

一种基于 FPGA 的容错嵌入式系统设计

陈国林, 章立生

(中国科学院 计算技术研究所, 北京 100080)

(glchen@ict.ac.cn)

摘 要:在 FPGA 内部使用各种 IP 软核搭建了完整的嵌入式系统, 实现了用三个 MicroBlaze CPU 软核进行表决的三模冗余容错方案。同时对 $\mu\text{C}/\text{OS-II}$ 操作系统以及应用程序进行改进, 在程序的内部加入了错误检测和校正(EDAC)、函数堆栈保护等容错功能。通过实验证明, 该系统减小了器件本身和内存模块受到的 SEU(Single Event Upset)影响。

关键词:FPGA; SEU; 容错; 三模冗余; 错误检测和校正

中图分类号: TP316.2 **文献标识码:** A

Design of a FPGA-based fault-tolerant embedded system

CHEN Guo-lin, ZHANG Li-sheng

(Institute of Computing Technology, Chinese Academy of Science, Beijing 100080, China)

Abstract: Using all kinds of soft IP cores, a complete embedded system was set up in the FPGA. Due to the flexibility of FPGA design, the TMR of the three MicroBlaze soft cpu cores and a voter was implemented on the FPGA. At the same time, the support of software fault tolerance by the way of the EDAC(Error Detection And Correction) and the protection of system stack were added. The experiment results show that the system is equipped with the ability of the tolerance of the effect caused by SEU(Single Event Upset).

Key words: FPGA; SEU(Single Event Upset); Fault Tolerance; TMR; EDAC(Error Detection And Correction)

0 引言

SEU(Single Event Upset)是由带电粒子投射到集成电路器件的敏感区域引起的,通常会导致处理器内部寄存器内容改变或内存位翻转(bitflips),带来的后果可能是计算结果错误,程序执行序列错误,甚至是系统的崩溃。国外的许多研究机构都开展了抗 SEU 计算机系统的研究,提出了许多解决措施,如基于系统级冗余的方法,采用抗辐射的计算机组件方法^[1]以及基于软件的容错方法(比如 N 个版本软件^[2],基于算法的容错^[3],代码跳转检测^[4]),这些方法各有优缺点。系统级冗余和抗辐射组件尽管可以保证可靠性,但带来成本地急剧提高和系统规模的增加。软件容错技术主要应用在内存容错,对处理器本身的寄存器和运算器等的错误并没有提供很好的解决方法,同时软件的容错技术会引起性能地下降和软件程序规模地增长。

半导体技术的发展,特别是 FPGA 技术的迅猛发展,为我们提供了设计计算机系统更灵活的手段和平台。目前 FPGA 的规模已经达到几百万门,甚至有抗辐射的 FPGA 器件。选用基于 FPGA 的片上系统(SOC)方案来实现容错的嵌入式系统,具有重要的意义和广阔的应用前景。这种方案有以下几个优点:采用 FPGA 其集成度高,计算机系统设计简单,可靠性高,体积重量小;具有较高的扩展性,很容易实现定制的硬件容错逻辑,并且具有重新配置的功能;可以使用已有的 IP 核,缩短了开发周期。

本文提出了一种基于 FPGA 的容错嵌入式系统设计方法,并给出了 Xilinx FPGA 的实现方案。本系统主要考虑 SEU

对计算机系统的影响,通过多 CPU 核的设计方法,使得在单个功能单元失效的情况下系统的可靠性依然得以保证,并通过软件错误检测和恢复机制,使系统的额外负载和容错功能达到了一个较好的平衡。

1 多处理器内核的 FPGA 内部设计

由于采用了基于 FPGA 的设计方式,在定制系统的逻辑功能上具有了极大的灵活性和可扩展性。在 FPGA 内部系统设计中,本系统采用 CPU IP 软核的三模冗余,达到传统的板级三模冗余效果。图 1 是采用 Xilinx Vertex II FPGA 实现的容错计算机系统框图。

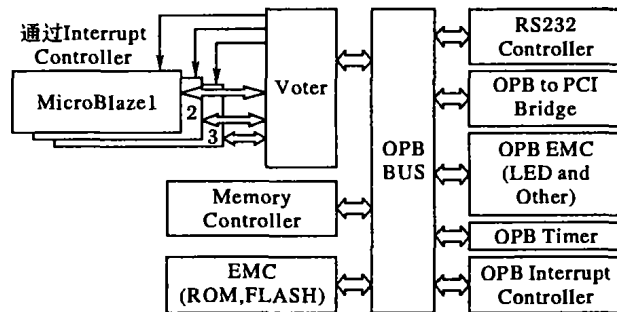


图 1 FPGA 内部结构

Xilinx 的 Vertex II 支持 MicroBlaze 处理器软核,并支持 IBM CoreConnect 片上互连总线。在 FPGA 内部集成了 3 个 MicroBlaze 处理器软核,通过一个 Voter 逻辑单元进行表决,实现了处理器的三模冗余。MicroBlaze 处理器采用 RISC 架构和哈佛结构,32 位地址总线,独立的指令和数据缓存,并且

有独立的数据和指令总线连接到 IBM 的 OPB (On-chip Peripheral Bus) 总线,这样它能很容易和其他外设 IP 核一起完成整体功能。如图 1 所示,三个 MicroBlaze 通过 Voter 连接到 OPB 总线上,所有其他设备的控制逻辑通过 OPB 接口挂在 OPB 总线上, MicroBlaze 通过 OPB 总线访问内存、ROM、FLASH 以及串口等设备。

三个 MicroBlaze 的数据总线和指令总线在接入 OPB 总线之前都经过了 Voter 进行比较。如果发现比较结果不一致,那么某个 CPU 所在的逻辑单元可能发生 SEU 错误。Voter 判断哪个 CPU 的数据或地址与其他的两个不同,这时 Voter 自动屏蔽出错的 CPU 发出来的地址和数据。此时整个系统由一个三模冗余转化成一个双机比较,同时向没有发生错误的 CPU 发出一个异常处理中断, CPU 响应中断,并且调用异常处理程序,记录下此时 CPU 的各种状态,在软件层面把出错的 CPU 屏蔽出去。假如其中又有一个 CPU 受到 SEU 的影响发生了错误,那么 Voter 判断出错误, FPGA 进行全局重新配置,回到原来的三模冗余状态。但由于 SEU 的发生频率较小,这样的情况会较少发生。

Voter 逻辑的结构设计如图 2 所示。

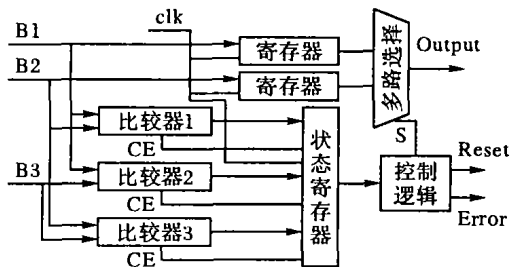


图2 voter逻辑设计

Voter 模块有三个输入 B1、B2、B3,代表三个 MicroBlaze 数据总线上的数据输入,三个输出 Output 表示经过比较的总线数据,Reset 信号用来复位系统,Error 信号表示出错状态。如果 Voter 的三个比较器自身发生错误,那么比较后的输出有误,可以由控制逻辑判断纠正。如果控制逻辑发生了错误,那么只能等待 watchdog 超时,等待外部计算机对之进行重新配置。

2 μ C/OS-II 系统的容错设计和实现

FPGA 内部运行 μ C/OS-II 操作系统。 μ C/OS-II 操作系统具有如下的特性:可移植性,体积小,可裁减,多任务,服务执行时间可确定性,最重要的是它具有很高的稳定性和可靠性。为了减轻不可避免的 SEU 现象带来的影响,必须在软件方面增强 μ C/OS-II 操作系统和应用程序的容错特性。

2.1 加入系统的 EDAC 支持

EDAC (Error Detection And Correction) 是容错系统中用来避免内存中数据变化的常用方法,我们在本系统中增加了对 EDAC 的支持。

2.1.1 校验码的选择及其内存分配

海明码因为其高效简单的特性,在很多系统中都使用它进行差错的校验。本系统使用 (12,8) 的海明码,一个字节的码字对应 4 位的纠错位。海明码的编解码通常都是使用计算的方式来完成,但是由于本系统采用的码字较短,所以可以建立一个映射表进行简单的原码到纠错码的映射,表的大小为 256,其占用的空间很小。使用查找表进行编解码时只需要访问一次内存,不需要额外执行计算指令,大大加快了编解码速度。

为了使 EDAC 的功能对整个系统尽可能的透明,我们采用把原始数据和校验码分离的方法,保证原始数据的完整连续性,在不需要 EDAC 时可以对数据进行正常的连续操作,否则访问每一个数据前都需要先除去校验码,只能按单字节操作。存放分离的校验码的方法有两种,一是在每次生成校验码时动态分配一段地址,但这需要用户自己去分配这样一个空间,因为 μ C/OS-II 的空间分配必须要由用户来完成,这是一个比较明显的缺陷。本系统应用程序比较简单,占用的空间并不是很大,而且片上内存有 16M,可以通过简单映射到内存高端的一半地址,每一个低端地址的数据都对应一个高端地址的校验字节。虽然这种映射方式对代码段的数据也预留了空间,但是对于数据处理起来非常方便,原始数据和相应校验码的地址之间相差一个固定的偏移,不需要多余的计算。

2.1.2 两种纠错原语操作及其封装

对于内存数据的 EDAC 校验,构造了两个基本的原语操作, ENCODE 和 DECODE。这两个操作的对象是一个内存数据, ENCODE 原语对内存数据进行计算并更新校验码,而 DECODE 进行内存数据的检验纠错。

使用这两个原语可以构造出更多的功能较强的宏。常用的是下面两种宏,分别对应单个变量和内存段的保护和纠错。

1) PROTECT (var): 保护变量 var

VALIDATE (var): 对变量 var 进行检验纠错

2) PROTECT_STACK (argc): 保护参数个数为 argc 的堆栈空间

VALIDATE_STACK (argc): 对堆栈空间进行检验纠错

2.1.3 EDAC 功能的具体实现

本系统基于上面介绍的各种原语来实现 EDAC 容错:

1) 对应用程序中比较重要的数据变量在对其赋值后利用 PROTECT 宏进行处理,在使用之前用 VALIDATE 宏校验纠错。当然,使用这种方式对用户有较高的要求,他们需要分析各种变量对程序可能造成的影响。由于本系统应用程序并不是大规模的计算程序,进行的算术运算并不是很多,可以近似地认为在变量赋值之后,对它进行一定的算术运算依然是有效的,只有在数据作为函数的调用参数时,才需要对它进行保护。

2) 对应用程序运行过程中函数堆栈的保护。在程序的运行过程中,比较活跃的是函数调用时堆栈的变化。在进入函数时,用 PROTECT_STACK 对传入的参数进行保护,在退出函数时,用 VALIDATE_STACK 对传入的参数进行验证。根据 MicroBlaze 的 ABI 可以得到在函数调用时内存布局如表 1 所示。

表 1 函数调用的堆栈分布

| | |
|--------|-----------------------|
| 高地址 | |
| | 被调用函数需要的参数 (R5 ~ R10) |
| 旧的堆栈指针 | Link Register (R15) |
| | 本函数需要使用的寄存器 |
| | 本函数变量 |
| | 被调用函数需要的参数 (R5 ~ R10) |
| 新的堆栈指针 | Link Register (R15) |
| 低地址 | |

如表 1 所示,需要传递的函数参数是通过调用者自己的堆栈空间来分配的。这样只需要知道旧的堆栈指针就可以访

问传递到本函数的所有参数的值,而旧堆栈的指针就在 Link Register 中,PROTECT_STACK 和 VALIDATE_STACK 就是通过这个指针来进行访问的。在函数中使用堆栈保护的函数基本形式如下:

```
int function( par1, par2, ... )
{
    PROTECT_STACK( argc );
    ...
    VALIDATE_STACK ( argc );
}
```

3) 内存刷新进程。对于内存中一些固定不变的数据,比如操作系统和应用程序的代码段和常量,可以用一个刷新进程来进行定时的更新,纠正可能存在的错误,此进程为 Scrubber,可以根据需要设置需要刷新的时间间隔或是频率。Scrubber 主要的功能如下:

1) scrub_register: 登记需要刷新的区域和频率。

2) scrub_lock: 锁定需要进行刷新的区域并且进行校验纠错。

3) scrub_unlock: 对需要刷新的区域进行保护,并且解除锁定。

每一个进程如果需要使用 Scrubber 进行刷新时,通过调用 scrub_register 函数来进行登记,对需要刷新的区域进行操作时必须对他们进行锁定,以保证其一致性。

2.2 分离操作系统以及各个程序的地址空间

因为 MicroBlaze 并没有自己的 MMU,而且 $\mu C/OS-II$ 操作系统上的应用程序并不是以单独的 ELF 文件加载执行,而是和操作系统静态链接,操作系统和应用程序的代码和数据段都是在一起的。这种内存的分布方式在发生错误时,不容易判断到底是那个程序发生了问题。为此我们对编译脚本进行了更改,使每个程序的各个段都分离开来,中间保留一个隔离区域,其中填充 brk 指令。

2.3 异常及异常处理

对于程序异常的发生,以尽可能恢复程序运行为准则。

MicroBlaze 能够捕捉到的中断和异常如下:

1) 未定义的指令,指令的操作数部分改变;

2) 内部指令和 BUS 错误;

3) 用户中断,用于系统调用;

4) 软件中断,用于单步调试;

5) 看门狗中断,在程序没有响应时产生。

SEU 对程序执行结果的影响主要来源于内存中的程序指令和数据的改变,会直接导致以上的 1), 3), 5) 这几种错误。在各种中断和异常中比较特殊的是软件中断,它是通过 brk 或者 brki 指令来实现的。这两条指令在 $\mu C/OS-II$ 和各种应用程序中都没有使用到,利用这种特性可以用来判断是否有错误的程序跳转。对于没有使用的内存空间可以用 brk 指令的格式来进行初始化,如果程序进入这段区域,就会触发软件中断,此时可以判断 SEU 错误的发生,对之进行相应的处理。

每当一个异常发生时,异常处理程序的主要任务是记录下异常发生的时间以及地址,以便随后的统计分析,并且尽可能的恢复程序的运行。异常地恢复主要针对以下两种情况进行处理:

1) 发生了未定义的指令,这种情况是由于指令的操作码发生了变化, MicroBlaze 会记录下异常发生的地址即该指令的地址,通过和 ROM 中该程序地址的代码进行比较可以把出错的代码更新成原始值。

2) 软件中断异常,这是由执行了 brk 指令引起的,即程序发生了错误的跳转,进入了隔离区域。通过对指令集的分析发现,该例外发生时必然是一条跳转指令,其跳转的地址出现了错误,跳转的地址可能是立即数,或者存放在寄存器中。立即数错误发生在不可改变的代码段,前面的 EDAC 措施已经降低了它发生的概率,那么出现较多的将是寄存器中的跳转地址错误。可以构造一个简单的 RollBack 功能,因为已经保证了函数调用的各个参数的正确性,如果此函数没有使用全局的数据,那么可以在异常发生时那个函数的入口重新开始程序的执行。

3 容错性能测试和分析

3.1 测试环境

实验的测试环境由目标机和主机构成,目标机是运行在 FPGA 上的 $\mu C/OS-II$ 系统,主机是 Linux 系统。我们通过一个故障注入进程向内存中的地址随机注入错误位,模拟 SEU 的效果,算法见文献[6]。由于 $\mu C/OS-II$ 的进程调度都是在中断处理程序中进行,此进程的优先级设置成最高,保证其能按一定的频率运行,并根据传入参数控制注入的 bitflips 浓度。为了提高随机数的可靠性并且避免多余的计算,我们没有采用 $\mu C/OS-II$ 系统产生随机数的方法,而是由主机 Linux 系统产生随机数,在模拟进程开始执行时,通过串口与主机上运行的监控进程进行通信,获得随机数,用于 bitflips 的生成。最后的各种测试数据同样通过串口发送回主机,主机监控进程对之进行记录分析。

3.2 测试结果

由于 FPGA 下载文件的格式资料较少,不容易通过对其注入 bitflip 来模拟某个 CPU 出现工作异常,我们通过仿真工具,对 Voter 逻辑进行模拟,结果表明 Voter 逻辑设计正确,能够正确处理 CPU 出现各种异常的情况,并且产生相应的错误信号。

在加入了操作系统和应用程序的容错特性后,最后生成的程序大小比原始程序有所增加(不包括填充的 brk 代码),如表 2 所示。

表 2 程序代码的各个段

| 文件名 | text | data | Bss | dec |
|----------------|-------|------|------|-------|
| executable.elf | 21350 | 1643 | 6540 | 29533 |
| excutable1.elf | 34466 | 1956 | 8330 | 44752 |

表 3 测试结果

| 错误原因 | Rate 1000 | | | Rate 6000 | | |
|--------|-----------|----|----|-----------|----|----|
| | #1 | #2 | #3 | #1 | #2 | #3 |
| 未定义的指令 | 25 | 7 | 2 | 28 | 6 | 1 |
| 内部指令错误 | 15 | 2 | 2 | 17 | 3 | 2 |
| 软件中断 | 12 | 2 | 1 | 19 | 2 | 1 |

从表 2 分析可知,程序的大小比原始程序增加了 51%,这个比例相比一些软件冗余方式,如变量的复制、代码跳转检测,还是比较低的。表 3 为系统运行时所有容错特性开启前后的一次测试结果。测试时间 120 min,两种 Bitflips 的注入速率为 1000 和 6000。#1 表示在没有容错的情况下发生异常的次数,#2 表示在有容错的情况下发生异常的次数,#3 表示在进行了异常的处理后程序依然不能正确运行的次数。

(下转第 1922 页)

现 $M1$ 在 17 ~ 19、 $M2$ 在 5 ~ 6.5、 $M3$ 在 1.5 ~ 2 时的结果指标值比较好, $O1$ 、 $O2$ 固定取值; 让 $M1$ 、 $M2$ 、 $M3$ 分别取 40 个、30 个、10 个, 这样一共有 12000 个方案; 用模型对这些方案进行结果预测, 从中选出结果指标最好的一个, 作为最优方案推荐给用 户: $M1(17.65)$, $M2(6.55)$, $M3(1.85)$, $O1(45)$, $O2(1.5)$, 预计 R 是 94.5。对该方案做试验后 R 是 94, 确实优化了。

表 4 试验数据

| | $M1$ | $M2$ | $M3$ | $O1$ | $O2$ | R |
|---|-------|------|------|------|------|-----|
| 1 | 15.42 | 5.75 | 2.25 | 50 | 1.5 | 78 |
| 2 | 16.25 | 6.75 | 3.75 | 45 | 1.5 | 86 |
| 3 | 17.08 | 7.75 | 1.75 | 40 | 1.5 | 92 |
| 4 | 17.92 | 5.25 | 3.25 | 50 | 1.5 | 88 |
| 5 | 18.75 | 6.25 | 1.25 | 45 | 1.5 | 84 |
| 6 | 19.58 | 7.25 | 2.75 | 40 | 1.5 | 81 |
| 7 | 17 | 8 | 1.5 | 43 | 1.5 | 89 |
| 8 | 17.5 | 6.5 | 2 | 46 | 1.5 | 93 |
| 9 | 18 | 5 | 3 | 49 | 1.5 | 85 |

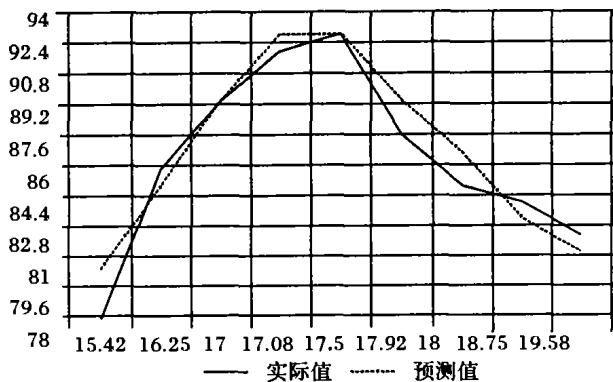


图 2 模型拟合图像

由于数据并不多, 这里也采用了基于贪婪搜索的方案优化算法进行了优化, 给出梯度最大方向上搜索得出的最优方案: $M1(17.67)$, $M2(6.56)$, $M3(1.98)$, $O1(45.67)$, $O2(1.5)$ 。对该方案做试验后 R 是 95。

(上接第 1918 页)

从相应的比较中可以看出, 相比在容错特性开启前, 各种系统异常的次数都有很大降低。在异常处理阶段, 对未定义指令的效果最好, 因为可以直接和原来的指令进行比较, 而其他两种异常采用的纠错措施起到的作用并不是很明显。

综合以上的结果可以看出, 在采用了多种容错手段后, 虽然不能够完全避免 SEU 对内存中操作系统和应用程序的运行带来的影响, 但是在一定程度上提高了整个系统的健壮性。

4 结语

在本系统中, 通过采用 FPGA 设计把通常使用的板级硬件冗余方式融合到 FPGA 芯片内部。这样不仅达到了冗余纠错效果, 而且减小了开发成本; 同时基于 FPGA 良好的可定制性, 项目的开发周期大大缩短, 系统的调试变得更加便捷。在加入了对 $\mu C/OS-II$ 嵌入式操作系统的容错功能后, 系统的健壮性得到了一定的改善, 系统的额外开销相比其他的一些软件容错手段, 如变量复制、代码跳转检测等方法要小。

利用 FPGA 进行系统设计, 更容易让硬件和软件有机的结合, 实现优势互补。这是在使用 FPGA 进行容错嵌入式系

经过以上实例验证, 引进试验设计方法的确减少了试验次数, 并且设计出的方案比较科学合理, 具有很强的代表性。对试验数据采用 LIBSVM 这种支持向量机的方法, 建立起来的回归模型比较符合真实模型, 是科学有效的回归建模方式。用回归模型来进行方案优化, 得出的方案也的确能取得更好的试验结果。对少量数据用贪婪搜索来做优化更容易实现, 其结果也达到了优化的目的。

3 结语

在新药试生产过程中, 引进正交试验设计、均匀试验设计方法科学合理的生成试验方案, 实现以较短的时间、较低的成本得到更好的新药配方和工艺。对于已有的小样本试验数据, 则采用基于统计学习理论的 SVM(支持向量机)方法来进行回归建模, 使用 LIBSVM 算法软件包建立起比较符合真实情况的回归模型, 并且用它来进行结果预测和优化, 对方案优化起到一定的指导作用。还介绍了比较简单的贪婪式搜索优化等方法, 作为 SVM 回归建模的有利补充。

参考文献:

- [1] 方开泰, 马长兴. 正交与均匀实验设计[M]. 北京: 科学出版社, 2001.
- [2] 方开泰. 均匀设计与均匀设计表[M]. 北京: 科学出版社, 1994.
- [3] 方开泰. 均匀设计及其应用[J]. 数理统计与管理, 1994, 13: 57 - 63.
- [4] SEBER GAF. 线性回归分析[M]. 方开泰, 等译. 北京: 科学出版社, 1987.
- [5] 方开泰, 全辉, 陈庆云. 实用回归分析[M]. 北京: 科学出版社, 1988.
- [6] 徐洪泉, 沈世镒. 基于计算机试验的均匀设计[J]. 高校应用数学学报, 1998, 13: 167 - 173.
- [7] 李敬芬, 李彬, 佟德成, 等. 均匀试验设计在 6-甲基-8-氰基-2', 3', 4'-三甲氧基黄酮合成中的应用[J]. 数理医药学杂志, 2003, 16: 67 - 68.
- [8] CHANG CC, LIN CJ. LIBSVM: a Library for Support Vector Machines[EB/OL]. <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>, 2004.

统开发的一个有益尝试, 相信在今后的项目中基于 FPGA 的应用系统会更加的普遍和成熟。

参考文献:

- [1] SIEWIOREK DP, SWARZ RS. Reliable Computer Systems: Design and Evaluation[M]. Digital Press, 1992.
- [2] AVIZIENIS A. The N-Version Approach to Fault-Tolerant Software[J]. IEEE Transactions On Software Engineering, 1985, 11(12): 1491 - 1501.
- [3] HUANG KH, ABRAHAM JA. Algorithm-based Fault Tolerance for Matrix Operations[J]. IEEE Transactions On Computers, 1984, 33(6): 518 - 528.
- [4] YAU SS, CHEN FC. An Approach to Concurrent Control Flow Checking[J]. IEEE Transactions On Software Engineering, 1980, 6(2): 126 - 137.
- [5] XILINX. MicroBlaze Processor Reference Guide EDK(v6.2)[EB/OL]. http://www.xilinx-china.com/ise/embedded/mb_ref_guide.pdf, 2003 - 12 - 09.
- [6] HOLDT D. Software-based Fault-Tolerance[D]. IMM, Technical University of Denmark, DTU, 2004.