

基于 FPGA 实时错误检测技术

琚小明*, 张皆浩, 张逸中

(华东师范大学 软件学院, 上海 200062)

(*通信作者电子邮箱 xiju@sei.ecnu.edu.cn)

摘要:高可靠性的系统都要求具备实时错误检测。针对内建错误检测,提出了三种在线模型的自我实时检测方法。错误检测模型利用了现场可编程门阵列(FPGA)中的两个管道,通过比较当前配置信息与 FPGA 外配置内存中的原始信息是否一致,可以实时地检测错误,而且可以通过比较它们的配置数据来定位那些具有单粒子翻转(SEU)错误的逻辑块。仿真测试结果表明所提出的方法比在线 BIST 有着更好的性能。

关键词:错误检测;实时;可靠性;自检单元;现场可编程门阵列

中图分类号: TP202.1; TP306.3 **文献标志码:** A

Real-time error detection techniques based on FPGA

JU Xiaoming*, ZHANG Jiehao, ZHANG Yizhong

(Software Engineering Institute, East China Normal University, Shanghai 200062, China)

Abstract: Real-time error detections are needed in highly reliable systems. This paper presented three online models with self-checking method for built-in error detection. The error detection model adopted two pipes in the Field Programmable Gate Array (FPGA). By comparing whether the current configuration information and FPGA configuration memory of the original information were consistent, the model can detect errors in real-time, and by comparing their configuration data, it also can locate the error where logic blocks have undergone an Single Event Upset (SEU). The testing result shows that the method proposed in this article has better performance than that of online Built-In Self-Test unit (BIST).

Key words: error detection; real-time; reliability; Built-In Self-Test unit (BIST); Field Programmable Gate Array (FPGA)

0 引言

单粒子翻转(Single Event Upset, SEU)^[1]是引起现场可编程门阵列(Field Programmable Gate Array, FPGA)功能错误的最常见原因。在一些需要高可靠性的系统中,仅仅提高FPGA的可靠性是远远不够的。因为SEU是不可避免的,并且FPGA也没有配置自我检测能力。为了满足系统对于高可靠性的要求,错误检测和错误恢复都必须足够迅速。

本文提出了针对于FPGA的快速在线自我检测方法,可以迅速地检测出输出信号的错误。判断输出信号是否正确功能很重要,因为即使逻辑块出错,输出信号也可能不立即出错。另外,不同于其他仅仅提示错误的方法,该方法将能够定位错误并且指明出错的块。

1 相关工作

目前已经有许多检测方法,其中之一是三模冗余(Triple Mode Redundancy, TMR)^[2-3]。TMR由三个并行冗余部件构成,冗余结果的不一致表明产生了错误,但该方法不能够定位错误。

一种更加广泛使用的方法是在线自检单元(Built-In Self-Test unit, BIST)技术^[4-5],这种方法使用了叫作自我测试区

(sSelf-Testing Area, STAR)^[6-7]的特殊区域。在测试过程中,在线BIST将STAR中的块拷贝到备用行中,并且恢复连接。在检测中,检测矩阵作为块的输入,如果块的输出没有达到预期目标,错误将会被检测出来。但是,在这个过程中,检测所需的备用行必须被拷贝到备用区域中使FPGA可以继续工作。拷贝过程会消耗大量时间,在此期间STAR区域外的错误无法被检测。另一个问题是在线BIST需要FPGA供应商通常没有提供的部分配置。

另一种方法需要空余列和检测列两列^[8],这种方法基于在线BIST技术,但检测过程仍会耗费大量时间。

2 本文方法

2.1 基本思想

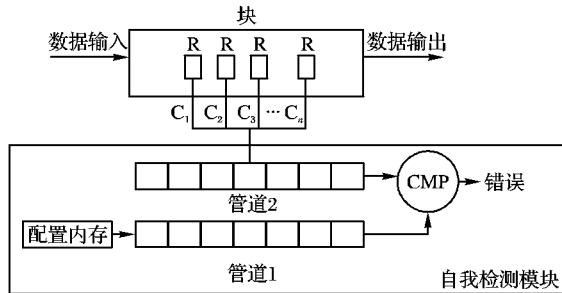
本文提出的错误检测模型借鉴了软硬件错误检测方法^[9-12],利用了FPGA中的两个管道,如图1所示,每个管道作为一个队列。进入管道1的数据来自存储FPGA配置数据的配置内存,进入管道2的数据来自特定时间内逻辑块的配置流。两个管道之间的任何不一致均表明信号出错,并能指出出错的逻辑块。

接下来的三种模型讨论了管道的三种不同实现方式。两个管道在一个模型中必须为相同的结构,两个管道输出数据

收稿日期:2012-11-27;修回日期:2013-01-11。 基金项目:上海高校知识服务平台资助项目(ZF1213);上海市高可信计算重点实验室开放课题(07dz22304201109);工业和信息化部电子信息产业发展基金资助项目。

作者简介:琚小明(1967-),男,浙江衢州人,副教授,博士,CCF会员,主要研究方向:嵌入式系统设计、软硬件协同设计方法、嵌入式系统可靠性;张皆浩(1992-),男,上海人,主要研究方向:软件工程、嵌入式技术;张逸中(1986-),男,上海人,硕士研究生,主要研究方向:软硬件协同设计方法、FPGA逆向工程。

的不一致表明错误的产生。模型 A 通过流水线结构设计管道,模型 B 利用异步先进先出(First In First Out, FIFO)来实现管道设计,模型 C 则是模型 A 和模型 B 的结合模型。



注: R表示块中的触发器; C表示块中的配置位

图1 用来错误检测两个管道

三种模型基于两个假设:一个假设是FPGA的错误仅仅由SEU导致,因为把FPGA中通常由电子迁移导致的传输错误作为一种老化问题,本文不考虑该问题。某单粒子翻转可以导致FPGA中的一些位翻转,一种简单的检测方法是比较当前配置信息与FPGA外配置内存中的原始信息是否一致。两个数据的任何不一致都表明逻辑块中出现了SEU,这就是为什么错误检测需要两个管道。

另一个假设是FPGA的信号流具有确定性,这也就是说一旦给出逻辑块的位置,可以通过跟踪来计算出何时输出信号会经过它。本文用一个叫作块树(block tree)的树结构来描述这种关系,它可以清楚地表示信号如何产生。连接两个逻辑块的有向箭头表示信号流向,如图2所示,很明显输出信号由E块产生,它的输入信号是块D和C的输出信号,块C的输入信号是块A和块B的输出信号。

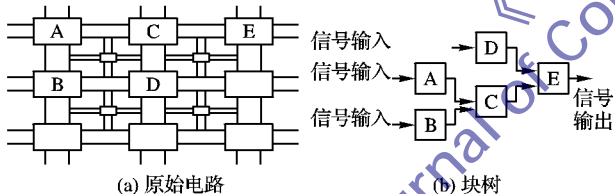


图2 块树的树结构

这种块树的优势一方面是一个块被破坏时,仍可判断出哪个输出信号是可用的。考虑B块被破坏的情况,假设B块在 t 时刻被破坏,每一级的块都会在每个周期 T 内产生一个信号。本文中, τ 时刻产生的信号用 e^τ 来表示。从块树中可以很明显地得出,在时间 $t \leq \tau \leq t + 2T$ 的所有信号仍是有效的。因此, $2T$ 是恢复错误的有效期。

本文规定距离信号输出最近的块具有最高优先级。从块树中可得出无序块优先级越低,允许恢复的时间越长,这对于计算错误恢复剩余时间具有重要意义。

2.2 模型 A

在模型 A 中,利用流水线来实现管道。第 i 个信号通过块 S 时的配置流用 Se_i 来表示。因此虽然 Se_i 和 Se_j 属于同一个块,但其配置数据可能不同,因为它们产生于不同时刻,这就是在流水线中相同块被重复记录的原因。

图3给出了每个寄存器组的操作描述:每个寄存器组负责对级逻辑块的持续追踪。同时,当一个时钟信号达到的时候,它将记录逻辑块特征的数据传递给下一个寄存器组。

图3(a)是流水线的结构,图3(b)、(c)、(d)分别是寄存器组1、2、3的数据。

最后的寄存器组记录流过的相同信号下的配置流,这些数据和另一个管道内的配置内存数据相比较来检测信号是否正确。

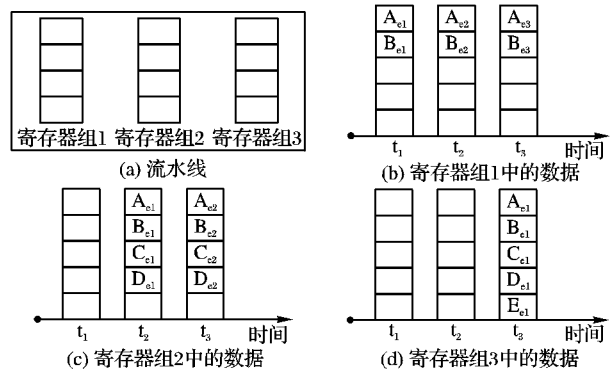


图3 模型A的内部结构

假设整个FPGA是一个工作区,每个逻辑块是一个员工。每个寄存器组是一个检测点来记录该点工人的状态,最后的寄存器组记录所有参与特定信号产生的工人状态。最后一个寄存器组的数据流和另一个管道中的配置内存数据相比较,如果结果不一致意味着当产生这个信号时工人处于低谷期,这就表明这个信号是错误的。

该模型的一个优点是所有逻辑块(工人)的状态和任何时刻产生的信号都可以被记录,就像飞机中的黑匣子。另一优点是即使是在块出错的情况下,它也可以准确地表明输出信号是否有效。这一特征也可以用工人的例子来解释:如果工人处于低谷期,它产生的下一个信号或许是错误的,但是之前的信号仍然有效。该特征允许在逻辑块出错时产生更多信号,这对于应用程序而言是合理的,就像高安全性系统的控制单元。模型A的缺点是它消耗了更多的FPGA资源,但是这仍适用于那些需要高可靠性却忽视所需硬件资源的昂贵设备。

2.3 模型 B

模型 B 中的管道利用了一个调整过的FIFO结构,与模型 A 的主要区别是所需的硬件资源减少,另一个不同是管道结构不同。

模型 B 充分利用了FPGA的时钟频率,利用锁相环(Phase-Locked Loop, PLL)和异步FIFO实现。PLL是FPGA中的电子线路,可以提供更快的时钟频率;异步FIFO能连接两个具有不同时钟频率的系统。

一个较快的时钟频率用来解决模型 A 中开销过大的问题,因为频率提高5倍,管道中每列的深度将被降低到1。

与模型 A 相比,模型 B 不被FIFO允许在其内部插入数据,其余的寄存器组则被用来保留同一数据流下的块内数据。

模型 B 的管道结构如图4(a)所示。在FIFO中连接寄存器组的实例负责记录产生相同信号的一组块的轨迹,需要寄存器组的原因是需要被比较的块在FIFO中是不连续的。寄存器组的内容与模型 A 中最后寄存器组的内容完全相同,它们被用来与配置内存的配置数据作比较。

图4(b)是流水线中的数据流。在 t_1 时刻,块 A_{e1} 和块 B_{e1} 与连续3个空数据排队进入FIFO。因为进入管道的数据流速度是逻辑块频率的5倍,所以看起来是5个数据一起移动的。在 t_3 时刻,FIFO中的数据被填满。然后 A_{e1} 、 B_{e1} 、 C_{e1} 、 D_{e1} 、 E_{e1} 进入寄存器组,并准备进行比较。

图4(c)是寄存器组的内容。在FIFO被装满之前的 t_1 和 t_2 时刻,寄存器是空的。在FIFO被装满之后,寄存器的内容变成同一信号流过的块的配置信息。例如,寄存器组在 t_3 时刻保留信号 e_1 经过的块的特征轨迹,在 t_4 时刻保留信号 e_2 经过的块的特征轨迹。

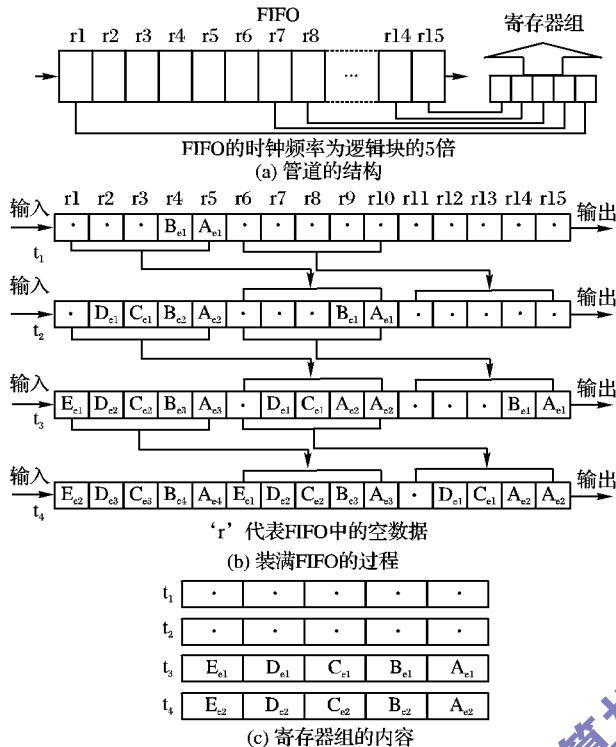


图4 模型B的内部结构

模块B的优势很明显,它在没有增加错误检测时间的基础上降低了模型A的高消耗。目前,Cyclone III FPGA PLLs能够很好地支持倍频,倍数范围是1~512。但是,FPGA频率有限制性,倍频不能过高,所以模型B在一些小型电路上应用性更强。

2.4 模型C

上文提出的模型A和模型B均可迅速检错,但是它们要么需要大量硬件资源,要么需要更高的时钟频率。在本节中则尝试着将两种模型的优势相结合来解决高消耗问题。

模型C和模型A相比,将列的深度降低到原来的 $1/(p+1)$,并且将频率提高 p 倍。参数 p 的值很重要,如果选择得当,FIFO的深度和时钟频率均可被接受。

考虑拥有三级块树的FPGA的电路(如图2)。起初,若按照模型A,流水线深度是6。如果流水线深度降低到原来的 $1/(2+1)$,同时将频率提高2倍,结果是流水线深度变为2,时钟频率是原来的3倍(如图5)。如此深度和频率均可以被现有FPGA所接受。

图5给出了该方法的内部结构。因为基础结构仍旧是FIFO,所以需要有一个寄存器组来记录产生相同信号的块组。模型C中FIFO的结构与模型B中的类似,只是FIFO的深度不同。实际上,模型B是模型C的特例,即将模型C中FIFO深度设为1即为模型B。

图5(a)给出了管道结构。FIFO的实体被划分成三组,当一个系统时钟到达时,一个组将移动到下一个组的位置,如图5(b)所示,寄存器组保留了相同信号流经的块的配置流记录。

图5(c)给出了寄存器组的内容。在FIFO被装满之前的 t_1 和 t_2 时刻,寄存器是空的;在FIFO被装满之后,寄存器的内容变成同一信号流过的块的配置信息。最后一个寄存器中的数据被用来和配置内存中的数据相比较以判断信号是否正确。

模型C的优势是可以在庞大的硬件资源和有限时钟频率中找到一个平衡点,可以被应用于一些有严格硬件资源要求的高可靠系统。

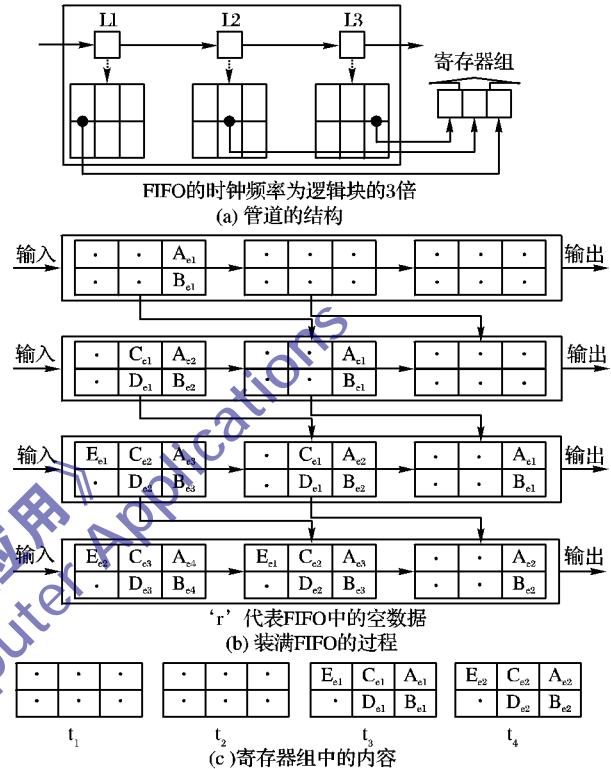


图5 模型C的内部结构

3 测试和仿真

3.1 测试环境目标

仿真环境基于Xilinx Chip XC3S400-4PQ208实现。

第一个测试主要是在设备利用率和吞吐量上来比较三种模型的性能,所测试的电路是一个 4×4 的可配置块。模型A,B,C的时钟间隔分别为80 ns,70 ns,80 ns,模型B和C的FIFO时钟频率分别为10 ns和20 ns。

在第二个测试中,仿真了一种错误检测最常见的模型——在线BIST模型,并与本文提出的模型B进行了比较。

在线BIST中测试下的行被叫作STAR。在STAR中的块由测试模式发生器(Test Pattern Generator,TPG)所产生的测试向量检测。当STAR中的块完成检测时,STAR移动到后续的行中来检测新STAR的行。由于它的机制原因,检测整个线路的时间和电路的范围有关。

在测试中,不同规模的线路用来检测本文模型B和在线BIST模型,并利用检测SEU的平均命中率来进行比较。平均命中率是指当出现SEU时成功地检测出一个SEU的概率。因为SEU是一个可能在检测到达错误点之前消失的软错误,所以发现一个已出现的SEU的可能性和被检测的线路规模有关。

3.2 测试结果和分析

表1~3列出了第一个测试中三种模型的资源利用率和

性能。表 1、2 给出了 FPGA 的设备利用率,可以看出,对于硬件资源的使用,模型 A 利用最多,模型 B 最少;模型 B 的 FIFO 频率最高,是系统时钟频率的 7 倍。测试结果表明,这三种模型均能迅速检测出输出信号的正确与否,但是三种模型的吞吐量有很大不同。如表 3 所示,模型 C 拥有最大的吞吐量,几乎是模型 A 和模型 B 的三倍之多。这说明模型 C 平衡了硬件资源和 FIFO 频率;另外,它也有着最大的吞吐量。

表 1 设备占用可配置逻辑模块

分类	模型 A	模型 B	模型 C
片型触发器数	1 821	1 666	1 838
四输入 LUT 数	1 538	1 521	1 513
芯片占用数	1 658	1 507	1 629
只包含相关逻辑的芯片数	1 658	1 507	1 629
四输入 LUTs 总数量	1 538	1 521	1 513
用作逻辑的数量	1 474	1 345	1 449
双端口 RAM 的数量	64	64	64

表 2 设备利用率 %

模型	片型触发器数	四输入 LUT 数	芯片占用数	只包含相关逻辑的芯片数	四输入 LUT 总数量
模型 A	25	21	46	100	21
模型 B	23	21	42	100	21
模型 C	25	21	45	100	21

表 3 设备性能

模型	吞吐量/MB	时钟频率/ns	FIFO 的时钟频率/ns
模型 A	1 576.9	80	—
模型 B	1 576.9	70	10
模型 C	4 731.8	80	20

表 4 列出了第二个测试的比较结果。从中可以看出,尽管本文提出的模型 B 比在线 BIST 利用了更多的资源,但在 SEU 的错误检测上,它比在线 BIST 有许多方面的优势。首先,本文模型 B 和在线 BIST 均可以在运行时间内检测错误,但在线 BIST 不能实时检测错误,因为只有 STAR 中的行可以被检测;其次,对于在线 BIST,检测 SEU 的平均命中率随着检测线路规模的上升而下降(从 46.9% 下降至 32.8%)。命中率和线路规模密切相关,因为在线 BIST 需要在所有块被检测之前连续移动 STAR。相反,本文提出的方法保证了 100% 的命中率,这意味着命中率不会受线路规模的影响。

表 4 模型 B 和在线 BIST 的对比

模型	在线	实时性	检测 SEU 的平均命中率/%		
			8 × 8	16 × 16	32 × 32
在线 BIST	Yes	No	46.9	38.7	32.8
模型 B	Yes	Yes	100	100	100

4 结语

仿真结果表明:本文提出的三种模型结构均可以实时检测出输出信号是否正确。因为设备利用率和吞吐量的不同,它们可以被用于不同的应用场合。模型 A 需要大量的硬件资源,因此它可以被拥有足够硬件资源的系统更好地接受;模型 B 需要提高 FPGA 的时钟频率,它可以被应用于小规模的应用场合;模型 C 比模型 A 有着更大的吞吐量和更少的资

源,因此它可以被那些允许提高频率并且需要约束硬件资源的系统所应用。

本文的模型优势是它可以实时地检测和定位错误,这些特点对于一些需要高可靠性的系统来说很重要。尽管在线 BIST 可以定位错误,但一些 SEU 可能由于它低速的错误扫描而被忽略掉。另外,在线 BIST 检测 SEU 的平均命中率随着线路规模的上升而降低。本文所提出的模型的优势是无论电路规模的大小,都可以保证 100% 的命中率。模型 B 的另一个特点是它可以实时检测输出信号是否正确。

参考文献:

- [1] BIDOKHTI N. SEU concept to reality (allocation, prediction, mitigation) [C]// 2010 Proceeding of Reliability and Maintainability Symposium. Piscataway: IEEE, 2010: 1-5.
- [2] MITRA S, MCCLUSKEY E J. Word-voter: a new voter design for triple modular redundant systems [C]// Proceedings of 18th IEEE VLSI Test Symposium. Washington, DC: IEEE Computer Society, 2000: 465-470.
- [3] MATSUMOTO K, UEHARA M, MORI H. Stateful TMR for transient faults [C]// 2010 World Automation Congress. Piscataway: IEEE, 2010: 1-6.
- [4] ABRAMOVICI M, STROND C. BIST-based test and diagnosis of FPGA logic blocks [J]. IEEE Transactions on Very Large Scale Integration Systems, 2001, 9(1): 159-172.
- [5] HSU C-L. Built-in self-test design for fault detection and fault diagnosis in SRAM-based FPGA [J]. IEEE Transactions on Instrumentation and Measurement, 2009, 58(7): 2300-2315.
- [6] ABRAMOVICI M, STROND C, HAMILTON C, et al. Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications [C]// Proceedings of International Test Conference. Washington, DC: IEEE Computer Society, 1999: 973-982.
- [7] ABRAMOVICI M, EMMERT J M, STROUD C E. Roving STARs: an integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems [C]// Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware. Washington, DC: IEEE Computer Society, 2001: 73-92.
- [8] SHNIDMAN N R, MANGIONE-SMITH W H, POTKONJAK M. Fault scanner for reconfigurable logic [C]// Proceedings of the 17th Conference on Advanced Research in VLSI. Washington, DC: IEEE Computer Society, 1997: 238-255.
- [9] MELXNER A, BAUER M E, SORIN D J. Argus: Low-cost, comprehensive error detection in simple cores [J]. Microarchitecture, 2007, 28(1): 52-59.
- [10] REIS G A, CHANG J, VACHHARAJANI N, et al. Design and evaluation of hybrid fault-detection systems [C]// Proceedings of the 32nd International Symposium on Computer Architecture. Washington, DC: IEEE Computer Society, 2005: 148-159.
- [11] REIS G A C, CHANG J, VACHHARAJANI N, et al. SWIFT: software implemented fault tolerance [C]// Proceedings of the International Symposium on Code Generation and Optimization. Washington, DC: IEEE Computer Society, 2005: 243-254.
- [12] ALKHALIFA Z, NAIR V S S, KRISHNAMURTHY N, et al. Design and evaluation of system-level checks for on-line control flow error detection [J]. IEEE Transactions Parallel and Distributed Systems, 1999, 10(6): 627-641.