文章编号:1673-0291(2015)03-0109-05 DOI:10.11860/j.issn.1673-0291.2015.03.019

SOC快速内存检查点的研究与实现

袁小龙[1,2,3],蔡 翔[1],宋莉莉[4]

(1.北京交通大学 电子信息工程学院,北京 100044;2.广东省普及型高性能计算机重点实验室,广东 深圳 518000;
3.深圳市服务计算与应用重点实验室,广东 深圳 518000;4.中国科学院 计算技术研究所,北京 100190)

摘 要:随着系统芯片SOC设计技术的普遍应用,对应的可靠性问题逐渐成为关注的焦点.提出一种应用在SOC设计上的内存检查点技术,通过硬件逻辑将内存中的数据备份到非易失存储介质中,系统恢复时将数据从存储介质取出,重新拷贝到内存,避免内存中的数据丢失;同时给出外部存储器到RAM的程序代码拷贝设计,显著提高了内存检查点保存的速度.在断电等突发情况下,可以有效地保护内存中的数据,具有很好的通用性,占用资源较少,可以广泛应用于实时系统的容错机制中.

关键词:片上系统;内存;检查点;非易失性存储

中图分类号:TN4 **文献标志码**:A

A study and implementation on SOC's fast memory checkpoint

YUAN Xiaolong[1,2,3], *CAI Xiang*[1], *SONG Lili*[4]

(1. School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China;
2. Guangdong Key Laboratory of Popular High Performance Computers, Shenzhen Guangdong 518000, China;
3. Shenzhen Key Laboratory of Service Computing and Applications, Shenzhen Guangdong 518000, China;
4. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: With the wide spread application of system of chip(SOC) design technology, corresponding reliability problem become the focus of attention. This paper presents a method of memory checkpoint applying on SOC which storing the data to non-volatile memory storage medium by hardware logic. The system fetches the data and puts data to memory, avoiding loss of information. In addition, this paper presents a design of program copy from external storage to random access memory(RAM), significantly increasing the speed of memory checkpoint. In the case of power outages and other emergency situations, the design can effectively protect data in memory. The proposed design method has good generality and takes up less resource. It can be widely used in fault tolerance mechanism of real-time system.

Key words: system of chip; RAM; checkpoint; non-volatile memory storage

在现代信息社会中,嵌入式系统由于其灵活性及方便性得到了越来越广泛的使用.采用系统芯片技术可以将整个系统集成到单个芯片之中,其具有体积小、重量轻、功耗小和IP复用等优点.SOC技术目前正成为嵌入式实时系统发展的一个趋势,得到越来越广泛的应用[1].

当前SOC芯片的设计越来越复杂,随之而来的是系统可靠性问题越来越严重.在SOC芯片中,内

收稿日期:2014-11-19
基金项目:广东省普及型高性能计算机重点实验室/深圳市服务计算与应用重点实验室开放课题
作者简介:袁小龙(1971—),男,甘肃庆阳人,副教授,博士.研究方向为SOC和FPGA设计.**email**:yuanxl@bjtu.edu.cn.

存通常存储着应用程序的运行状态,一旦系统发生故障,程序只能从头开始运行,增加时间开销.为了保证在系统故障时程序可以持续运行,本文作者引进检查点技术[2].在检查点容错系统中,程序运行的状态被定期保存到非易失存储设备上,当系统出现故障时,应用程序可以从检查点映像中恢复执行.

检查点技术包括软件检查点和硬件检查点.目前的SOC平台多采用软件检查点方法,由开发人员在应用程序源代码中自行插入保存和恢复状态的代码,每隔一段时间将内存中的数据备份到稳定存储设备中,当系统故障时,通过读取保存在稳定存储设备的中间运行状态,从最近的中间运行状态恢复执行.由于受到稳定存储设备I/O带宽的限制,定期的数据保存会消耗大量的时间,降低了程序执行的效率[3].

本文作者提出了一种快速内存检查点方法.该方法属于硬件级检查点,具有通用性强、占用资源少的特点[4].这项技术可以广泛应用到对实时性要求比较高的系统上,如航天、医疗等领域,用来提高系统应对突发情况的能力.

目前硬件检查点有两种实现方式,可以在原有的系统中插入扫描链,用来访问系统中的寄存器,从而完成检查点的工作[5].或者分配一个可寻址的RAM,添加硬件逻辑来实现检查点功能[6].本文采用第2种方法.

1 目标SOC架构

该SOC用来研究容错性,包括CPU,外部存储器,UART接口,SPI接口,双端口RAM,片上总线等.SOC基本结构如图1所示.

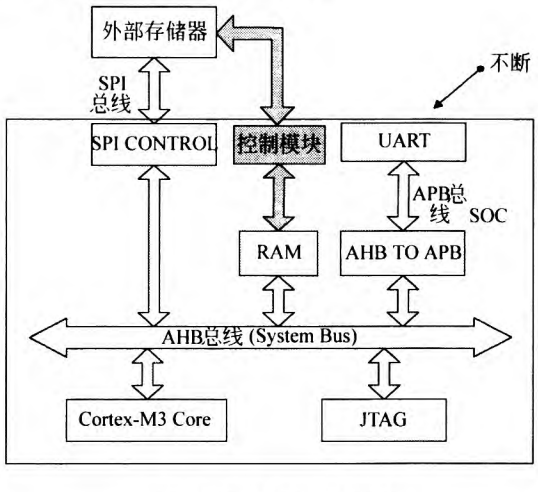

图1 SOC的基本架构

Fig. 1 Basic framework of SOC

图1中灰色部分为添加的硬件模块和通道.相关模块说明如下.

1)CPU软核:采用ARM公司提供的Cortex-M3,该产品主要定位于低成本单片机以及工业控制,它是一个32位处理器内核,使用THUMB-2指令集.

2)片上总线:采用AHB和APB总线,它是由ARM公司研发推出的总线类型.

3)外部存储器:采用256 KB的铁电存储器,型号为fm25h20,接口为标准SPI接口,铁电存储器具有RAM和ROM的优点,读写速度快并具有非易失的特性.在目标SOC中,程序可以从该存储器启动.

4)双端口RAM:由Xilinx的ISE工具提供IP核,大小为256 KB.

5)SPI CONTROL模块:CPU通过该模块访问外部存储器中的程序.

6)JTAG接口:方便开发者对程序进行调试.

7)控制模块:包括内存检查点模块和拷贝模块,内存检查点模块用来实现内存检查点功能,拷贝模块用来完善内存检查点,提高内存检查点的效率,这部分内容在3节会着重分析.

2 内存检查点设计实现

当前的检查点技术按其实现层次可分为应用级、系统级和硬件级.应用级检查点是在应用程序中加入备份自身内容的代码,可以根据应用程序的特点进行优化设置,但其通用性不好,只能针对具体的应用程序[7].系统级检查点由操作系统内核实现,对用户应用透明,但需要修改内核,同时周期性的检查点保存会占用大量时间[8-9].硬件级检查点通过特定的硬件逻辑来实现,每隔一段时间将运行的中间状态存入寄存器中,错误产生时将正确的中间状态恢复[10].

本文的设计方案属于硬件级的一种,检查点的实现完全由设计的硬件逻辑完成,不需要CPU参与.由于采用铁电作为外部存储,快速的读写可以保证系统在断电前完成检查点的操作.

2.1 内存检查点的设计思想

随着SOC芯片应用的领域越来越广,其在工作的时候很有可能会发生断电等异常情况.针对这种情况,本文作者提出的解决方案是在供电模块加入一个比较器,当供电电压小于一个阈值时,比较器输出中断信号,这时将内存中的数据备份到非易失的外部存储器中,等到下次系统上电时将这些数据恢复到内存,如果系统中寄存器的值也能在断电时保存,重新上电系统就可以从断电时的状态继续执行下去.

2.2　内存检查点模块的硬件架构

该 SOC 设计用 RAM 作为内存,中断触发时,希望将 RAM 中的数据存入外部存储器.外部存储器与 SPI CONTROL 模块直接相连,而 RAM 挂载在总线上,RAM 和外部存储之间没有直接的数据通道,设计一个内存检查点模块来实现数据交互.当中断到来,切断 RAM 与总线的数据通道,切断外部存储器与 SPI CONTROL 的数据通道,同时建立 RAM 与外部存储器之间的数据通道,见图 1 灰色部分.

2.3　内存检查点模块的设计方案

如上文所说,内存检查点模块用来实现内存检查点的功能.具体工作流程如下,首先从 RAM 的指定地址取出数据,然后向外部存储器发送指令,将取出的数据通过 SPI 总线送给外部存储器.

RAM 中的数据包括中断向量表,堆栈,变量和当前执行的程序.在本 SOC 设计中,RAM 数据的地址分配在 scatter 文件中定义,具体地址范围如图 2 所示.

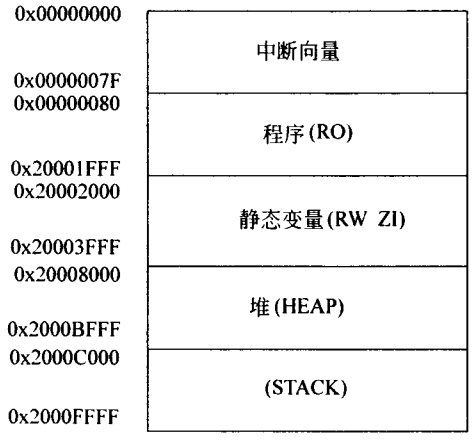

图 2　RAM 数据的地址分配

Fig. 2　Address assignments of RAM data

图 2 中,RO(Read Only)即要执行的程序和常量,RW(Read and Write)是已经初始化的静态变量,ZI(Zero)为没有初值的变量,在执行程序前系统将这些地址中的数据赋值为 0.

将每种数据类型的地址空间称为一个数据段.虽然各个数据段的地址索引是首尾相连的,但是执行程序时实际的数据量并不会占满整个地址空间,所以每个数据段会有大量的 0 数据,这些数据不需要备份.针对这种情况,该设计在每个数据段检测结束标志,从 RAM 中连续读到 50 个 0x00000000 时,内存检查点模块认为该数据段已经结束,后面的数据均为 0,跳到下个数据段的首地址取数据.当然这需要保证在程序中没有声明大量数据为 0 的数组,防止将其误判定为结束标志.

对应的硬件代码如下:

```
COUNT:
        begin
          state1 <= JUDGE;
          if(SRDATA == 32'b0)
            count<= count + 1;
          else
            count<= 0;
      end
JUDGE:
    if(count == 50)
      begin
      block<= block + 1;
      state1 <= INIT_ADDR;
    end
  else
    state1 <= ADD;
```

每从 RAM 中读出一个数据,就判断是否为 0,如果为 0,计数器累加,不为 0,计数器归 0,重新计数.在 JUDGE 这个状态里判断计数器的值,达到 50 跳到 INIT_ADDR 状态,更新 RAM 的访问地址,block 变量用来记录正在访问的数据段序号.

确定 RAM 数据的寻址方式后,下面分析内存检查点模块如何工作,其流程如图 3 所示.

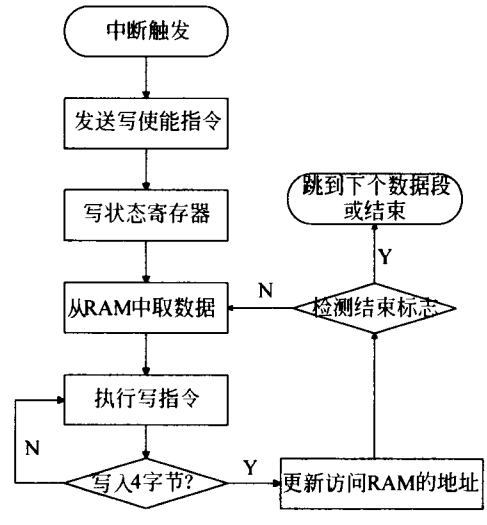

图 3　内存检查点模块的流程图

Fig. 3　Flow chart of memory checkpoint module

图 3 中工作原理如下.

1)当中断触发时,内存检查点模块产生写使能指令,此时可以将数据写入存储器.

2)写状态寄存器来控制写保护的区域及写入的方式,通过 AAI 模式将数据写入铁电存储器,见图 4.

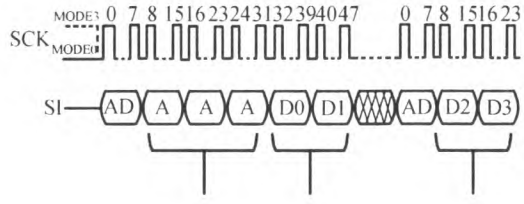

图 4 AAI 方式时序图

Fig.4 AAI sequence diagram

3)初始化访问 RAM 的地址,取出对应地址的数据.

4)在 AAI 模式下,内存检查点模块先送 1 个字节的 AAI 指令,然后发送起始地址,紧接着是 2 个字节数据.后面再发送时只需送入 AAI 指令和数据,地址默认为加 1,需要注意的是每发完两个数据需要有一段间隔时间,以保证外部存储器有充足的时间来处理接收到的数据.

5)因为发送给存储器的数据是从 RAM 中取出的,RAM 的数据位宽是 32 位,所以每发送 4 个字节需要更新访问 RAM 的地址,获取新的数据.

6)当检测到结束标志时,跳到下个数据段取数据或者结束备份操作.

3 程序代码拷贝设计

3.1 拷贝模块的设计思想

为了保证断电后有足够的时间备份内存数据,系统中加入一个电容提供电压,但是电容放电的时间有限,所以希望备份时间越短越好.考虑到程序代码是固定的,可以事先将其下载到外部存储器中,然后将程序代码拷贝到 RAM 中执行,这样就不需要备份 RAM 中的程序代码.

传统的方法是编译生成启动代码,系统复位时 CPU 先执行这部分代码,将下载到外部存储器的程序拷贝到 RAM,之后在 RAM 里执行[11].这种方法有它的局限性:只有系统复位时执行拷贝操作,在恢复内存数据时不希望系统重新启动.

本文作者设计的拷贝模块,如图 1 灰色部分由使能信号控制,与系统复位相互独立,保证在系统启动和恢复内存数据时都可以将外部存储器中的程序代码拷贝到 RAM 中.

3.2 拷贝模块的设计方案

与内存检查点的设计类似,本文设计一个拷贝模块完成外部存储器与 RAM 的数据交互,如图 5 所示,拷贝模块产生 SPI 总线的时钟和数据送给外部存储器,通过移位寄存器将接收到的程序代码转为并行,RAM 控制模块产生地址,共同实现对 RAM 的写操作.

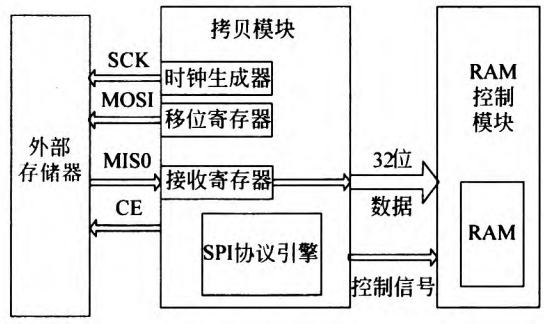

图 5 拷贝模块的接口设计

Fig.5 Interface design of copy module

拷贝模块是由一个状态机来实现的,图 6 说明了实现程序代码拷贝的整个过程.

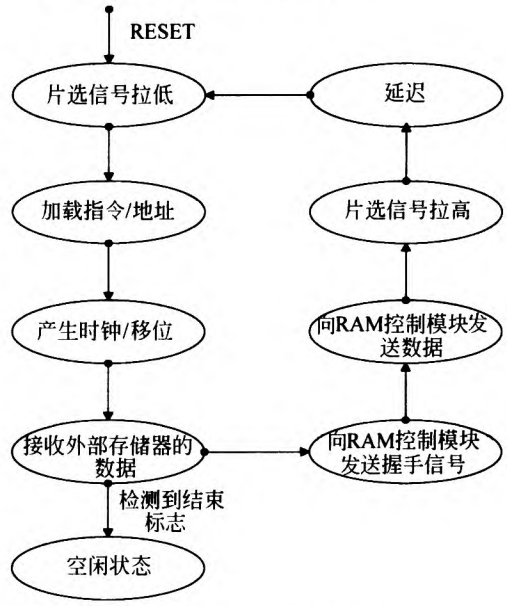

图 6 拷贝模块的状态机

Fig.6 State machine of copy module

图 6 中拷贝模块的工作原理如下:

1)在复位的情况下,首先将片选信号 CE 拉低,表示选中外部存储器,可以对其进行读写操作.

2)将读指令和访问地址加载到已经定义的一个寄存器中.

3)因为 SPI 接口是串行的,所以需要通过移位将读指令和地址顺序发送出去.同时,要给外部存储器提供对应的时钟.

4)指令和地址发送出去后,接收从外部存储器读到的数据.

5)向 RAM 控制模块发送握手信号和接收的数据,RAM 控制模块产生地址,完成对 RAM 的写入操作.

6)拉高片选信号,等待一段时间,执行下一次的

外部存储器访问和 RAM 写入操作.

4　实验结果及对比

加入程序代码的拷贝设计后,系统可以在任意时刻将外部存储中的程序代码拷贝到 RAM 中,所以在做内存检查点时不需要备份程序代码,这样极大提高了备份 RAM 数据的速度.

本文作者提出的设计在 FPGA 平台上实现,如图 7 所示.所用的 FPGA 芯片为 Kintex－7 系列.开发环境采用 Xilinx 提供的 ISE14.2,仿真环境采用 Modelsim 10.0,编程软件采用 Keil.

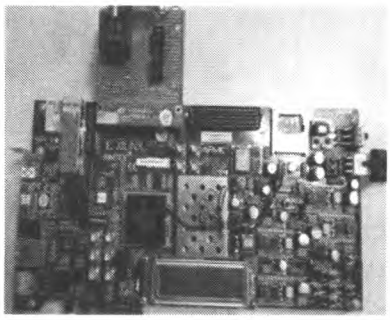

图 7　FPGA 系统平台

Fig. 7　FPGA system platform

表 1 列出了包含程序代码与不包含程序代码时备份 RAM 数据所需时间.

表 1　两种情况下备份 RAM 数据所需时间

Tab. 1　Required time when backing up RAM data

系统运行的程序	包含程序代码/μs	不包含程序代码/μs	备份时间减少/%
Uart 发送接收	1 292	512	60.4
定时器	2 650	784	70.4
Dhrystone	4 875	960	80.3
矩阵运算	5 620	720	87.2
计算器	7 216	845	88.3

表 1 给出系统执行的 5 个程序,分别比较它们在两种情况下备份 RAM 数据的快慢.

Dhrystone 是测量处理器运算能力的最常见基准程序之一,常用于处理器的整型运算性能的测量.计算器通过 FPGA 连接的液晶屏实现.可以看出,在执行同一程序时,忽略程序代码这部分数据,所需备份时间减少了 60%～80%,在备份时间较长时效果非常明显.表 2 列出系统执行不同程序时 RAM 各个数据段的数据量大小.

从表 2 中看出,程序代码的数据量明显高于其他几项,而且越复杂的程序数据量越大,占到了整个数据量的 90%以上.因此对于复杂的程序,我们希望忽略程序代码的搬移工作,所以,完成外部存储器到 RAM 的程序代码拷贝设计后,备份速度大幅提高.

表 2　不同程序下 RAM 各个数据段的数据量

Tab. 2　Data sizes of various data segment in different programs　　BYTE

系统运行的程序	中断向量	程序代码	静态变量	堆栈
Uart 发送接收	92	624	24	40
定时器	88	1 432	24	108
Dhrystone	92	3 420	328	256
矩阵运算	80	4 600	24	232
计算器	92	6 120	276	456

5　结语

本文在完成内存检查点的硬件设计后,同时提出一种外部存储器到 RAM 的程序代码拷贝设计.

1)在原始的内存检查点工作情况下,随着程序复杂性的增加,所需要的备份时间会相应增加.

2)加入程序代码的拷贝模块,针对任何程序,所需的备份时间都控制在 1 ms 以内.

3)加入程序代码的拷贝模块,当运行的程序足够复杂时,备份时间比原始的内存检查点系统减少了 80%以上,相应地,面积只增加了 7.8%.这项技术可以在系统运行大规模程序时,依然实现高效地对数据的保护,对航天、医疗和军工领域的实时性系统有着非常重要的意义.

参考文献 (References):

[1] Bin Y, Kang K, Kim H, et al. The development of SoC platform for embedded system applications[C]//International Conference on Convergence Information Technology, 2007:2286－2291.

[2] Zhang C, Deng S, Ning H. A local checkpoint mechanism for on-board computing[C]//IEEE International Conference on Information Science and Technology, 2012:520－526.

[3] 易会战,王锋,左克,等.基于内存缓存的异步检查点容错技术[J].计算机研究与发展,2014,51(6):1229－1239.
YI Huizhan, WANG Feng, ZUO Ke, et al. Asynchronous checkpoint/restart based on memory buffer[J]. Journal of Computer Research and Development, 2014, 51(6):1229－1239. (in Chinese)

[4] Koch D, Haubelt C, Teich J. Efficient hardware checkpointing: concepts, overhead analysis, and implementation [C]//Proc of ACM/SIGDA IntSymp on Field Programmable Gate Arrays, 2007:188－196.

[5] Tiwari A, Tomko K A. Scan-chain based watch-points for efficient run-time debugging and verification of FPGA designs[C]//Asia and South Pacific Design Automation Conference, 2003: 705－711.

(下转第 121 页)

[11] 刘坤会. Richard 模型的平均期望费用问题[J]. 数学学报,1988,31(6):786－793.
LIU Kunhui. The problem of average expected cost of Richard's model[J]. Acta Mathematica Sinica, 1988,31(6):786－793. (in Chinese)

[12] 刘坤会. 一类推广型脉冲控制[J]. 自然科学进展,2001,11(12):1324－1325.
LIU Kunhui. A class of extended impulse controls[J]. Progress in Natural Science, 2001,11(12):1324－1325. (in Chinese)

[13] 刘坤会,陆传赉,秦明达. 一类半鞅状态的平稳型脉冲随机控制[J]. 系统科学与数学,2004,24(2):249－260.
LIU Kunhui, LU Chuanglai, QIN Mingda. A class of stationary impulse stochastic control with state of semimartingale[J]. J Sys Sci and Math Scis, 2004,24(2):249－260. (in Chinese)

[14] Yang R C, Liu K H, Xia B. Optimal impulse and regular control strategies for stochastic diffusion problem[J]. Far East J Theo Stat, 2004,12(2):149－165.

[15] Egami M. A direct solution method for stochastic impulse control problems of one-dimensional diffusions[J]. SIAM J Control Optim, 2008,47(3):1191－1218.

[16] Bensoussan A, Long H W, Perera S, et al. Impulse control with random reaction periods: a central bank intervention problem[J]. Oper Res Letters, 2012,40:425－430.

[17] Liu Xiaopeng, Liu Kunhui. The generalization of a class of impulse stochastic control models of a geometric Brownian motion[J]. Science in China: Series F-Information Sciences,2009,52(6):983－998.

[18] Liu Xiaopeng, Liu Kunhui. A new class of impulse stochastic control models with nonnegative control quantity[J]. Science in China: Series F-Information Sciences, 2011,54(3):638－652.

(上接第 113 页)

[6] Mignolet J Y, Nollet V, Coene P, et al. Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip[C]//Proc of IEEE/ACM Design, Automation and Test in Europe, 2003:986－991.

[7] 韦中伟,陈海涛,王强,等. 支持数据库访问的进程检查点技术研究与实现[J]. 计算机工程与科学, 2011,33(8):84－88.
WEI Zhongwei, CHEN Haitao, WANG Qiang, et al. Research and implementation of a process checkpoint technology that supports database access[J]. Computer Engineering and Science, 2011, 33(8):84－88. (in Chinese)

[8] Hursey J, January C, O'Connor M, et al. Checkpoint/restart-enabled parallel debugging[C]// Proceedings 17th European MPI Users Group Meeting, 2010: 219－228.

[9] 门朝光,焦亮,李香,等. 基于 Linux 内核的进程检查点系统设计与实现[J]. 计算机科学,2009,36(4):192－195.
MEN Chaoguang, JIAO Liang, LI Xiang, et al. Design and implementation of process checkpoint system based on Linux kernel[J]. Computer Science, 2009, 36(4):192－195. (in Chinese)

[10] Li T, Ragel R, Parameswaran S. Reli: hardware/software checkpoint and recovery scheme for embedded processors[C]//Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2012:875－880.

[11] 万永波,张根宝,田泽,等. 基于 ARM 的嵌入式系统 Bootloader 启动流程分析[J]. 微计算机信息,2005,21(11):90－94.
WAN Yongbo, ZHANG Genbao, TIAN Ze, et al. Start-up code analyze of embedded system's Bootloader based on ARM processor[J]. Microcomputer Information,2005,21(11):90－94. (in Chinese)