第38卷 第1期
2011年1月
计算机科学
Computer Science
Vol. 38 No. 1
Jan 2011

一种基于RSOC的软硬件协同设计方法研究

刘召煜 李仁发 陈 宇
（湖南大学计算机与通信学院 长沙 410082）

摘 要 针对可重构片上系统(RSOC)应用设计复杂、编程困难的问题，提出了一种软硬件协同设计方法。该方法整体框架基于特定功能模块的软硬件协同函数。给出了方法的基本流程、涉及的关键技术及实现方式，并验证了关键部分的设计方案及可行性。该方法为目标应用设计人员屏蔽了特定功能模块的软硬件实现细节，提高了基于RSOC应用的运行效率和灵活性。

关键词 可重构片上系统，方法学，协同设计，协同函数，约束文件

中图法分类号 TP301 **文献标识码** A

Research of a Method for Hardware and Software Co-design Based on RSOC

LIU Zhao-yu LI Ren-fa CHEN Yu
(College of Computer and Communication, Hunan University, Changsha 410082, China)

Abstract This paper presented a hardware-software co-design method for overcoming the design complexity and programming difficulty of reconfigurable system on chip(RSOC). The overall framework of the method is based on specific hardware-software co-functions. We gave the basic processes of the method, involved key technology and implementation methods. Also, verified a key part of design method and feasibility. The method shields hardware-software implementation details of specific function modules for target application designers, improves operational efficiency and flexibility of the application based on RSOC.

Keywords Reconfigurable computing, Methodology, Co-design, Co-function, Constraint file

1 引言

可重构片上系统(RSOC)在单个芯片中集成了可编程逻辑器件、微处理器核以及其他电路模块等部件，是一个兼具高效性和灵活性的混成系统。本质是利用可编程器件可多次重复配置的特性，在运行时根据需要动态改变系统某些部分的电路结构，而不影响其他部分功能。该系统在图像压缩、硬件演化计算、高速数字滤波器、定制计算、嵌入式系统等方面都有着广泛的应用前景。

可重构片上系统兼具灵活性与高效性的同时，随之带来了设计复杂度高、应用编程困难的问题。传统的软硬件协同设计方法学[1]一般先将系统按模块分割，然后逐步实现。系统设计定型后，目标应用程序的软硬构成不能再改变。传统方法没有考虑到可重构片上系统的动态重构能力，很难有效利用可重构计算资源；另外系统的软硬件划分过程复杂度很高，需要设计人员具备很高的专业技能。传统的软件编程语言只能利用设计空间的时间维度，传统的硬件语言则只能利用设计空间的空间维度。可重构片上系统兼具时间和空间维度编程的特性，给应用设计人员造成了编程方面的困难，应用设计人员需同时使用软硬件语言进行设计，并且还要考虑动态重构及软硬件通信结构。

针对可重构片上系统设计与编程存在的问题，当前的一些研究如G. Stitt等人[2,3]从底层入手，在额外的微处理器上直接对软件机器指令进行在线反汇编和在线综合，并配置到可重构区域。K. S. Hayden等人[4]设计了一个专门的操作系统，把配置到可重构区域的功能模块看作一种硬件进\线程。鉴于传统设计方法的不足，T. Wiangtong等[5]在软硬件划分时加入了对硬件任务动态调度的支持，提高了可重构计算资源的利用率。

我们把以协同函数为基础的软硬件协同设计方法不妨称之为过程级(或函数级)软硬件协同设计方法，该软硬件协同设计方法考虑到了可重构片上系统设计与编程的困难，对目标应用的某些功能模块封装成协同函数，屏蔽了具体的软硬件实现细节，从而方便用户设计和编程，普通的软件开发人员就能开发出高效率的软硬件混合系统。本文专门设计了协同函数资源调度器和约束信息在程序运行环境的注册机制，资源调度器在综合阶段生成目标应用程序的运行约束指导文件，既减少了系统运行时额外负担，又可使目标应用程序的执行更加灵活。

2 过程级软硬件协同设计方法流程结构

在软硬件协同函数的支持下，过程级软硬件协同设计方法将目标应用中关键部分的软硬件实现透明化，屏蔽了软硬件的具体实现细节。过程级软硬件协同设计详细流程如图1

到稿日期：2010-03-05 返修日期：2010-05-20 本文受863国家重点基金项目(2007AA01Z104)资助。

刘召煜(1982－)，男，硕士生，主要研究方向为软硬件协同设计方法学，E-mail：y841018@126.com；李仁发(1956－)，男，博士，教授，主要研究方向为嵌入式计算系统及体系结构、无线传感网络等；陈 宇(1982－)，男，博士生，主要研究方向为嵌入式系统结构及设计方法。

所示，大体可分为描述、综合与运行时环境支持 3 个阶段。

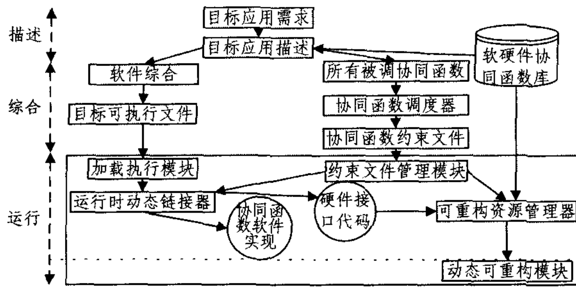

图 1 软硬件协同设计流程

系统设计与描述阶段主要包括目标应用需求分析与目标应用系统描述，该阶段的工作主要由应用设计人员完成。应用设计人员首先对目标应用做深入的分析，明确目标应用的需求以及确定所要用到的软硬件协同函数库，然后使用高级语言（例如 C 语言）对目标应用进行描述。通过调用软硬件协同函数库中的协同函数完成目标应用功能代码的编写。软硬件协同函数库中的每一个协同函数均有软件和硬件两种实现方式，应用设计人员在进行目标应用描述过程中只看到协同函数对外的统一接口，无需关心协同函数的具体实现方式。

目标应用描述综合阶段主要由两部分组成。在软件综合所在部分，编译器将目标应用描述转换为可在目标平台上运行的可执行程序，该转换过程与一般程序编译过程相比，添加了一些约束限制，比如需使用适合目标平台的交叉编译器，采用动态方式编译等。在协同函数资源调度所在部分，首先从目标应用描述中抽取所有调用的软硬件协同函数；然后，协同函数调度器确定每一个协同函数的执行方式（即软件或硬件方式），当某一个协同函数以硬件方式执行时，还需确定该协同函数硬件功能实现模块的资源布局约束信息；最后，协同函调度器会生成协同函数约束文件。协同函数约束文件是目标应用程序运行方式的指导文件，它确定了每一个目标应用程序的执行方式。

在程序运行环境阶段，首先定制可重构片上系统的硬件平台，然后在硬件平台之上移植操作系统，操作系统为可执行的目标程序提供基本的软件运行环境。目标应用加载模块将可执行的目标程序加载到操作系统提供的运行环境中。在目标程序开始执行之前，程序的运行环境还要做一些初始化工作。约束文件管理模块读取协同函数约束文件中的信息，将各协同函数的约束信息组织成一个链表结构。当目标应用调用软硬件协同函数时，执行线索转移到动态链接器。动态链接器在协同函数约束信息链表中查找约束信息，根据对应的约束信息链接到该协同函数的软件实现部分或硬件实现部分。被调用的协同函数如果被约束为软件实现方式，则直接切换到软件实现代码部分，和一般的共享库函数调用解析的过程类似；如果被约束为硬件实现方式，则切换到硬件接口代码，硬件接口代码负责与可重构资源管理器交互。可重构资源管理器管理片上系统的可重构资源。

3 过程级软硬件协同设计方法的关键问题

3.1 软硬件协同函数的结构和设计方法

软硬件协同函数由协同函数开发人员设计，每一个协同函数均有软件和硬件两种实现方式，如图 2 所示。协同函数的软件实现方式由纯软件代码编写完成，与一般的软件函数实现方式相同。协同函数的硬件实现方式由硬件功能模块硬件接口代码组成，硬件功能模块是硬件功能的具体实现，使用硬件描述语言（如 VHDL，Verilog 等）进行描述，并最终综合成在可重构片上系统可配置的位流文件；硬件接口代码是对硬件功能实现模块的封装，类似于设备驱动程序，它把协同函数的硬件实现方式封装为函数形式。

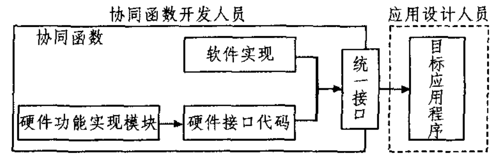

图 2 软硬件协同函数

每一个协同函数的软件实现和硬件实现重复实现同一个功能，功能上等价。硬件接口代码与软件实现函数具有相同的参数和返回值，可约定在软件实现的函数名前增加某个前缀，用该加了前缀的软件函数名作为硬件接口代码的函数名，以区别协同函数的不同实现方式。每一个协同函数对外提供一个统一的接口，可以约定将软件实现的函数名作为该协同函数的统一对外接口。应用设计人员在描述目标应用程序过程中，当调用协同函数完成某项功能时，和一般的函数调用相同，它只需了解协同函数提供的统一对外接口，无需关心协同函数的实现方式。

另外，协同函数的硬件实现方式是借助 RSOC 的可重构资源完成的。而可重构片上系统的可重构资源有限，动态重构技术可在运行时替换被配置到动态重构区域且处于空闲状态的协同函数，使某些协同函数共享动态重构区域，有效利用片上动态可重构资源。动态可重构资源按列分配[6]，动态重构硬件单元模块如图 3 所示，其由一组排成二维矩阵的可重构逻辑块（CLB）组成，有限数量的专用资源列被分布在 CLB 列之间。使用基于 Slice 的总线宏作为通信通道，它通过硬件布线而得到，放在固定的位置，在重构时固定不变。

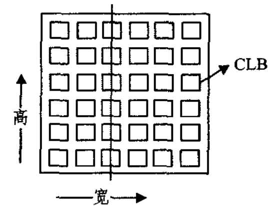

图 3 具有动态重构功能的 FPGA 模块

3.2 协同函数调度器工作模型

目标应用程序是建立在软硬件协同函数库之上的，协同函数库中的每一个协同函数在目标应用程序运行之前需确定软硬件执行方式。如果协同函数以硬件方式执行，还需确定所占资源的位置约束信息。协同函数资源调度器负责这些约束信息的生成与组织，同时尽可能减少目标应用程序的执行时间。

在调度器工作模型中，首先从目标应用的源程序中抽取所有的软硬件协同函数，按照软硬件划分算法[7]分成两组，即在处理器上以软件方式执行的协同函数和在可重构区域以硬件方式执行的协同函数。协同函数的软件执行方式与普通函数相同，而硬件方式需确定位置约束信息。设 r_i（$i=1\cdots n$）为一个以硬件方式执行的协同函数，w_i 表示 r_i 占用的可重构资源列数，h_i 表示 r_i 的硬件方式执行时间。令 w_i 为宽，h_i 为

高，r_i 可看作由 w_i 和 h_i 组成的二维抽象小矩形。另外，设动态可重构区域的资源列总数为 W，H 表示时间，令 W 为宽，H 为高，则可表示一个抽象的二维大矩形 R。协同函数布局约束信息的确定问题可以抽象为将 n 个 $r_1, r_2, \cdots, r_n$ 小矩形排列在 R 中，不能超过 R 的宽度边界，且 h_i 的高度方向与 R 的高度方向保持一致，求 $r_1, r_2, \cdots, r_n$ 在 R 中以何种方式排列，使 H 尽量地小。使用回溯算法可以求解该问题，核心程序描述如下：

```
int solve(void)
{
  如果将所有小矩形放入空间 R′中且比最优解低，则记录该结果；
  如果本节点满足剪枝定界的要求，则退出；
  在所有子空间中挑出一个高度最低的空间 R″；
  在最低点子空间 R″中遍历所有小矩形，如果可以放入，则将其放入，并递归调用本函数；
  若所有小矩形都不能放入，则将此子空间 R″归并到上层空间去；
}
```

算法的时间复杂度为 $O(n!)$。为了提高搜索效率，设置变量 b_{Height} 记录当前得到的结果中的最优解，用于定界。如果在求解搜索过程中的高度高于这个最优解，则不需要继续递归下去。设置变量 $r_{\text{Area}} = \sum_k (w_k * h_k)$，$k \in$ 还未被排列的小矩形标号，每次递归中挑出的最低子空间的高度为 sh，宽度为 sw，如果 $sh + r_{\text{Area}}/sw > b_{\text{Height}}$，则可以判定该节点以下的搜索达不到最优解，无需继续往下搜索，从而达到剪枝的目的。

通过上述算法最终可生成 $r_1, r_2, \cdots, r_n$ 的最佳排列方式，根据该排列方式可以构造协同函数硬件实现方式的位置布局约束信息。调度器最终将协同函数软硬件执行方式、位置布局、配置信息存储路径等信息按照一定的格式组织到目标应用运行时约束文件中，指导目标应用的执行。

3.3 约束信息运行时的注册机制

协同函数调度器生成的约束文件中至少应记录如下运行时指导信息：协同函数名、协同函数执行方式（软件或硬件）、协同函数硬件配置文件的存储路径、可重构资源管理器所需的协同函数位置布局约束信息等。

协同函数约束文件按照一定的格式和顺序存储约束信息，供约束文件管理模块使用，指导目标应用程序的执行。如图 4 所示，在目标程序执行之前，协同函数约束文件管理模块首先向程序运行环境注册，然后按照约束文件的规定格式读取协同函数的约束信息，建立协同函数的约束信息链表，每一个协同函数对应链表中的一个节点。运行环境中的动态链接器负责协同函数和普通库函数的动态解析和链接，它还可读取约束信息链表，然后在链表中查找当前协同函数的约束信息，根据查找到的约束信息确定协同函数的执行方式。如果当前正在解析的协同函数是被切换到硬件实现，则动态链接器实际切换到的是硬件实现的接口代码。硬件接口代码完成一些简单的初始化工作后，将控制转移到可重构资源管理器模块。可重构资源管理器模块首先检查当前协同函数是否已被配置，如果已经被配置到可重构区域，则直接启动具体硬件实现模块执行；否则读取约束信息链表，获得硬件功能实现模块的配置位流文件路径和位置布局约束。然后从存储器的相应目录中读取当前协同函数的硬件配置位流文件，并按照位置约束信息将硬件功能实现模块配置到动态重构区域。配置完毕后，启动硬件模块执行。

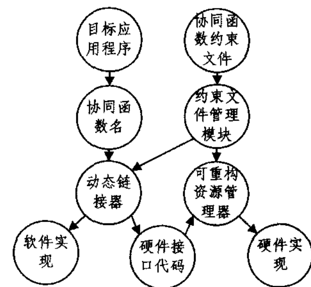

图 4 协同函数约束信息运行时注册

4 实验

4.1 目标应用程序运行的软硬件实验环境

本文采用 Xilinx Virtex-II Pro XC2VP30 FPGA 作为实验开发板[8,9]，整个片上系统被分成动态和静态两个区域，如图 5 所示，静态区域主要包括 CPU，DDR SDRAM，OPB 总线，ICAP 以及 Sys_ACE。动态区域即为动态重构部分，通过总线宏（Busmacro）与片上外围总线（OPB）连接。

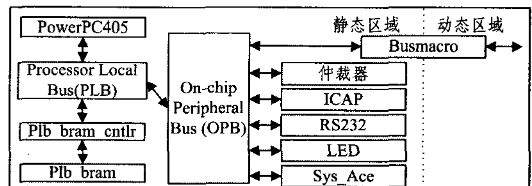

图 5 平台体系结构

静态部分和动态部分共同构成一个可重构片上系统硬件基础平台。在该硬件平台上移植 Linux（内核版本 2.6）操作系统，负责整个系统平台的控制，并且为用户目标应用程序提供运行时所需的软件环境。

4.2 软硬件协同函数实例

图 6 所示为截取一个软硬件协同函数的实现。该协同函数实现了海明编码功能，主要组成部分有软件实现、硬件接口代码、硬件功能模块以及协同函数对外接口。对外接口部分默认采用软件实现方式的函数名。

```
//cofun_hamming.h(协同函数对外接口)
extern void hamming_encode(unsigned *frame);
```

```
//hamming_encode.c (协同函数的软件实现部分)
...
void hamming_encode(unsinged *frame)
{
 int funcid=0;
 (*uptimefront)(&funcid);
 *frame=
hamming_process(hamming_expand(*frame));
 (*upstime)(funcid);
}
 unsigned hamming_proccss(unsigned frame)
 {
  u32 tmp;
  u08 xor;

  tmp = frame&P0_MASK;
  xor = ( (* (((u08 *)&tmp) + 0) ) ^
        (* (((u08 *)&tmp) + 1) ) ^
        (* (((u08 *)&tmp) + 2) ) ^
        (* (((u08 *)&tmp) + 3) ) );

  if(tab_parity[xor])
  ...
```

```
//hw_hamming_encode.c (协同函数的硬件接口代码)
void hw_hamming_encode(unsigned *frame)
{
...

HAMMING_mWriteSlaveReg1(base_addr,*frame);
  HAMMING_mWriteSlaveReg0(base_addr,3);
  HAMMING_mWriteSlaveReg0(base_addr,3);
...
}
```

```
--hanmming.vhd (协同函数的硬件功能模块)
...
entity HammingCodec_29_24 is
 port (
  ENC_I : in  std_logic;
  DAT_I : in  std_logic_vector(23 downto 0);
  PAR_I : in  std_logic_vector(4 downto 0);
  DAT_O : out std_logic_vector(23 downto 0);
  PAR_O : out std_logic_vector(4 downto 0);
  ERR_O : out std_logic;
  UNC_O : out std_logic
...
```

图 6 一个协同函数主要组成部分实例

软硬件协同函数的软件实现部分使用 C 语言完成，与一般海明编码软件实现类似。协同函数硬件实现部分的接口代码类似于驱动程序，主要使用 C 语言编写，硬件功能实现模块的编写使用 VHDL 硬件描述语言实现。协同函数对外的统一接口包含在 *.h 头文件中。协同函数的软件实现部分和硬件接口代码通过交叉编译，被打包到系统的共享库中（Linux 操作系统下为 *.so 文件）。硬件功能实现模块使用综合工具以及特定平台的布局布线工具生成位流配置文件，

并且还可获得可重构资源占用信息。协同函数的软硬件执行时间使用专门的时间测量函数获得。

4.3 目标应用程序运行时支持

协同函数调度器生成的目标应用的运行时约束文件如表1所列。通过约束文件对目标应用程序进行控制，使程序的执行更加灵活高效，方便控制。应用设计人员可手动更改某个协同函数的执行方式。若将硬件执行方式的协同函数指定为软件方式执行，需将执行方式列的H改为S，并将位置布局约束对应列清空（设置为"无"）；若将软件执行方式改为硬件执行方式，相对复杂一些，不仅要将执行方式列的S改为H，还要修改所占用的位置约束列，不能使硬件实现的协同函数位置重叠。协同函数约束文件在目标应用运行前被加载，用来约束目标应用中被调用的协同函数。具体加载过程由专门设计的 loadConstraintFile(char * path)函数实现，并在运行环境中建立约束信息链表。

表1 约束文件所包含信息

协同函数名	占用列数	执行方式	布局约束	硬件配置文件存储路径
hamming_encode	3	H	1	/cofun/hamming/encode. bit
hamming_decode	3	H	4	/cofun/hamming/decode. bit
tri_des_encrypt	5	S	无	/cofun/3des/encrypt. bit
tri_des_decrypt	5	H	1	/cofun/3des/decrypt. bit
draw_line	2	S	无	/cofun/shape/line. bit
draw_circle	2	H	6	/cofun/shape/circle. bit

协同函数的不同实现方式的切换是通过修改的动态链接器（Linux的动态链接器为 ld-linux. so）实现的。动态链接器截取未定位的协同函数名，查询软件函数名对应的地址及硬件接口函数名对应的地址，根据约束文件中的信息让应用程序跳到正确的函数地址处执行。修改的动态链接器可以通过设置的环境变量来区别普通共享库函数和软硬件协同库函数。若是普通共享库函数，保持加载解析过程不变。可重构资源管理器基于可重构片上系统的内部配置访问端口（ICAP）模块实现，系统将 ICAP 作为一个字符设备集成到 Linux 内核中，依据约束信息链表中的位置约束，配置动态可重构区域。有了上述各模块的支持，目标应用程序就可以顺利执行。

结束语 以协同函数库为基础的过程级软硬件协同设计方法的目标是解决可重构片上系统存在的设计及编程困难问题。本文分别从目标应用系统的描述、综合和运行时环境3个阶段展开，研究了过程级软硬件协同设计方法流程。我们给出了软硬件协同函数的结构和设计方法；专门设计了协同函数调度器工作模型，并对目标应用程序的运行环境作了必要的修改，以支持调度器所产生的目标应用约束文件的运行时注册，为目标应用程序提供了完善的执行环境；验证了协同设计方法的关键部分设计及可行性。本文方法对应用设计人员屏蔽了可重构片上系统的软硬件实现细节，降低了可重构片上系统的设计与编程复杂度。另外，目标应用程序的执行方式由综合阶段生成的约束文件控制，既增加了程序的灵活性，又降低了系统运行时额外的开销。

在目标应用约束文件生成流程中，构造高效的调度器搜索算法，是下一步要深入研究的问题。

参 考 文 献

[1] Wayne W. A Decade of Hardware/Software Co-design[J]. Computer, 2003, 36(4): 38-43

[2] Stitt G, Vahid F. A Decompilation Approach to Par-titioning Software for Microprocessor/FPGA Platfor-ms [C] // Proceedings of the Conference on Design, Automation and Test in Europe. 2005: 396-397

[3] Stitt G, Guo Z, Vahid F, et al. Techniques for Synt-hesizing Binaries to an Advanced Register/Memory Structure[C]// Proc. of ACM Symp. on FPGA. 2005: 118-124

[4] Hayden K S, Brodersen R. A Unified Hardware/So-ftware Runtime Environment for FPGA-based Rec-onfigurable Computers using BORPH[J]. ACM Tra-nsactions on Embedded Computing Systems, 2008, 7(2): 59-64

[5] Wiangtong T, Cheung P, Luk W. Hardware/software co-design: a systematic approach targeting dataintensive applications[J]. IEEE Signal Processing Magazine, 2005, 22(3): 14-22

[6] Huang Wei, Edward J. Column-based Precompiled Configuration Techniques for FPGA[C]// Proceedings of the the 9th Annual IEEE Symposium on Field Programmable Custom Computing Machines. 2001: 137-146

[7] 吴强，边计年，薛宏熙. 基于抽象体系结构模板的多路软硬件划分算法[J]. 计算机辅助设计与图形学学报，2004，16(11)：1562-1567

[8] Xilinx Inc. Application Note: Virtex Series Con? gu-ration Architecture User Guide[EB/OL]. http://www. xi-linx. com/support/documentation/application_notes/xap-p151. pdf, 2004

[9] Xilinx Inc. Virtex-II Pro Data Sheets[EB/OL]. http://ch-ina. xilinx. com/support/documentation/virtex-ii_pro_da ta_sheets. htm, 2007

（上接第294页）

[4] Blome J A, Gupta S, Feng S, et al. Cost-efficient Soft Error Protection for Embedded Microprocessors [C] // Proc of the Int'l Conf on Compilers, Architecture and Synthesis for Embedded Systems. 2006: 421-431

[5] Hsueh M C, Tsai T K, Iyer I K. Fault Injection Techniques and Tools [J]. IEEE Computer, 1997, 30(4): 75-82

[6] Mukherjee S S, Weaver C, Emer J, et al. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-performance Microprocessor [C] // Proc of the 36th Int'l Symp on Microarchitecture. 2003: 29-40

[7] Lee J, Shrivastava A. Static Analysis to Mitigate Soft Errors in Register Files [C] // Proc of the Design, Automation, and Test in Europe. 2009: 1367-1372

[8] Montesinos P, Liu W, Torrellas J. Using Register Lifetime Predictions to Protect Register Files Against Soft Errors [C] // Proc. of the 37th Int'l Conf. on Dependable Systems and Networks. 2007: 286-296

[9] MIPS32 Architecture for Programmers Volume II: The MIPS32 Instruction Set [R]. MIPS Technologies Corp, 2002

[10] Ziegler J F. IBM Experiments in Soft Fails in Computer Electronics(1978-1994) [J]. IBM Journal of Research Development, 1996, 40(1): 3-18

[11] Wu Y, Larus J. Static Branch Frequency and Program Profile Analysis [C] // Proc of the 27th Int'l Symp on Microarchitecture. 1994: 1-11

[12] Nielson F, Nielson H R, Hankin C. Principles of Program Analysis(2nd) [M]. New York: Springer-Verlag, 2005

[13] Lunts A G. A Method of Analysis of Finite Automata [J]. Soviet Physics, 1965(10): 102-103

[14] SPIM: A MIPS32 Simulator [EB/OL]. http://pages. cs. wisc. edu/~larus/spim. html