文章编号：0253-374X(2018)11-1575-07 DOI:10.11908/j.issn.0253-374x.2018.11.016

基于有界模型检测的门级软件自测试方法

张 颖[1]，张嘉琦[1]，王 真[2]，江建慧[1]
(1. 同济大学 软件学院，上海 201804；2. 上海电力学院 计算机科学与技术学院，上海 200090)

摘要：提出了基于有界模型检测的门级软件自测试方法.将处理器中的模块简化成约束模块，缓解状态爆炸问题.将难测故障的触发条件逐个转化成性质并且采用有界模型检测技术，搜索触发这些性质的违例.最后，将违例映射成测试指令序列，并为测试指令序列添加观测指令序列，构成自测试程序.实验结果表明：该方法在不引起状态爆炸问题的情况下，有效地测试控制器中难以测试的故障，提高了在线测试的测试质量.

关键词：基于软件的自测试(SBST)；模型检测；抽象
中图分类号：TP302.8 **文献标志码**：A

Gate-level Software-based Self-testing Method Based on Bounded Model Checking

ZHANG Ying[1], *ZHANG Jiaqi*[1], *WANG Zhen*[2], *JIANG Jianhui*[1]
(1. School of Software Engineering, Tongji University, Shanghai 201804, China; 2. School of Computer Science and Technology, Shanghai University of Electric Power, Shanghai 200090, China)

Abstract: A gate-level software-based self-testing method based on bounded model checking is proposed in this paper. The module in processor is abstracted and simplified into a constrained module to alleviate the state explosion problem. Then, the trigger conditions for unpredictable faults are transformed into properties one by one, and the bounded model checking is used to search violations which trigger these properties. Finally, the violation is mapped into the sequence of test instructions, and a sequence of observation instructions is added to form a self-test program. The experimental results show that the method can effectively test the faults which are difficult to be tested in the controller but without causing the state explosion problem, and improve the test quality of online test.

Key words: software-based self-testing (SBST); model checking; abstraction

由于处理器主频的不断升级而外部测试仪的速度提升缓慢且造价昂贵，因此直接使用外部测试仪对处理器进行实速测试变得越来越困难.一方面，处理器的很多故障只有在实速条件下才能被激活；另一方面，用户有强烈的处理器现场测试需求.因此，迫切需要一种在线测试方法能够实速测试处理器.

研究人员提出了针对处理器的多种实速测试方法，如内建自测试(BIST)方法[1-2].然而，BIST方法导致大量性能和面积开销，同时BIST电路的功耗较大，可能会损伤处理器电路.随后研究人员提出了基于软件的自测试(SBST)方法[3-4].通过运行正常程序测试处理器自身的结构和功能故障，在不引入任何硬件开销的前提下，完成处理器的在线测试.SBST方法是非侵入式测试方法，避免了过度测试，而且可以在现场条件下进行实速测试.然而，处理器的控制模块是复杂的时序电路，现有的SBST方法难以直接采用测试向量自动生成(ATPG)技术产生时序电路的测试向量.控制模块是处理器的关键模块，研究新的指令级自测试方法来有效检测测试控制模块中的难测故障是非常必要的.

本文采用有界模型检测(BMC)方法[5-6]，针对处理器的控制器中难测故障，在门级电路上直接产生测试程序.

1 相关工作介绍

1.1 基于软件的自测试方法

SBST方法通过运行程序直接检测处理器中功能与结构故障.这种方法不需要引入任何硬件开销，就能够在线测试处理器，并且达到理想的故障覆盖率，已经成为一种非常理想的测试方法.

如图1所示，SBST方法可以分为以下3步：① 下载测试程序与测试数据到片上系统；② 处理器执

收稿日期：2017-12-29
基金项目：国家自然科学基金(61432017,61404092)
第一作者：张 颖(1984—)，男，副教授，工学博士，主要研究方向为集成电路容错与测试. E-mail:zhangying@ict.ac.cn

行测试程序,将测试向量施加到被测模块上,激活被测故障;③ 测试响应被保存在存储器中,并上传到外界用于观察.

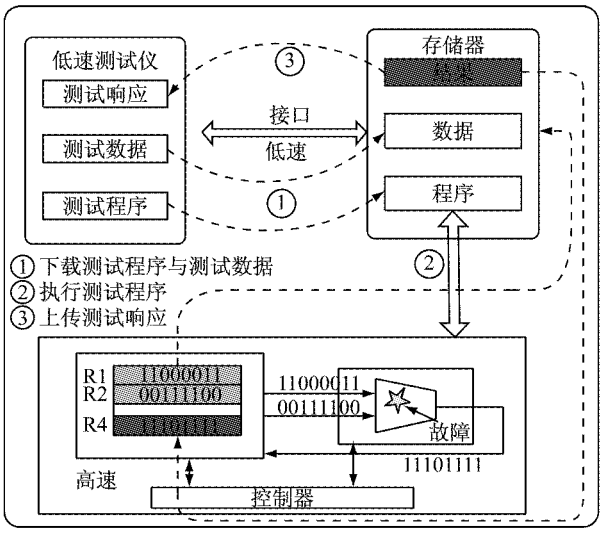

图1 处理器的基于软件的自测试方法示意图[7]

Fig.1 Framework of SBST method on processors[7]

近年来,SBST技术得到迅速发展,有学者使用测试指令自动生成(ATIG)方法自动产生SBST程序,取得了较好的效果[8-12].然而,处理器中的控制模块结构复杂,而且是时序电路,目前除了使用侵入式扫描链测试方法,其他方法均难以有效测试模块故障.侵入式扫描链测试方法无法在线测试,需要大量额外的测试开销,因此本文提出一种可以更加有效测试控制模块的软件自测试方法.

1.2 有界模型检测

BMC[5-6]是一种很重要的自动验证技术,它不仅可以自动验证有穷状态系统中命题的正确性,一旦命题有错,它还能够提供不满足命题的违例.因此,采用BMC方法能够在控制器的门级电路上,直接获得触发难测故障性质的测试序列.由于BMC方法穷尽地搜索了所有测试序列,因此能够控制故障难以触发的问题,进而在控制器上达到良好的测试效果.

控制模块受处理器中其他模块的约束,仅在控制模块上运行BMC,无法产生满足现场测试要求的测试序列.然而,将整个处理器输入到BMC工具中,由于处理器规模较大,会遭遇状态爆炸问题.本文拟将处理器的其他模块抽象成简单的功能约束,缓解BMC的状态爆炸问题.

1.3 测试指令自动生成方法

ATIG[9]是一种新型测试技术,它能够自动生成处理器的自测试程序.通常基于寄存器传输级(RTL)软件的自测试方法需要人工生成测试程序,而ATIG方法首次实现了测试程序的自动生成.首先,通过数据挖掘获得扩展指令和信号之间的映射,并将指令中数据的范围映射到信号上,作为指令级约束;其次,将基于指令级的约束转化为虚拟电路并连接到被测电路上,然后自动生成测试向量;最后,根据测试向量中控制信号的值和映射关系,将测试向量转化为测试指令,形成测试程序.ATIG方法尚未提出针对处理器中时序控制单元的测试生成方法,这种控制模块的故障覆盖率仍有待提高.

2 基于BMC方法的门级软件自测试方法

本文提出了基于BMC方法的门级软件自测试(GB-SBST)方法.该方法能够针对控制模块中的难测故障产生测试程序,有效测试时序的控制模块,进一步提升现有ATIG方法的测试效果.

具体而言,本文方法专门针对ATIG方法未检测到的故障,采用BMC方法来获取能触发该故障的指令序列,最后通过观测指令序列得到测试响应.图2展示了本文方法的流程.对于所有未检测的故障逐个产生测试程序,包括以下3个主要步骤:① 模型化被测电路,并采用抽象的方法进行简化;② 将选中的故障转化成性质,运行BMC,搜索激活该性质

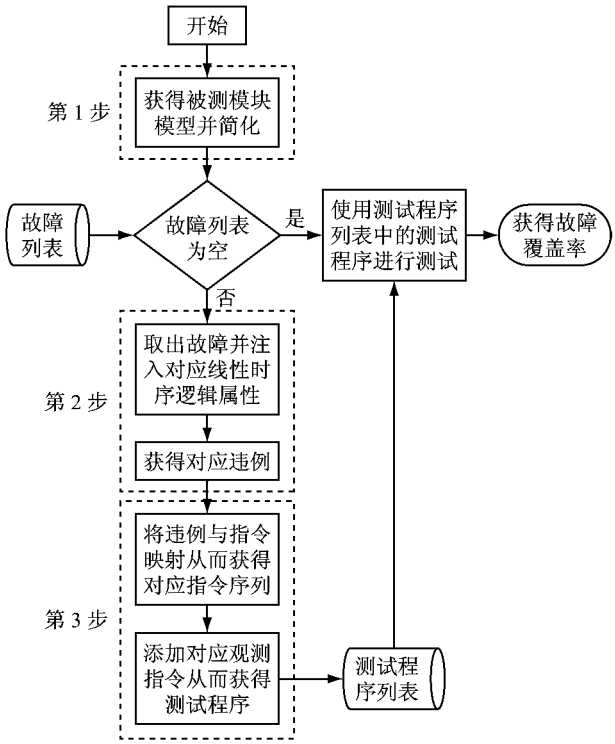

图2 基于有界模型检测的门级软件自测试方法流程

Fig.2 Flow chart of GB-SBST method

的违例；③ 若获得违例，则将违例中输入信号序列映射成测试指令序列，最后添加观测指令序列，从而生成指令级自测试程序.

2.1　模型的获得与简化

目前使用的 BMC 工具 NuSMV[13] 只能处理采用 smv 语言[14]编写的设计，被测电路的门级电路需要采用 smv 语言建模描述. 由于门级电路中的组合门对应 smv 语言中的线变量，时序的寄存器对应 smv 语言中的状态变量，因此编写了一个转化程序，将被测电路的门级电路文件直接转化成 smv 语言描述的文件.

被测模块在处理器内部，它受到众多其他模块的影响，单独的被测模块无法模拟其在正常模式下的功能. 然而，如果将处理器中的所有模块载入 BMC 工具，那么状态变量众多，BMC 求解过程会遭遇状态爆炸问题. 因此，需要抽取被测模块的功能约束，采用抽象技术来简化其他模块.

数值抽象[15]是一种针对数据通路中数值之间关系而提出的抽象技术. 在测试处理器的时候，假如某个寄存器的值始终与另外一个寄存器相等，这种关系是恒定不变的，那么这两个寄存器就需要抽象为一个寄存器. 抽象后的系统规模必定小于原系统的规模，这样就缩减了模型规模，降低了 BMC 的复杂度. 通常，抽象是由给定实际值集作用到抽象域的映射关系达到. 然后，将这个映射关系作用到状态集以及变迁关系集，进而再生成一个映射关系，但是规模相对于原系统更小，从而降低了 BMC 的难度.

具体而言，为了缩减被测模块的规模，整个系统按功能被分成若干功能模块. 系统中模块之间并不是相对独立的，而是相互协作的，模块上游输出的取值范围就是该模块的输入约束，模块下游能够接受的输入取值范围就是该模块的输出约束. 一旦获得这些约束，待测模块就被转化为约束模块，如图 3 所示. 约束模块的作用就是控制待测模块的输入与输出，以及替代其他模块的功能，因此其他模块就抽象为约束模块. 抽象模块与待测模块共同组成了一个功能完备的系统，它等价于原系统.

2.2　难测故障的违例抽取激活

在获得约束模块后，需要针对 BMC 产生的性质，激活被测电路中的难测故障. 如图 2 中第 2 步所示，首先获得被测电路的难测故障，判断这个故障集是否为空. 如果故障集不空，则逐个从故障集中取出难测故障，编写激活基于线性时序逻辑（LTL）的性质. 然后，运行BMC工具验证这条性质. 如果在给定

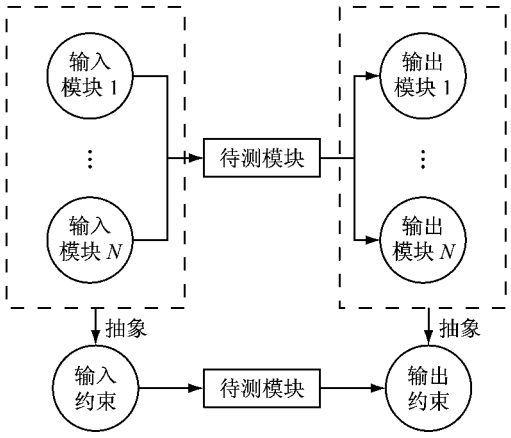

图 3　约束模块提取

Fig. 3　Extraction of constraint component

的时间片内，BMC 工具未能搜索到任何违例，那么性质恒成立，难测故障在给定时间片内无法被激活；如果性质不成立，那么 BMC 工具将给出一个违例，违反这条性质.

具体而言，针对被测电路中难测的固定型故障，将激活故障使能条件 P 恒为否作为验证性质，搜索激活故障的测试序列. 例如，图 4 展示了一个时序电路，假设该电路的输出端口存在一个固定为 0 的故障. 为了激活这个故障，需要搜索一个输入序列，使得输出值变成 1，这就是故障的激活条件 P. 只需要将激活条件一直为否（$G(!\ P)$）作为性质，与被测电路的模型同时输入 BMC 工具，BMC 工具将搜索激活该故障的输入序列. 如果在给定时间片内，BMC 工具未能发现任何违例，该性质恒为真，就不存在激活该故障的测试序列，该故障不可测试；如果 BMC 工具搜索到一个违例使得性质为假（见图 5），就满足故障激活条件. 只需将违例中输入序列施加到被测电路输入端，就可以测试给定故障.

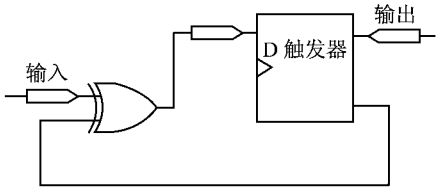

图 4　一个时序电路

Fig. 4　A sequential circuit

2.3　自测试程序构建

SBST 方法测试故障的步骤包括触发故障和观测故障导致的响应，因此激活难测故障的违例需要转化为测试该故障的指令序列，而后添加观测程序形成最终的自测试程序. 具体而言，在获得违例后，

```
NuSMV > check_ltlspec_bmc -p 'G(output = TRUE)'
-- no counterexample found with bound 0
-- specification  G output = TRUE    is false
-- as demonstrated by the following execution sequence
Trace Description: BMC Counterexample
Trace Type: Counterexample
-> State: 11.1 <-
  cp = TRUE
  d = FALSE
  q = TRUE
  input = FALSE
  output = TRUE
-> State: 11.2 <-
  cp = FALSE
  d = TRUE
  q = FALSE
  output = FALSE
```

图5 违例

Fig.5 A counterexample

抽取违例中的输入序列,然后将输入序列映射为测试程序,如图2所示.映射过程包括两部分:指令映射和状态位设置.指令映射是根据指令类型和指令控制码的对应关系,将违例中的输入序列转化为测试指令或者测试指令序列.状态位设置需要通过指令与操作数协同来达到,即指令执行前一个时间片中设置操作数,通过指令利用操作数进行运算,将状态位设为给定的值.例如,对于某违例的输入序列,根据指令类型和指令控制码之间的映射关系,该输入序列被转化为由2条指令构成的测试序列.在第1条指令前,添加addi指令以及合适的操作数,为第1条指令初始化状态位的值.同时,为第1条指令准备合适的操作数,使得这条指令能够初始化第2条测试指令状态位的值,形成一个测试指令序列.然而,这种指令序列只能触发难测故障,还不能确保激活该故障的响应可以被观测.因此,需要为测试指令序列添加相应的观测指令序列,组成完整的自测试程序.针对难测故障集中的故障,重复上述步骤,逐个产生自测试程序,最终产生GB-SBST程序.

3 GB-SBST方法对Parwan处理器控制模块的测试

与处理器中的功能模块相比,控制模块逻辑复杂,并具有一定的时序深度,很多难测的故障只有特定的测试向量序列才能激活,随机向量难以确保这些特定的测试向量序列的覆盖.表1展示多种现有测试方法在Parwan处理器上的故障覆盖率.虽然基于随机向量的SBST方法[16]和ATIG方法都测试了Parwan处理器中超过90%的故障,但是它们在控制模块上的故障覆盖率都显著低于全扫描链方法所获得的全处理器故障覆盖率.这表明基于随机向量的SBST方法和ATIG方法难以有效地测试复杂的控制模块,关键的控制模块需要新型的自测试技术.本文拟将GB-SBST方法应用于Parwan处理器的控制模块,测试难测故障.

表1 现有方法在Parwan处理器上的故障覆盖率

Tab.1 Fault coverage of the existing methods

项目	各方法故障覆盖率/%		
	全扫描链方法	基于随机向量的SBST方法	ATIG方法
控制模块	95.1	79.0	84.4
全处理器	97.9	92.1	94.8

3.1 Parwan处理器

图6展示了本方法中Parwan处理器的架构.Parwan处理器规模较大,包含了8个主要模块,分别为累加器(AC)、计算单元(ALU)、移位单元(SHU)、状态寄存器(SR)、指令寄存器(IR)、程序计数器(PC)、地址生成单元(MAR)和控制模块(CTRL).控制模块为其他模块提供控制信号,而且只接收指令寄存器提供的指令,以及状态寄存器提供的状态信息.

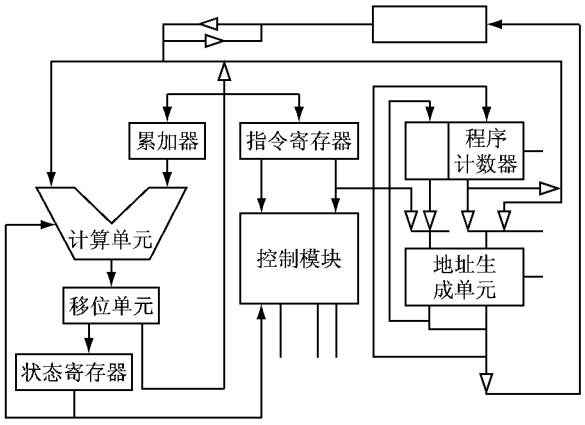

图6 Parwan处理器架构

Fig.6 Architecture of Parwan processor

3.2 控制模块的模型化与精简

采用门级网表描述的设计需要转化为smv语言描述的设计,才能够被BMC工具处理.使用Java开发并完成了一个读取门级网表逻辑并转化为smv文件的程序,使用该程序将Parwan控制模块的门级网表转化为smv文件.同时,为了避免全处理器进行模型检测时的状态爆炸问题,采用抽象技术,使用处理器中其他模块施加到控制器的约束来替代它们本身的功能.

首先,对Parwan控制模块的输入进行抽象仿真,控制模块的输入信号来源共有2个,分别是IR与SR,如图6所示.SR的输出为四位CPU状态位:V表示溢出位,C表示进位,Z表示零位,N表示负数位.根据这些状态位的意义,可以得到这些状态位

之间的约束关系. 表 2 显示状态位之间的约束关系，也是 SR 对于控制模块四位输入的约束情况. 当 Z 为 1 的时候，N 一定为 0，且 V 与 C 的值相等；当 Z 等于 0 时，V 的值等于 C 与 N 的异或，或者 V、C 和 N 的值都等于 0. 总而言之，这些约束可以由布尔等式来表示，如下所示：

$$!V*!Z*N+!V*!C*N+V*C*!N+V*!C*!Z*N=1 \quad (1)$$

对于每一个状态位的约束，也可以由以下 4 组布尔等式表示：

$$V_{out}=V*C*!N+V*!C*!Z*N \quad (2)$$

$$C_{out}=V*C*!N+!V*C*!Z*N \quad (3)$$

$$Z_{out}=V*C*Z*!N+!V*!C*Z*!N \quad (4)$$

$$N_{out}=!V*!Z*N+V*!C*!Z*N \quad (5)$$

表 2　SR 状态位的约束关系

Tab.2　Constraints for SR state signals

约束	V	C	Z	N
约束 1	无关位	$C=V$	1	0
约束 2	$V=C\oplus N$	无关位	0	无关位
约束 3	0	0	0	0

根据每个状态位的约束，设计约束电路，表示在处理器正常工作模式下，SR 对控制模块施加的影响. 将等式(2)～(5)转化为约束电路的门级网表文件，进一步转化为 smv 文件，施加到控制模块 smv 文件的四位输入状态位上.

基于同样方法将 IR 的约束转化为对应的约束电路，并使用 smv 文件表示其约束逻辑，其输出就作为控制模块的来自 IR 的输入信号，为控制模块提供合法指令，完成对应的功能. 因此，IR 的约束电路将约束控制模块的输入信号，确保每次输入都是合法指令. 随后，将 SR 和 IR 的约束电路一起施加到控制模块上. 这样，在进行模型检测的时候，控制模块的所有输入信号值符合原始电路实际输入，控制模块在抽象后的模型中是正常工作的，约束模块等效替换了处理器的其他模块. 由于约束模块的规模远小于处理器中其他模块的规模，抽象技术确实成功缩减了被测模型的状态数量. 同时，由于有界模型求解问题的计算开销随状态数量增大呈指数型增长，采用抽象技术约简模型能够明显减少测试时间，降低测试成本.

3.3　难测故障对应的违例获得

在构造和简化被测模块后，需要针对控制模块中的难测故障采用 BMC 工具逐个搜索触发序列. 本文假设 ATIG 方法测试 Parwan 处理器时，控制模块中未测试到的故障是难测故障. 图 7 展示了触发控制模块中难测故障的测试程序的示例. 首先，根据未检测到故障列表的信息，获得故障对应的门单元、端口信息以及故障类型. 假设控制单元中逻辑门 U10 的 A0 端口的连接线 n19 存在固定为 1 的难测故障，这个故障的激活条件 P 是导线 n19 的信号变为 0. 其次，将故障激活条件转化成 LTL 性质，采用 BMC 工具搜索存在违反性质的违例. 具体而言，将 A0 端口不可能一直等于 1(AG!（n19 == 0))作为性质. 如果在给定的时间片内，BMC 工具无法找到性质的违例，即导线 n19 的信号恒等于 1，难测故障在正常工作模式下不可测试；如果有界模型搜索到违反性质的违例，即违例中的输入序列使得导线 n19 的信号变成 0，那么故障被激活，违例中的输入序列就是测试该故障的测试序列. 通过 BMC 工具检测，n19 固定为 1 故障是可测试的.

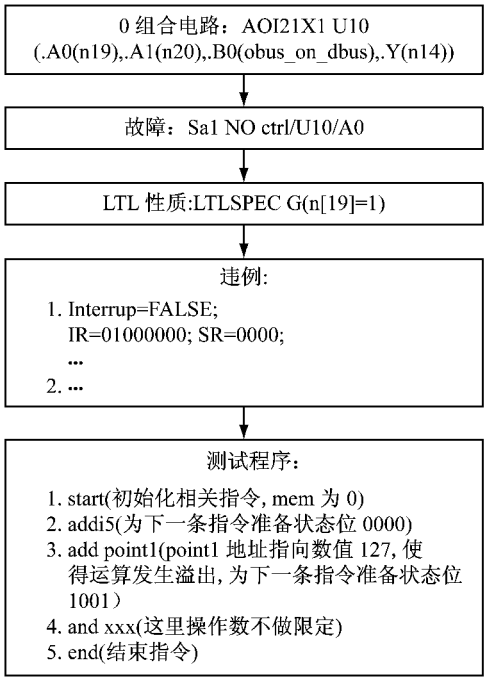

图 7　获得违例的过程

Fig.7　Process of counterexample generation

基于同样方法可以测试控制模块的所有难测故障. 表 3 展示了针对难测故障的 BMC 工具运行结果，其中控制模块有 40 个难测故障，BMC 工具成功地产生了 32 个故障的违例，但是有 8 个故障不存在激活它们的违例. 这是因为在控制模块的状态机中，这些信号始终保持固定值，它们是不可测试的故障.

3.4　基于违例产生的自测试程序

触发难测故障的违例需要转化为测试指令序列，才能执行基于软件的自测试，在线测试控制模块中的难测故障. 根据第2.3节，转化过程包括指令映

表3 针对难测故障的BMC工具运行结果

Tab.3 BMC results for hard-to-test faults

难测故障数	获得违例的故障数	未获得违例的故障数
40	32	8

射和状态映射. 首先,指令映射是指根据控制码与指令类型之间的对应关系,将违例中输入序列转化为指令序列. 在Parwan处理器中,控制模块直接从IR中获取指令,而且IR约束模块的输出等价于IR的输出,那么违例中IR约束模块的输出就是控制模块的测试指令. 这样,违例中的输入序列可以转化为指令序列,定义为T:[(Inst1,S1),(Inst2,S2),…],其中S1,S2,…是每条测试指令的状态信号. 其次,添加控制指令和相应的操作数,为下一条指令设置状态信号. 具体而言,首先增加一条指令和操作数,设置第1条指令的状态信号;然后,为第1条指令添加操作数,通过第1条指令的运行,为第2条指令设置状态信号;最后,继续采用这种方式设置指令的状态信号,最终得到测试指令序列.

由图7可见:在第1个时间片中,违例中IR约束模块的输出指令是add,SR约束模块的输出状态信息$(V,C,Z,N)==0000$;在第2个时间片中,违例中输出指令是and,输出的状态信息是1001. 为了触发n19固定为0的故障,需要在第1个时间片之前添加一条指令与操作数,设置指令add的状态信号. 添加指令"addi 5"就可以将add指令的状态位变为0000. 接着,为第1条指令add准备操作数,通过add指令的加运算,使得ALU运算发生溢出,并且数值变为负,状态位变为1001,符合第2条测试指令的状态信息. 这样就得到了满足要求的测试指令序列. 最后,本文方法在测试指令序列后添加观测指令序列,输出测试结果与相关符号位,形成难测故障的自测试程序.

4 实验设计与结果分析

实验采用商业ATPG工具进行故障模拟,获得故障覆盖率. 图8显示了实验的主要步骤. 首先,触发指令序列与观测指令序列组成自测试程序,并且与测试数据一起编译为内存文件. 然后,将内存文件加载到Parwan处理器上,在modelsim工具中进行模拟,并且保存模拟过程中处理器输入输出端口值的变化. 最后,将值变化文件和Parwan处理器加载到TetraMAX进行故障模拟,并且评估故障覆盖率.

本文方法主要针对ATIG方法中难测故障,提

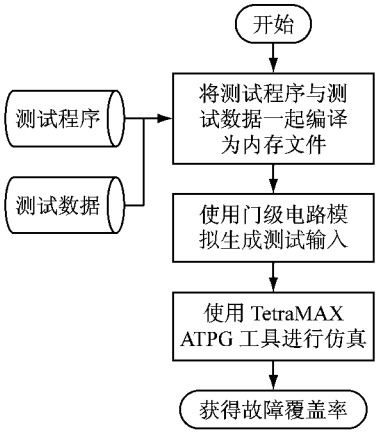

图8 实验过程

Fig.8 Experiment process

升现有自测试程序在核心控制模块中的测试性能,改进后的自测试方法被称为ATIG+GB-SBST. 表4显示了不同方法在Parwan处理器上的故障覆盖率结果. 从表4可知,ATIG+GB-SBST方法确实提升了现有自测试方法的测试性能,Parwan处理器的故障覆盖率达到了95.5%,控制模块的故障覆盖率从84.4%提升到了88.9%,解决了28.8%的难测故障. 控制模块中尚未检测到的故障主要是工作模式下的不可测故障,产生这些故障的原因有以下几个方面:一方面,处理器的某些导线,在正常工作模式下恒为0(或者恒为1),不存在激活这些变量上固定为0(或固定为1)故障的测试序列,如表3所示;另一方面,控制模块中某些导线无法影响任何的输出信号,进而它们的测试响应无法观测.

表5显示了4种不同测试方法的测试开销. ATIG+GB-SBST方法的全部测试程序包括310条指令,以及775 Byte的数据. ATIG+GB-SBST方法只需要指令121条,测试数据258 Byte. ATIG+GB-SBST方法的总数据量比基于随机向量的SBST方法的数据量节省了36.2%,也少于全扫描链方法的测试数据量. 虽然基于随机向量的SBST方法、ATIG方法和ATIG+GB-SBST方法的测试时钟周期数大于全扫描链方法,但是这三种方法的测试时间不会超过全扫描链方法的测试时间. 这是因为这三种方法采用高速的工作时钟,而全扫描链方法采用低速的测试仪时钟. ATIG+GB-SBST方法仅比ATIG方法少量增加了测试时间,但是相比于基于随机向量的SBST方法来说依然减少了87.3%,在测试开销方面提升明显. ATIG+GB-SBST方法也不需要任何额外的面积开销,更不需要采用昂贵的外部测试仪.

表 4　不同测试方法的故障覆盖率

Tab.4　Fault coverage of different testing methods

项目	不同方法故障覆盖率/%			
	全扫描链方法	基于随机向量的 SBST 方法	ATIG 方法	ATIG+GB-SBST 方法
控制模块	95.1	79.0	84.4	88.9
全处理器	97.9	92.1	94.8	95.5

表 5　不同测试方法的测试开销

Tab.5　Test cost of different testing methods

方法	指令数/条	数据/Byte	时钟周期	空间占用/%
全扫描链	0	846	6 957	6.1
基于随机向量的 SBST	569	1 214	121 105	0
ATIG	189	517	9 413	0
ATIG +GB-SBST	310	775	15 433	0

5　结语

处理器中核心的控制模块由于结构复杂，并且时序深度较大，现有方法难以有效进行测试. 本文设计了一种基于 BMC 的门级自测试方法，有效地测试了控制模块中的难测故障. 这种方法利用抽象技术，将控制模块的输入约束替代处理器中的其他模块，从而简化了被测处理器. 然后，将难测故障转化为 LTL 性质，利用 BMC 工具搜索触发难测故障的输入序列. 最后，将这些违例中的输入序列转化为测试序列，并添加控制和观测指令，形成自测试程序. 实验结果表明，本文方法提高了控制模块中难测故障的故障覆盖率，提升了现有自测试方法对核心控制模块的测试性能.

参考文献：

[1] LU J M, WU C W. Cost and benefit models for logic and memory BIST[C]// Proceedings of Design, Automation and Test in Europe Conference and Exhibition. Paris: IEEE, 2000: 710-714.

[2] MIRANDA J M. A BIST and boundary-scan economics framework[J]. IEEE Design & Test of Computers, 1997, 14(3): 17.

[3] RIEFERT A, CANTORO R, SAUER M, *et al*. A flexible framework for the automatic generation of SBST programs[J]. IEEE Transactions on VLSI Systems, 2016, 24(10): 3055.

[4] LI C, DEY S. Software-based self-testing methodology for processor cores[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2001, 20(3): 369.

[5] 侯刚，周宽久，勇嘉伟，等. 模型检测中状态爆炸问题研究综述[J]，计算机科学，2013(增 1)：77.
HOU Gang, ZHOU Kuanjiu, YONG Jiawei, *et al*. Survey of state explosion problem in model checking[J]. Computer Science, 2013(S1): 77.

[6] BRADLEY A R. SAT-based model checking without unrolling [C]// Proceedings of International Conference on Verification, Model Checking, and Abstract Interpretation. Austin: [s.n.], 2011: 70-87.

[7] PSARAKIS M, GIZOPOULOS D, SANCHEZ E, *et al*. Microprocessor software-based self-testing[J]. IEEE Design & Test of Computers, 2010, 27(3): 4.

[8] TUPURI R S, KRISHNAMACHARY A, ABRAHAM J A. Test generation for Gigahertz processors using an automatic functional constraint extractors [C]// Proceedings Design Automation Conference. New Orleans: IEEE, 1999: 647-652.

[9] ZHANG Ying, LI Huawei, LI Xiaowei. Software-based self-testing of processors using expanded instructions [C]// Proceedings of IEEE Asian Test Symposium. Shanghai: IEEE, 2010: 415-420.

[10] ZHANG Ying, LI Huawei, LI Xiaowei. Automatic test program generation using executing-trace-based constraint extraction for embedded processors[J]. IEEE Transactions on VLSI Systems, 2013, 21(7): 1220.

[11] ZHANG Ying, PENG Zebo, JIANG Jianhui, *et al*. Temperature-aware software-based self-testing for delay faults [C]// Proceedings of Design, Automation & Test in Europe Conference & Exhibition. Grenoble: [s.n.], 2015: 423-428.

[12] ZHANG Ying, CHAKRABARTY K, LI Huawei, *et al*. Software-based online self-testing of network-on-chip using bounded model checking [C]// Proceedings of IEEE International Test Conference. Fort Worth: IEEE, 2017: 1-10.

[13] CIMATTI A, CLARKE E, GIUNCHIGLIA F, *et al*. NUSMV: a new symbolic model checker [EB/OL]. [2017-12-01]. http://nusmv.fbk.eu/.

[14] ARCAINI P, GARGANTINI A, RICCOBENE E. A model advisor for NuSMV specifications[J]. Innovations in Systems and Software Engineering, 2011, 7(2): 97.

[15] WANG Jian, LI Huawei, LÜ Tao, *et al*. Abstraction-guided simulation using Markov analysis for functional verification[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016, 35(2): 285.

[16] MERENITIS A, THEODOROU G, GIORGARAS G. Directed random SBST generation for on-line testing of pipelined processors[C]// Proceedings of IEEE International On-Line Testing Symposium. Rhodes: IEEE, 2008: 273-279.