_AddDoublexy_Run ( ){
bool bRunToFinish_1=true ;
int nReady_2=1 ;
run p_AddDoublexy_InitVIConstantList ( ) ;
{
```

一般形式

```
  int a_3;
  int b_4;
  byte c_5;
  int d_6;
  PDAFltBinop(a_3,b_4,c_5,d_6);
}
}
```

Promela 模型

图 5　函数模型的处理

处理被调用函数模型时，需要注意对其参数的处理（见图 3、图 4），主要有两种方式：1)与一般语言类似的处理形式，即值传递；2)利用通道的 chan 的形式，利用 ！和？进行信息的发送和接收，类似于引用传递。

4)指针

在 G 语言中，因为同一变量可分别位于前面板和程序框

图内，即它们操作同一内存区域，所以这些变量都是用指针来标识的，方便值的传递。而 Promela 模型中，由于没有指针类型，因此在模型提取时，需要用原语进行 G2Promela 的转换。

如果 G 语言的某一语句涉及指针，则选择用 c_code[..]{.. }将语句嵌入；如果 G 语言的某一变量定义是指针，则选择 c_decl[..]{.. }完成模型转换；如果表达式中有指针，则选择 c_expr[..]{.. }来实现。总之，G 源代码中的指针都使用原语嵌套来实现模型提取。

5)基本结构(3 种基本结构)

G 语言中，基本结构主要有 do…while，for，if…else 及 swicth…case 4 种，而 Promela 中主要有 for(varref:expr…expr)，for(varref in array)，for(varref in channel)，do::sequence [::sequence] od，if::sequence [::sequence]* fi，else 及 unless{} 7 种结构。它们之间的映射都以关键字为基准，具体方法为：for 一般选择 promela 中的 for，具体类型由 G 语言中 for 循环的参数决定。若是范围类型，则以第一种 for 为主，若是集合类型，则以第二种或第三种为主。do…while 或 while 一般与 Promela 中的 do…od 对应，有时需要结合 unless 和 break 进行循环结束判断；if…else 和 switch…case 则选择 Promela 中的 if…fi，多分支选择时可与 else 及 skip 联立来增加分支的路径。上述 3 种结构的模型转换中，都需要至少一个临时变量来存储结构起始值，具体实例如图 6 所示。

```
for (i_1=0;(i_1 < l_Constant)
&& !gAppStop && !gLastError;
(i_1)++)
{
    {…}
}
if(condition1){…}
else if(condition2){…}
else{ … }
do{ … } while(condition);
```

G 语言

```
int temp_i ;
for(temp_i:0…L_Constant)
{ … }
if
  ::condition1->…
  ::condition2->…
  ::condition3->…
fi
do
  ::… ->
od unless {condition};
```

Promela 模型

图 6 结构模型的部分转换

4 实验结果

目前，对业务模型的验证可采用 BPMN 实现，对模型的验证方法则利用 SPIN 模型验证工具完成，使得对系统的验证得到全方位的解决。文中提出的方法已经可以提取简单 G 语言程序的 Promela 模型，结果展示以简易计算器为例。

在进行 syntax check 检查时，没有任何错误报告；在 Automata View 阶段中，利用 Dot 可生成模型的主进程的流程，如图 7 所示。

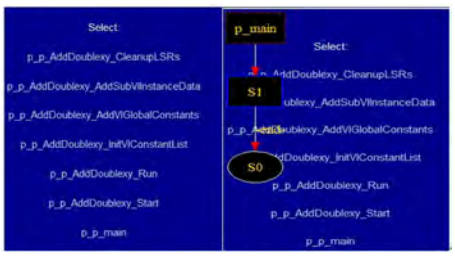

图 7 Dot 流程提取结果

在 Simulate/Replay 阶段，其结果如图 8 所示。

图 8 仿真结果

在 Verification 验证阶段，其结果如表 3 所列。

表 3 验证结果

Statevector	280
Depth reached	48
Transitions	10745
Actual memory	1.335
Total memory	65.575
Deadlock	NO
Elapsed-time	0.011

表 3 中的验证结果基于 Spin 工具验证的结果，其中 State vector 的单位是 byte，而 Actual memory 和 Total memory 的单位是 MB。从结果可以得出，提出的 G2Promela 方法是有效的，可以通过 G 语言到 Promela 的转换实现对 G 语言系统模型的验证。

随着智能技术的快速发展，G 语言在智能机器人领域的应用越来越广泛，但 G 语言缺乏系统模型验证工具所带来的不安全感也越来越引发人们的担忧，本课题的研究正好可以减轻人们在这方面的忧虑。

结束语 目前，计算机软件技术的发展已经离不开模型验证方法的保障。随着越来越多的智能产品走入人们的日常生活，其为人们提供服务的同时使得人类被伤害的可能性提高，模型验证的提出可以从根本上消除产品因忽略的编程错误或系统错误导致的危害。Spin 工具目前可支持 Promela 和 C 语言；对于 G 语言，由于其本身的独特性，还未有针对其的验证工具，本课题的提出弥补了这一空白，而且还为 G 语言实现的系统增强了安全性、稳定性、可靠性及高效性。

本文提出的技术可以实现对简单 G 语言程序模型的提取，但是对于复杂的 G 语言程序，由于模板未全部分类处理(如信号处理模块)，还未能实现完整的验证。因此，下一步工作是利用 G 语言的其他模板设计程序，基于上述方法利用 SPIN 完成对其系统模型的验证。通过不断的实践，在源程序中进一步完善，使得 SPIN 的验证范围逐步扩大，从而达到最初预设的目标。

参 考 文 献

[1] CATTEL T. Modeling and Verification of sC++ Applications [C]// Proceedings of the Tools and Algorithms for the Construction and Analysis of Systems. Lisbon, Portugal, LNCS, 1384.

[2] KOREL B. Automated software test data generation[J]. IEEE Transactions on Software Engineering, 1990, 16(8): 870-879.

(下转第 544 页)

表 1 MLC 闪存的仿真参数

闪存结构	参数
Block	32Pages/Block
Page	17260Cells/Page
NAND 闪存类型	MLC(2bits/Cell)

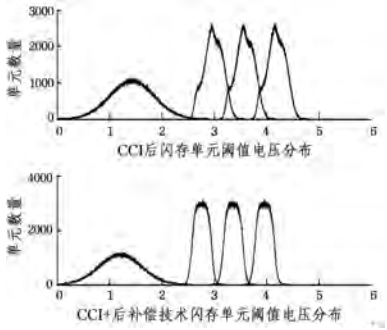

图 6 $s=0.8$ 时闪存模型阈值电压分布直方图统计

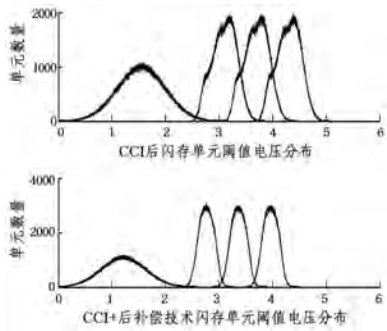

图 7 $s=1.4$ 时闪存模型阈值电压分布直方图统计

图 8 给出了 MLC 闪存单元阈值电压在均衡化前后的误比特率 BER 的性能比较，均衡化算法可以有效降低闪存单元中的原始比特错误率。随着耦合强度因子 s 的减小，CCI 干扰强度变弱，与 RBER 相比，均衡化算法的性能得到提升。

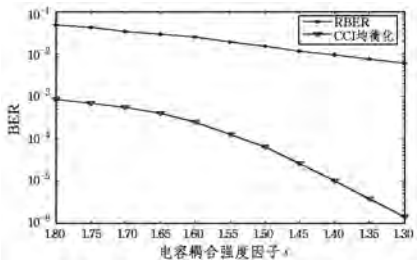

图 8 MLC 闪存单元在均衡化前后的 BER 性能

结束语 针对 CCI 成为影响 MLC 型 NAND 闪存可靠性的主要因素，提出了一种 MLC 闪存的 CCI 噪声均衡化算法。该算法对感知后的 MLC 闪存单元阈值电压进行有效补偿，提高了 MLC 闪存单元阈值电压的准确度。CCI 均衡化补偿算法的使用，可以有效减少相邻状态的阈值电压交叉现象，有助于降低原始比特错误率，提高 MLC 闪存的可靠性。

参 考 文 献

[1] TAKEUCHI K. NAND flash application and solution[J]. IEEE Solid-State Circuits Magazine, 2013, 5(4): 34-40.

[2] DEAN K. The history of semiconductor memory: from magnetic tape to NAND flash memory[J]. IEEE Solid-State Circuits Magazine, 2016, 8(2): 16-22.

[3] LIU R, CHUANG M, YANG C, et al. Improving read performance of NAND flash SSDs by exploiting error locality[J]. IEEE Transactions on Computers, 2016, 64(4): 1090-1102.

[4] SALA F, IMMINK K, DOLECEK L. Error control schemes for modern flash memories solutions for flash deficiencies[J]. IEEE Consumer Electronics Magazine, 2015, 4(1): 66-73.

[5] DONG G, XIE N, ZHANG T. On the use of soft-decision error-correction codes in NAND flash memories[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2011, 58(2): 429-439.

[6] CHAUDHRY A, GUANG Y, CAI K. Detector for MLC NAND flash memory using neighbor-a-priori information[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2016, 24(9): 2827-2836.

[7] DONG G, LI S, ZHANG T. Using data postcompensation and predistortion to tolerate cell-to-cell interference in MLC NAND flash memory[J]. IEEE Transactions on Circuits and Systems I: RegularPapers, 2010, 57(10): 2718-2728.

[8] PARNELL T, DÜNNER C, MITTELHOLZER T, et al. Capacity of the MLC NAND flash channel[J]. IEEE Journal on Selected Areas in Communications, 2016, 34(9): 2354-2365.

[9] XU Q, GONG P, CHEN T, et al. Modelling and characterization of NAND flash memory channels[J]. Measurement, 2015, 70: 225-231.

[10] CHAUDHRY A, GUANG Y, CAI K. Read and write voltage signal optimization for multi-level-cell (MLC) NAND flash memory[J]. IEEE Transactions on Communications, 2016, 64(4): 1613-1623.

[11] SUH K, SUH B, LIM Y, et al. A 3.3V 32Mb NAND flash memory with incremental step pulseprogramming scheme[J]. IEEE Journal of Solid-State Circuits, 1995, 30(11): 1149-1156.

[12] PARK K, KANG M, KIM D, et al. A zeroing cell-to-cell interference page architecture with temporary LSB storing and parallel MSB program scheme for MLC NAND flash memories[J]. IEEE Journal of Solid-State Circuits, 2008, 43(4): 919-928.

(上接第 540 页)

[3] HOLTZMAN G. The SPIN Model Checker, Primer and Reference Manual[M]. Addison-Wesley Professional, 2001.

[4] CHAN W, ANDERSON R J, BEAME P, et al. Model checking large software specifications[J]. IEEE Transactions on Software Engineering, 1998, 24(7): 498-520.

[5] HOLZMANN G J. The model checker SPIN[J]. IEEE Transactions on Software Engineering, 1997, 23(5): 279-295.

[6] HOLZMANN G J, SMITH M H. A practical method for verifying event-driven software[C]// Proceedings of the 21st International Conference on Software Engineering. ACM, 1999: 597-607.

[7] 王大伟，张大方，缪力. 一种自动化模型检测 ANSI-C 程序的实用方法[J]. 计算机工程与科学，2010，32(4)：79-82.

[8] 郭伟，缪力，张大方，等. 基于 Spin 的 UML 状态图模型检查的设计与实现[J]. 计算机工程与应用，2008，44(10)：43-47.

[9] 高晓星，李晓霞，薛冰. 基于 UML 和 SPIN 的软件安全模型验证[J]. 长沙大学学报，2013，27(5)：69-71.

[10] 舒良春，饶俊，肖美华，等. 基于 Promela 的 UML 建模方法及其应用[J]. 计算机与现代化，2010，2010(2)：101-104.