第32卷 第14期 计 算 机 工 程 2006年7月
Vol.32 № 14 Computer Engineering July 2006
·工程应用技术与实现· 文章编号：1000—3428(2006)14—0224—03 文献标识码：A 中图分类号：TP302

动态部分重配置及其FPGA实现

李 涛，刘培峰，杨愚鲁
(南开大学计算机科学与技术系，天津 300071)

摘 要：动态部分重配置充分利用了FPGA芯片提供的可重配置功能，提高了FPGA芯片的利用率，减小了FPGA芯片的配置时间，有效地提高了系统的整体性能。该文介绍了动态部分重配置的两种实现方法，并在Spartan-II FPGA上进行了验证。
关键词：FPGA；Spartan-II；可重配置计算；动态部分重配置

Partially Run-time Reconfiguration and Its Implementation on FPGA

LI Tao, LIU Peifeng, YANG Yulu
(Department of Computer Science and Technology, Nankai University, Tianjin 300071)

【Abstract】 Partially run-time reconfiguration(RTR)fully takes advantage of the partial reconfigurable characteristics in FPGA devices and promotes the development of reconfigurable computing. It saves hardware resources as well as the time to configure the FPGAs, and improves the overall performance of the system dramatically. This paper introduces two methods to implement partially RTR, and successfully verify them by Spartan-II FGPA.
【Key words】 FPGA; Spartan-II; Reconfigurable computing; Partially run-time reconfiguration

可重配置计算(Reconfigurable Computing)[1,2]结合了传统的ASIC技术和通用处理器技术的特点，既具有通用处理器的灵活性，又具有ASIC的计算速度。随着FPGA应用领域的不断扩大，即使采用百万门级的FPGA，也很难一次容纳所有的电路设计。动态重配置(Run-time Reconfiguration，RTR)[2]是在一定的控制逻辑的驱动下，对基于SRAM的FPGA芯片的全部或部分逻辑资源实现在系统的快速重配置，能够满足更大规模的计算需求。动态全局重配置[2,3]只能进行全部芯片资源的重配置，虽然节约了芯片空间，提高了FPGA的资源利用率，但由于电路运行状态的转换和全部硬件资源的重配置信息量比较大，大大降低了系统的整体性能，在RRANN[4]系统中的重配置时间大约占了整个任务执行时间的80%。

动态部分重配置(Partially RTR)[2,4,5]是将任务划分成更细粒度的功能模块，根据任务执行的阶段分别下载，动态地重配置相应的部分电路，而不需要重配置的部分不受影响。它减少了重配置的范围和单元数目，重配置位流及配置时间显著减小。而且，这种更细粒度的任务划分，使得FPGA资源得到了更加有效地利用。

1 动态部分重配置系统及其实现方法

目前只有Atmel和Xilinx提供支持动态部分重配置的FPGA，因为Atmel的设备容量小，很难支持片上系统的实现，因此现有的很多系统大都采用了Xilinx的设备，如Spartan和Virtex系列的FPGA。根据FPGA芯片重配置时模块的不同替换方法，主要分为基于模块和基于差异的动态部分重配置两种方法[4~6]。

1.1 基于模块的动态部分重配置

在这种系统中，任务被划分为相互独立的子模块，这些子模块的总大小一般要超过FPGA芯片的最大容量，运行时只需将当前任务执行所需要的子模块配置到芯片上。因此，这种方法不受硬件资源的限制，可以完成任意规模的任务。任务模块之间的通信使用总线宏(Bus Macro)[6]来完成，总线宏允许信号穿越模块的边界而保持正确通信。

动态指令集计算机(DISC)[4]就是利用基于模块的动态重配置方法实现的，它是在FPGA中的二维配置逻辑单元按照行的方式构成的一维线性硬件模型下实现的可重定位硬件的系统结构，提供了面向应用的指令集，每个指令都作为一个独立的电路模块实现。DISC中的重配置模块按照水平方向布局，每个模块的宽度等于芯片的宽度，由于模块所占用的FPGA逻辑阵列的高度可以改变，因此指令模块库中的各模块可以按照其功能的需求被设计成不同的尺寸。静态的全局控制器和通信网络构成了一维线性模型上的全局上下文，在整个任务的执行过程中保持不变。通信网络起到了总线宏的作用，实现了重配置模块之间以及重配置模块与全局资源之间的通信。动态指令空间存放着完成特定任务的动态重配置模块，可重定位硬件使得任一指令模块都可以被配置到动态指令空间的任意部分，实现了“虚拟硬件”的概念，从而可以利用有限的资源完成更大规模的计算任务。而且，还实现了计算和指令模块重配置的重叠，隐藏了配置时间，有效地提高了系统的整体性能。

1.2 基于差异的动态部分重配置

这种方法是通过比较需要替换的模块与原有模块配置信息之间的差异，生成一个差异配置信息，替换时更新两个电路模块之间的差异部分，由于模块之间的差异通常都比较小，因此差异信息比较小，配置时间比较短。

与基于模块的方法不同的是，在配置时不是替换某一模块，而是更新整个动态指令空间。动态指令空间越大，重配置时生成的差异配置信息可能就越大，配置时间就越长，所

作者简介：李 涛(1977—)，男，博士生、讲师，主研方向：可重配置计算，并行与分布处理；刘培峰，硕士生；杨愚鲁，教授、博导
收稿日期：2005-09-21 **E-mail：**litao@mail.nankai.edu.cn

以在设计基于差异的部分重配置系统时，要尽可能最大化静态电路，将全局控制功能和各模块的共有功能等都交给静态电路完成，模块之间没有共性的部分才使用动态电路完成。

RRANN2[5]系统应用基于差异的部分重配置思想实现了BP算法，它通过增加静态功能模块，将其配置信息量减少了53.4%。系统中的全局控制器负责控制动态模块中子程序的执行，是不随任务的执行进度而动态改变的静态模块。正向传播、反向传播和更新 3 个神经处理器构成了系统的可重配置模块，它们在整个算法的执行过程中要动态改变，以实现BP算法中所有的计算任务。

2 动态部分重配置的实现与验证

除了具有适于动态部分重配置设计所需的器件之外，实现这样的系统还涉及一系列设计、优化的软件和方法。

(1)要有支持动态部分重配置的 FPGA 芯片；

(2)要有约束重配置模块在 FPGA 芯片中的位置和形状的开发　工具；

(3)要有支持部分重配置位流的生成和下载的开发工具；

(4)要有支持部分重配置的模块之间的通信接口；

(5)要实现动态部分重配置的控制。

本文采用了基于Xilinx Spartan-II XC2S100 设计实现的火龙刀开发板，Spartan-II系列FPGA芯片逻辑如图 1 所示。Xilinx ISE 6.3i软件包中的Floorplanner可用于确定模块在FPGA芯片上的位置和形状，BitGen命令及相应的参数可以生成部分重配置位流，iMPACT可以实现全局位流及部分重配置位流的下载。在基于模块的动态部分重配置方法的验证中，采用总线宏实现了各模块之间的通信。本文采用手工方式实现系统的动态部分重配置的控制[7]。

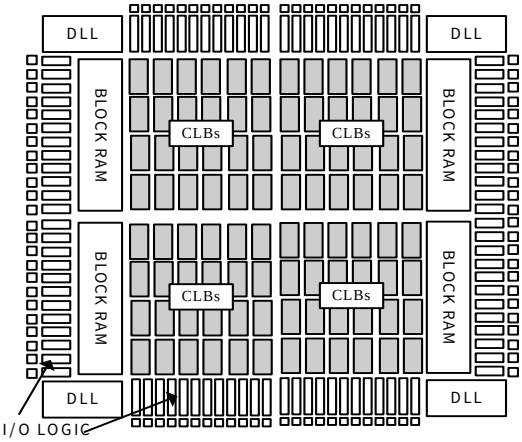

图 1 Spartan-II 系列 FPGA 逻辑图

2.1 基于模块的动态部分重配置的实现与验证

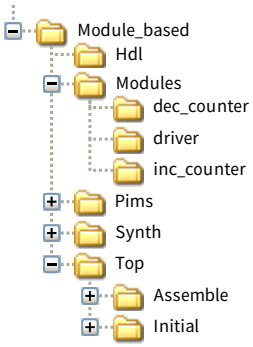

图 2 项目目录结构

基于模块的部分重配置采用了模块化设计方法的一个变种[6]，在设计过程中采用了如图 2 所示的目录结构，以利于组织每个设计和实现阶段所生成的文件。

本文实现了一个计数器程序，数码管的驱动作为静态模块实现，递增和递减计数作为两个相互独立的重配置模块，为了便于观察在模块重配置时不影响静态模块的工作，静态模块同时还驱动发光二极管 LED 进行显示。

(1)设计输入与综合(Design Entry and Synthesis)

首先将整个设计划分为相互独立的模块，并为其定义输入输出端口，然后使用顶层设计来组装各模块。顶层设计、驱动模块、递增和递减计数模块的 VHDL 设计输入保存为 Hdl 下的 top.vhd、driver.vhd，inc_counter.vhd 和 dec_counter.vhd 文件，使用 XST 对顶层设计进行综合生成 top.ngc 文件并存放在 Initial 目录下。

(2)总线宏(Bus Macro)

总线宏是位于相邻的或者不相邻的模块之间的预布线硬件电路，不随模块的重配置而动态改变，从而保证了重配置模块之间以及重配置模块和静态模块之间的正确通信。目前使用三态缓冲器(TBUF)来实现总线宏，8 个三态缓冲器一组，

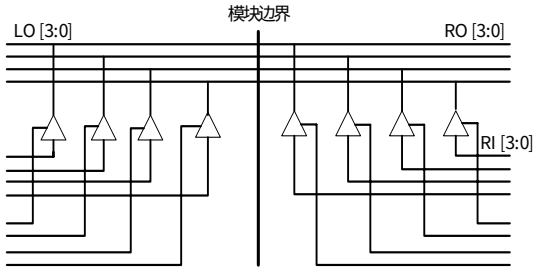

图 3 总线宏的物理实现

每个三态缓冲器传输 1 位信息，如图 3 所示。

根据顶层设计划分的模块之间的通信要求，在 FPGA Editor 中制作总线宏，并存为 Initial 目录下的 bm.nmc。

(3)初始预算阶段(Initial Budgeting Phase)

首先将 top.ngc 作为 Ngdbuild 命令的输入，生成 top.ngd 文件，然后使用 Floorplanner 对其进行位置和面积约束，产生约束文件 top.ucf，并手动增加对总线宏位置、重配置模块和物理管脚的约束，该约束文件用于每个模块的具体实现。如有需要，使用 Constraint Editor 对整个设计做全局时序限制。最后，使用 Ngdbuild 命令重新生成 tog.ngd 文件，用于每个模块的具体实现。

(4)模块实现阶段(Active Module Phase)

所有的静态和重配置模块都需要结合顶层设计和约束进行实现。首先将 top.ucf 和 top.ngd 文件拷贝到各模块目录下，并对每个模块运行 Ngdbuild、Map、Par 和 Pimcreate 命令，操作完成后将生成已布局布线的模块并放在 Pims 目录下。对重配置模块还要使用 BitGen 命令产生部分重配置位流 inc_counter.bit 和 dec_counter.bit，这里需要指定 BitGen 的-g ActiveReconfig : yes 参数。

(5)组装阶段(Final Assembly Phase)

本阶段将所有模块组合成一个完整的 FPGA 设计，首先将 top.ngc、top.ucf、bm.nmc 拷贝到 Assemble 目录下，运行 Ngdbuild 命令产生 top.ngd，其中的参数-modular assemble 表示保留模块实现阶段获得的布局布线结果。然后运行 Map、Par 和 BitGen 命令，得到完成布局布线的设计及其配置位流

文件 top.bit。

最后，使用 iMPACT 通过并口电缆下载 top.bit 并使用 JTAP 模式配置 Spartan-II FPGA 芯片，运行递增(递减)计数程序，然后使用部分重配置位流 dec_counter.bit (inc_counter.bit)可以将芯片配置为递减(递增)计数的程序，通过数码管显示计数过程，而在重配置过程中，LED 显示仍正常进行。

2.2 基于差异的动态部分重配置的实现与验证

在许多情况下，只需对设计进行小幅度的修改，比如在运行时需要对查找表(LUT)进行编程或者对 I/O 标准进行修改，这可以通过修改前端设计(HDL 或者原理图)或后端设计(NCD 文件)来实现对设计的修改。修改前端设计必须重新综合并生成新的布局布线 NCD 文件；FPGA Editor 工具可以对后端设计 NCD 文件的局部进行修改，然后使用 BitGen 及相应的参数生成远小于全局位流的差异位流。

对于前端设计，递增和递减计数程序的差别仅在于递增/递减控制的产生不同；对于后端设计，体现为产生递增/递减逻辑的LUT的配置方式不同。本文采用了修改前端设计的方法[6]。

(1)生成全局配置位流

在 ISE 6.3i 中，首先将递增和递减计数的 VHDL 代码保存为 inctimer.vhd 和 dectimer.vhd，对其进行布局布线分别生成 inctimer.ncd 和 dectime.ncd，然后生成全局配置位流文件 inctimer.bit 和 dectimer.bit。

(2)生成差异配置位流

BitGen –g ActiveReconfig : Yes –r inctimer.bit dectimer.ncd inctodec.bit 生成由递增到递减的差异配置位流。

BitGen –g ActiveReconfig : Yes –r dectimer.bit inctimer.ncd dectoinc.bit 生成由递减到递增的差异配置位流。

(3)验证

首先使用 inctime.bit 通过 JTAG 配置 FPGA 芯片，数码管显示递增计数结果，然后使用差异位流 inctodec.bit 可以将芯片配置为递减计数的程序，反之亦然。注意不能直接使用差异位流对芯片进行配置。在重配置过程中，整个芯片都暂停运行，重配置完成之后重新开始运行。

2.3 动态部分重配置的优势

动态部分重配置提供了一种利用有限的硬件资源完成计算任务的有效方法，提高了 FPGA 芯片的利用率，在任务的执行过程中根据需求动态地进行模块替换，使得可重配置计算不再受到硬件资源的限制。

由于只重配置部分电路，重配置位流和配置时间都明显减小(如表 1 所示)，有效地提高了系统的性能。在DISC[5]中，部分配置位流减为原来的 1/60～1/3，配置时间也大幅减少；RRANN2[6]系统的配置时间比RRANN[3]减少了大约 25%，但其性能却提高了近 50%。

特别地，基于模块的方法将各模块的公共部分和全局控制部分设计为静态模块，在重配置时可以将系统运行状态和中间结果等存入该静态电路中，既减少了相应的路由选择和控制等外围电路所需要的硬件资源，又节省了向外传送这些数据所需要的时间。

表 1 配置位流大小和(重)配置时间

比较项目 \ 重配置方式	全局配置	基于模块	基于差异
递增位流大小	96 kB	28.1 kB	29kB
递减位流大小	96 kB	29.4 kB	27.5 kB
递增(重)配置时间	5 s	2 s	2 s
递减(重)配置时间	5 s	2 s	2 s

3 总结

动态部分重配置提高了硬件资源的利用率，缩短了电路的配置时间，能有效地提高可重配置计算系统的整体性能。随着 FPGA 及其开发工具的逐步完善，必将更好地支持和推动可重配置计算的发展。

参考文献

1 Hartenstein R. A Decade of Reconfigurable Computing: a Visionary Retrospective[C]. Proceedings of Design, Automation and Test in Europe, 2001: 642-649.

2 Hutchings B L, Wirthlin M J. Implementation Approaches for Reconfigurable Logic Applications[C]. International Workshop on Field-programmable Logic and Applications, 1995: 419-428.

3 Eldredge J G, Hutchings B L. RRANN: The Run-time Reconfiguration Artificial Neural Network[C]. Proceedings of the IEEE for Custom Integrated Circuits Conference, 1994: 77-80.

4 Wirthlin M J, Hutchings B L. A Dynamic Instruction Set Computer[C]. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, 1995-04: 99-107.

5 Hadley J, Hutchings B. Design Methodologies for Partially Reconfigured Systems[C]. IEEE Workshop on FPGAs for Custom Computing Machines, 1995: 78-84.

6 Xilinx Inc. Two Flows for Partial Reconfiguration: Module Based or Difference Based[EB/OL]. http://direct.xilinx.com/bvdocs/appnotes/xapp290.pdf, 2003.

7 徐 欣, 于红旗, 易 凡等. 基于 FPGA 的嵌入式系统设计(Xilinx Edition). 北京: 机械工业出版社, 2005.

（上接第 220 页）

参考文献

1 Rivals I, Personnaz L. Nonlinear Internal Model Control Using Neural Networks: Application to Process with Delay and Design Issues[J]. IEEE Trans. on Neural Networks, 2000, 11(1)：80-89.

2 Lightbody G, Irwin G W. Nonlinear Control Structures Based on Embedded Neural System Models[J]. IEEE Trans. on Neural Networks, 1997, 8(3): 553:567.

3 刘小河. 基于神经网络的非线性系统内模控制的渐近完全控制[J]. 纺织基础科学学报, 1998, 11(1): 61-65.

4 许 昌, 吕剑虹, 郑 源. 最小资源分配网络及其在电站锅炉中的应用[J]. 中国电机工程学报, 2004, 24(11): 228-232.

5 张建华, 侯国莲. 电厂过热汽温神经网络内模控制系统的仿真研究[J]. 现代电力, 2000, 17(2): 19-24.