可重构系统的演化修复机制

朱继祥1) 李元香2) 邢建国3)

1)(浙江工商大学计算机与信息工程学院 杭州 310018)

2)(武汉大学软件工程国家重点实验室 武汉 430079)

摘 要 利用演化算法实现系统自修复是一种新的容错设计思路，但是演化是一个非常耗时的过程.已有的演化容错系统多属于静态演化，演化过程仅发生在系统设计阶段，系统在运行过程中不具有演化修复的能力.这类演化动态演化的修复机制，容错系统采用可重构系统和被检测系统的耦合设计方案.当被检测系统出现故障时，可重构系统通过系统演化实现在线自修复.为了减少演化耗时，系统根据错误类型采取不同措施：如果出现已知错误，系统直接在预置配置库中搜索修复配置；如果出现未知错误，则通过动态演化在线生成修复配置，并更新预置配置库.最后，将该容错设计方案用于典型电路的故障模式.实验结果表明，文中提出的演化修复机制提高了系统运行的实时可靠性，预置配置库设计减少了演化耗时.

关键词 演化硬件；演化容错；动态演化；故障模式；预置配置

中图法分类号 TP311　　　DOI号：10.3724/SP.J.1016.2014.1599

The EvoIvable Recovery of Reconfigurable System

ZHU Ji-Xiang1) LI Yuan-Xiang2) XING Jian-Guo3)

1)(School of Computer Science and In formation Engineering.Zhejiang Gongshang University，Hangzhou 310018)

2)(State Key Laboratory of Software Engineering，Wuhan University，Wuhan 430079)

**Abstract** Utilizing evolutionary algorithms to realize self-recovery is a new idea of fault tolerance，but evolution is time-consuming.Most evolvable fault-tolerant systems so far are off-situation evolution ，where the evolution process plays a role just in the system design stage.In this case，the system does not have capability of evolution recovery during its running time.Although this kind of evolvable fault-tolerant system overcomes the evolution time-consuming，it only repairs the anticipated errors rather than unanticipated errors，the fault-tolerant system is a coupling design of the reconfigurable system and the detected system..Once the target system fails，the reconfigurable system implements in-system evolution to achieve runtime recovery.To decrease the time overhead of evolution process，the proposed system takes different measures according to failure modes;if the anticipated errors are detected，it searches the pre-designed library for recovery configurations;otherwise，when the unanticipated errors happened，it generates new recovery configurations onlion by evolvable hardware，and updates the pre-designed library by the evolved configurations.At last，this fault-tolerant method is equipped in different failure modes of some typical circuits.The experiments illustrate that the proposed evolvable recovery approach promotes the reliability，and pre-designed configuration library decreases the evolution time overhead.

**Key words** evolvable hardware;evolvable fault tolerance;in-situation evolution;failure mode;pre-designed configuration

# 引言

传统的冗余容错技术需要较多的资源开销，而且只能修复已知错误.已知错误（Anticipated Errors）是指在系统设计阶段能够预见并可测试的错误，它们对电路的影响是可以预测的.在系统设计时注入这些错误并进行测试，然后通过冗余可以屏蔽故障[1].由于故障的诊断和定位存在困难，在系统设计阶段无法进行全面测试，系统在运行过程中更容易出现未知错误（Unanticipated Errors）.未知错误无法用若干个有限模型去描述，难以屏蔽或者修复.在这种情况下，系统即使有若干可用的冗余模块也无能为力.因此，很难设计出能够容忍未知错误的专用集成电路系统（Application Specific Integrated Circuit，ASIC）.

近年来，人们开始探索利用现场可编程逻辑门阵列（Field-Programmable Gate Array，FPGA)的可重构特性实现容错的新方法，如全局重构容错、局部重构容错[2-4].重构容错的特点是利用FPGA的在系统编程（In-System Programming，ISP）能力，通过备份电路的分时重构实现容错.这类容错技术本质上仍属于多模冗余结构，只是冗余资源是备份电路的配置信息，而非硬件逻辑.重构容错虽然避免了传统冗余容错资源开销大的问题，也能修复部分随机故障，但是芯片提供的备用电路是有限的，若出现任何备用电路都无法修复的错误，系统将会失效.另一种方案是故障模式和效应测试（Failure Modes and Effects Testing，FMET）.Greenwood[5-6]认为已知错误在设计阶段是可以预测的，可进行建模和测试，将其修复方案存储起来.当系统再次出现同样的错误时，只需从存储器中下载相应的电路配备即可实现修复.这两种方案的不同之处在于，前者的备份电路采用传统方法设计，后者则采用演化硬件（Evolvable HardWare，EHW）技术生成修复配置.

简单来讲，演化硬件是用演化算法对可编程器件实施重构，使系统在运行时不断改进自身的功能和结构[7].Greenwood引入演化机制的目的是为了找到最好的修复方案，演化过程发生在系统投入使用之前，属于静态演化（Off-Situation Evolution）.演化硬件的终极目标是在FPGA上集成基于演化算法的重配置机制，形成可演化的片上系统（Evolvable soC），使系统在运行的演化过程属于动态演化（In-Situation Evolution）[8].动态演化和ISP相结合使FPG具备了在系统演化（In-System Evolving，ISE）的能力，ISP使系统在运行时通过演化自主修复未知错误成为可能.

本文提出一种重构容错系统的演化修复机制；首先将演化硬件作为容错系统的辅助修复模块，然后通过演化算法对系统进行重构，从而实现在线自修复.为了减少演化耗时，在系统中建立预置配置库.修复已知错误只需查找预置配置库；修复未知错误则利用演化硬件技术在线生成修复配置，并更新预置配置库.

## 2 基于演化硬件的容错系统

## **2.1** 异构冗余容错系统

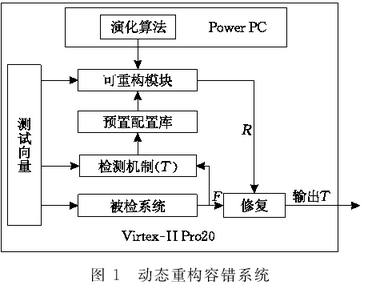
目前，基于演化硬件技术的容错系统多采用异构冗余方案，核心思想是利用演化算法设计出多个异构备份电路.异构冗余的关键是如何评价异构度，异构度越大，错误覆盖率越高，系统越可靠性.有关学者已围绕异构冗余进行了多方面研究：Schnier和Yao[9]用“无关性”（Negative Correlation）评价电路的异构程度，提高了冗余电路的可靠性；Shanthi等人[10]通过调整已演化电路的结构来提升系统的错误覆盖率；Hartmann和Haddow等人[11-12]演化设计容噪容错的异构数字电路.林勇等人[13]提出一种选择性进化冗余的方法，设计出比同构冗余更可靠的数字系统；高桂军和姚睿等人[14-16]采用其他异构评价函数也取得不错的效果.

研究表明，只要选择合理的异构评价函数，演化硬件便能设计出异构度大、错误覆盖率高的冗余系统.然而，异构冗余方案是通过在系统中注入若干确定的错误，然后采用静态演化，设计出具有容错能力的电路.异构冗余仍处于传统冗余容错的框架之内，只能覆盖可预测故障，不能修复随机故障.

2.2 动态重构容错系统

本文设计的动态重构容错系统如图1所示.它由监测机制、被监测系统、预置配置库、演化算法、可重构模块、修复模块等部分组成，演化硬件和被检系统通过修复模块耦合形成了整个动态重构容错系统.

可重构模块是利用FPGA（Virtex-II Pro20）的空余资源设计的虚拟可重构电路（Virtual Configurable Block，VCB），其配置信息是演化算法的染色体.可重构模块和演化算法一起构成了演化硬件.演化算法运行于Power PC，系统适时地启



动演化算法在线生成修复配置，可重构模块在演化过程中充当评估载体，演化结束后用于映射正确配置.

修复模块的两个输入信号分别来自于被检电路的输出信号F和可重构模块的输出信号R，整个系统的输出是F和R的耦合.选择何种运算（即耦合规则）视具体应用而定.

预置配置库专为修复已知错误而设计.为了减少不必要的演化，节省修复时间，系统修复已知错误时不启动演化硬件.通过对被检电路进行预测试，针对不同的故障设计相应的修复电路，并将其配置信息保存在片上存储器中.故障和修复电路之间是多对一的关系，一个修复配置可以修复多个故障.预置配置库可以是静态的或是动态的.

检测机制产生被检电路和演化硬件所需的测试输入信号.检测机制还需要一块存储区域，用于存放期望输出T及错误句柄.错误句柄是系统在测试阶段假设出现某一故障后被检电路的实际输出信号，系统将根据错误句柄查找预置配置库.

3 系统实现

3.1 错误检测机制

故障终会表现为输出信号的误差，通过检测被检电路的输出信号是否符合期望值可判断电路是否出错.本文通过加载测试输入向量，得到被检电路的实际输出信号，比对检测机制中的期望输出或者预设错误句柄，返回错误模式.错误模式分为3个级别，分别执行不同的操作：

（1）无错误，模式代码‘0’，不执行任何操作.

（2）发生已知错误，模式代码‘1’，查找预置配置库.

（3）发生未知错误，模式代码‘-1’，启动演化硬件.

已知错误是在系统设计阶段经过测试了的故障所引起的错误.虽然有些故障在理论上是可预测的，但要对所有可能的故障都进行测试是不现实的，我们把未经过测试的故障视为未知错误.换言之，不管是何种故障所引起的错误，只要实际输出信号已经存在于预置错误句柄中，就归为已知错误；反之，则为未知错误.

3.2 修复机制

系统根据错误代码采取不同的修复机制：对于已知错误，在预置配置库中查找能修复该错误的配置，并下载到可重构模块.对于未知错误，则启动演化算法，生成新的修复配置.未知错误的一个显著特点是具有不确定性，不同的未知错误会产生不同的错误句柄.演化算法根据错误句柄不断地更新目标函数，这种在运行时实施目标调整的演化过程被称为“动态演化”.可演化自修复机制即是基于这种动态特性实现的.

1. 确定演化目标

演化目标与修复模块所采用的耦合规则有关，结合数字电路的特点，我们选择异或运算作为耦合规则，即

T=F R.

异或规则具有可逆性，通过简单运算可得出演化目标：R=F T[17].显然，如果被检系统在不同时刻出现不同的错误，其输出F必然不同，则R也发生变化，演化算法则以R为目标实施动态演化.例如：假设被检系统的期望输出T=（0111 1101 1001 0101）.在T0时刻，F=（0011 1100 1011 0101），且预置配置库中无相应的修复配置，则根据异或规则得出R=（0100 0001 0010 0000）.为了保证整个系统T的正确性，演化算法以新的R为目标进行演化.

如果演化硬件最终成功搜索到能够实现功能R的电路配置，则将其下载至可重构模块，并通过修复模块与被检测系统进行耦合即可实现修复.否则修复失败，系统发出警告.

1. 编码及解码

电路染色体编码采用与笛卡尔遗传（Carte-

sian Genetic Programming，CGP）[18]类似的方法.它是一个节点矩阵，每个节点用一个函数编码（该节点的逻辑运算）和两个连接编码（分别表示参与逻辑运算的两个输入）来表示.染色体由连接编码和函数编码（图2下划线部分）两部分构成，节点的数量决定了染色体的长度.染色体的解码是通过将染色体下载至FPGA中完成的.为了将染色体在FPGA中映射成实际电路，我们按照上述编码规则，采用硬件描述语言（VHDL）设计相应的评估电路.该模块是一个虚拟可重构电路，它能够接受来自于PowerPC的染色体，并将其解码成实际电路等待测试和评估.

（3）适应值评估

通过“在线演化”（Intrinsic Evolution）方式计算适应值，演化耗时主要来自于评估时间.尽管评估单个个体的时耗为毫秒级，但演化过程中，所有染色体都需要映射到FPGA中进行评估，大量的映射和评估时间的累积是十分可观的，这不利于实时自修复.我们采用多路并行评估策略解决该问题，利用FPGA中的空余资源将虚拟可重构电路复制4份，它们分别接受不同的染色体，这4个电路构成了整个可重构模块（图一）.并行评估的好处是可同时解码和评估多个染色体，这样大量地减少了演化耗时.

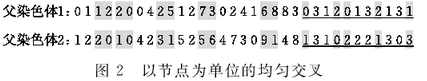
完成染色体解码后，测试向量发生器依次产生测试向量，每个评估电路都将得到一个输出向量Ci（i=1，2，3，4），它表示当前4个染色体所表示的实际电路功能.然后分别计算Ci与修复电路的期望输出R之间的海明距离d（R，Ci），这个值将返回PowerPC计算相应染色体的适应值.我们以当前电路的修复率作为染色体的评估函数[17]，用下式描述：



其中，e是可调因子，w(R)是R的重量.只有当d(R，Ci)<w(R)时，修复配置才是有效的（fi>e）.当d(R，Ci)=0，实现完全修复.

1. 演化算子

考虑到片子资源的限制和PowerPC的处理能力，本文选择简单遗传算法，选择算子为比例选择.由于编码阵列中存在冗余节点，采用均匀交叉算子时演化硬件找到修复配置的概率较大，且迭代次数较少.图2是均匀交叉的示意图，染色体中下划线部分是每个节点的函数编码（每个数字对应于函数集中的一个操作），前面部分表示连接编码（每个节点有连续两个连接编码），交叉算子是以节点为单位.



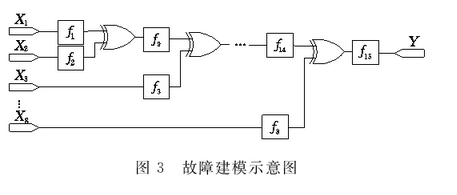
变异算子包括结构变异和函数变异，前者是对染色体的连接编码实施变异，随机地改变电路中某个（些）节点的输入连接；后者是对函数编码实施变异操作，从预先定义函数集中随机地选取另一个逻辑门替换发生变异的逻辑门.变异算子针对每个基因位，依变异概率发生变异.

4 实验与分析

我们进行两组实验：实验1以8位奇偶校验器为例，它由7个异或门依次串联而成，采用静态预置配置库.这种简单的实验方案是为了更清楚地描述故障建模和演化修复机制；实验2采用门级实现的四位乘法器 ，内部结构相对复杂，采用动态建库方案.通常一个故障不会引起所有输出端口的错误，可重构模块只对出错端口进行耦合运算.

4.1 故障建模

在描述被检电路时需要考虑故障建模.故障分为可预测故障和不可预测故障.可预测故障如晶体管损坏、连线短路或断开等等，建模为“固定故障”（Stuck-at Fault）.不可预测故障是由某些随机因素干扰引起的，无法 进行精准描述，建模为“随机翻转故障”[11-12]，即线路值按概率发生翻转.奇偶校验器和乘法器的故障建模类似，假定固定故障和翻转故障都出现在电路内部的连接线上.图3是注入故障模型后的奇偶校验器电路.



有15条连线，依次编号为f1，f2，f3，…，f14，f15，每条线路都有可能出现固定0故障（Stuck-at-0）、固定1故障（Stuck-at-1）或者随机翻转.被检电路总数为228—1个.显然，对所有的故障电路都进行测试是不切实际的.我们仅测试线路中出现的单个固定故障，并设计相应的修复电路，一共是30个，把它们放入预置配置库中.例如：f9出现固定0故障时，修复电路为X1 X2；f9出现固定1故障时，修复电路则为X1 X2.不同的故障组合可能导致不同的错误模式，当由多个故障所引起的错误和单个固定故障引起的错误一样，则为已知错误.如f1和f2分别发生固定0故障和固定1故障，和f9发生固定1故障所产生的错误相同，都可以用X1 X2修复，故为已知错误模式.只有在预置配置库中找不到修复电路时，系统才会启动演化修复机制.

4.2 实验平台及参数设置

实验平台选用的FPGA是XINLIX公司的Virtex-II Pro20（XC2VP20），PowerPC405硬核处理器，母版通过PCI与PC机相连，由SDK提供的接口函数（C语言描述）将运行结果返回PC.函数编码取{NOT，AND，OR，XOR，BUF，NAND，NOR，NXOR}，共8类基本逻辑门（依次编号0~7）.选用对称函数集的优势是可以利用组合逻辑的互补律来实施基因改良，改变适应值地形，加快演化速度.大种群规模会大幅增加待评估个体的数量，导致演化耗时急剧上升，难以满足实时性要求，本文选取种群规模PopSize=20.变异概率在0.1~0.25之间效果较好，采用自适应变异对演化效率并无显著提高，反而使计算量增加，故选择固定变异概率Pm=0.1.交叉概率在0.8~0.9之间时，平均迭代次数小于其它设定，故选择交叉概率Pc=0.8.终止准则：演化达到最大迭代次数（MaxGen=3000），或者找到能实现完全修复的新配置.

4.3 仿真实验

假定被检电路中的每条线路都依概率发生故障，线路故障的组合可能引起已知错误、未知错误，或者无错误.我们不预告系统将出现何种错误，仅给出若干个不同的故障率，并在系统中建立单个线路出错时的预置修复配置库.衡量容错性能的标准是可靠性和可用性，在文本中，可用性通过演化硬件搜索到有效配置所需的时间（<30s）来衡量；可靠性则是被检电路出错后找到有效修复配置的概率.

4.3.1 奇偶校验器

电路规模较小，我们依次设定5个高故障率：Pf=0.1、0.2、0.3、0.4、0.5.高故障率可使奇偶器尽量出现不同类型的错误，以描述演化修复机制的执行过程.系统生成15个随机数分别对应于被检电路中的15条线路.如果某条线路的随机数小于设定值，则该条线路发生故障，并随机选择一个故障类型.每个故障率分别独立运行100次，统计系统出错的次数（Nf）、完全修复次数（Nr）、平均迭代次数（G\_avg）以及平均演化修复时间（Tr）、系统可靠性（Re）.如表1所示，表中‘/’之前为已知错误的相关统计，‘/’之后为未知错误的相关统计.



根据表中第2行可得，故障率相对较低时，系统有少量机会不发生错误.如Pf=0.1时，发生已知错误64次，发生未知错误23次，另有13次未出错.随着故障率增加，系统检测到已知错误的次数减少，而未知错误的次数增加.在5种故障率下系统的错误率依次为87%，93%，96%，100%，奇偶器本身的可靠性几乎为0.系统在每个故障率下都出现了不同的错误，符合我们设定大故障率的初衷.

可靠性方面，比较Nf和N，演化机制修复了大多数的未知错误.如Pf=0.2时，出现未知错误32次，20次实现完全修复，仅有12次未能完全修复.高故障率使关键线路容易出错，增加了修复电路的复杂度，导致演化算法不能在规定迭代次数内找到符合条件的配置.实验发现，在Pf为0.4和0.5的情况下，修复失败大多是由于f14或者f15发生故障导致的.显然，增加迭代次数能够提高成功率，但同时会带来修复时间过长的问题.

可用性方面，我们统计了启动检测机制到实现自修复所需的时间.如果被检电路出现已知错误，其修复配置由查库所得，不必启动演化，故G\_avg=0，系统实现自修复所需的时间主要是检测耗时+查库时间，次过程耗时约12ms.发生未知错误时启动演化修复，G\_avg则是在30s内实现完全修复所需的平均时间，它主要是演化耗时，检测耗时和查库时间可忽略不计.演化耗时主要来自于评估时间，实验测得单个个体的在线评估时间约2ms.种群规模20，如果不采用并行评估，则每代种群仅评估时间就需40ms，显然不能满足实时性要求；若采用四路并行评估策略，每代种群的评估时间减少为10ms，此时大多数演化自修复都能在30s内完成，可以满足实时性的要求（见Tr）.进一步加大并行评估程度可满足更加苛刻的实时性要求，按同时需要更多的芯片资源.

4.3.2 四位乘法器

上述实验采用静态预置配置库，由演化硬件生成的修复配置没有保存.若系统在不同时刻出现相同的错误句柄，需要重复启动演化，造成较大的演

感悟部分

1、5页全部是手打，无复制粘贴，图是采用的截屏方式。

2、遇到的问题及解决方法：

1. 上标下标不会。上网百度了，明白了可以右击，选中字体，打开对话框后在上标下标那打对勾。
2. 式子中出现了上标的上标。百度后找到了合适的方法：调整字符位置。先把15和6同设为上标，然后单独选择6，右击选中字体，选字符间距，在下边设置位置为提升，确定即可，为使效果更好，可使6的字体大小变小。
3. 格式要求中出现了磅这个单位，查了

之后发现磅和字号是相对应的，比如5号对应

10.5磅，毫米是3.70。

1. 格式要求英文字体为Times New Roman，中文正文为宋体，因为打字过程中经常穿插英文，但一个一个的该会很麻烦，于是我就想能不能在查找替换中替换字体，将所有的英文字体由宋体替换成Times New Roman，在我的试验下，发现是可行的。在替换那一栏下，有格式下拉选项，选择字体，就行了。
2. 有的公式上面有横线。经查询，在插入一栏，点击形状中的线条即可。
3. 学会了如何分栏。

3、感悟：问题都是在练习中发现的，想要提升水平就要多多练习。