摘要

在嵌入式 SoC 的设计开发过程中，调试测试环节占用了大部分开发测试时间。因此如何提高调试效率，缩短整个系统的开发时间，将直接关乎产品的 TTM（time to market）以及后续相关产品的开发设计。在嵌入式系统中，操作系统和应用程序通常存储于以 Flash、ROM、EPROM 等为代表的非易失性存储器中，其中 Flash 以存储速度快、容量大、使用寿命长等优点在嵌入式 SoC 系统中有着广泛的应用。在大量的程序调试过程中，Flash 下载编程占用很长的调试时间，所以如何快速、灵活地进行 Flash 编程，已经成为嵌入式系统开发的重要研究内容。本文在了解熟悉基于 JTAG 标准的片上调试技术原理的基础上，通过对不同类型 Flash 编程过程的差异性进行分析，总结出不同类型 Flash 编程过程的异同点，进而提出了一种基于 JTAG 标准的快速可重构 Flash 编程方法及其硬件架构。快速编程的核心思想是先通过 JTAG 接口将所需下载的目标数据下载到 CPU 通用寄存器中，并通过嵌入式 CPU 运行预先载入的 Flash 编程控制软件程序，再将目标数据写入到 Flash 中。这种方法将传统编程过程中复杂的 Flash 编程时序操作交由 CPU 完成，通过 JTAG 接口与 CPU 的数据的交互实现快速下载，由于 JTAG 串行总线仅下载目标数据，有效地增加了 JTAG 的带宽利用效率并提高了下载速度。在硬件上，主要采用程序断点控制和片上内存复用等技术，以国产 CK510 芯片的片上调试模块为原型，对其原有 Flash 编程过程进行改进，通过向片上内存加载不同的 Flash 编程汇编程序，利用状态机控制，使得硬件单元能够自动执行编程程序，很好的解决了由于 JTAG 串行端口反复传送指令而导致的编程过程中有效数据带宽占用率低下的问题，提高了调试效率。由于预先下载的 Flash 编程控制程序可以根据 Flash 的不同类型进行相应的替换，所以该方法可以实现对不同 Flash 编程的灵活支持。仿真测试结果表明，编程速度相比传统方法提高 17 倍，且具有硬件资源开销小、实行能力强等优点，具有一定的参考应用价值。

关键词： 片上调试；JTAG；Flash；编程；可重构

Abstract

In the process of embedded SoC design and development, debugging and testing take the most of the time of the SoC development. How to improve debugging efficency and reduce the system development's time, is related to the products' TTM and the follow-up products development and design. In embedded systems, operating systems and applications are mostly stored in nonvolatile memory, such as Flash, ROM, EPROM, etc. Usually, Flash memory is fast, large capacity, long life, so in SoC system has wide application. In a large number of program debugging tests, due to the multiple programming Flash to load the debugger, so how quickly and flexibly to Flash programming, has become a big issue of the embedded system development. In this paper, we focus on the point of view, and put forward a new JTAG-based method and architecture of Flash fast programming. The gist of the fast programming can be divided into the following steps. Firstly, download the target data to the CPU general-purpose register through the JTAG interface; secondly, control the CPU to run the Flash programming assembler and write the target data to the Flash. This new approach is different from the traditional download process, the CPU can deal with the complex Flash programming quickly, and because the JTAG serial bus is almost used by the target data only, the bandwidth utilization efficiency is very high. The JTAG download speed can be improved significantly. The new architecture is an improved CK510 debug interface that is adopted the technology of break point controlling and memory reusing. It supports the different types of Flash programming through the hardware distinguishes and executes the agent program itself, which has been download to the chip's memory first. In this way, the problem of low bandwidth usage which is caused by the sending of redundancy command through the JTAG interface has been resolved. Simulation results show that the programming speed is 17 times than the traditional methods', and hardware design costs small area resources. It appears both powerful and practical and has some reference value.

Key words: On-chip debugging; JTAG; Flash; Programming; Reconfigurable

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 浙江大学 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名: 签字日期: 年 月 日

学位论文版权使用授权书

本学位论文作者完全了解 浙江大学 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 浙江大学 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名: 导师签名:

签字日期: 年 月 日 签字日期: 年 月 日

致谢

时光似箭，日月如梭。两年半的硕士生活即将结束。回顾这一路走来，我得到了许多老师和同学们的帮助。是他们友善的关怀和我自己的努力使我能够顺利完成硕士阶段的学业及本论文的工作。

在此，首先衷心感谢我的导师沈海斌老师，沈老师是一位具有渊博专业学识和超前管理理念的老师。在近三年的时间里，沈老师一直以严谨、认真的科研作风要求我，不仅在学业、科研上对我进行悉心的教诲和指导，而且在生活上还给我很多细心的关怀与鼓励。尤其是他那孜孜不倦的治学态度和精益求精的工作作风更是给我留下了深刻的印象，令我受益匪浅。本次毕业设计的选题、收集材料、以及关键问题的探讨，都包含了他的大量帮助和指导。在此，我要向沈老师表示衷心的谢意。

其次，我要感谢史峥老师，每当和他谈心交流时，他那朴实的言语、和善的笑容以及平易近人的态度都给我带来了莫大的慰藉和帮助；要感谢研究所所长严晓浪老师，是他为我们创造了如此优越的学习和科研环境；要感谢吴晓波老师、何乐年老师、张培勇老师、王国雄老师、王维维老师、罗小华老师、竺红卫等老师，感谢你们在我求学期间对我的关怀和帮助。

再次，我要特别感谢中天微系统公司的孟建熠师兄，感谢即将同期毕业的曹葵康师兄、董文箫师兄、周喜川师兄，特别感谢即将同期毕业的周知名、王艳芳、王杰、顾良、蔡志翔等同学，感谢你们两年多来对我生活和学习上的关心；感谢已经踏上工作岗位的张倩、张昀、周功待、袁生光、王嗣平、金轶安、梁田、侯咏佳、胡欣幸、李刘林、刘洪庆、楼斌等师兄师姐，感谢你们平时对我的项目和学习上的指导和帮助；感谢实验室的张雷雷、陈武、李袁鑫、赵宇、李刚、方东博、谭萧峰、王琨、马力、周祺等其他同学，姓名不分先后，此处不一一列举。感谢与你们一起度过的美好时光，与你们的友谊是我一生的财富。

最后，我要感谢我的父母多年来对我默默的支持和无微不至的关怀，你们在我身上倾注了大量的心血，没有你们的养育之恩，就没有我成长进步的今天。

再次感谢所有关心我支持我的人！

陈超

2010 年 1 月于浙江大学

1 绪论

1.1 论文研究的背景及立题意义

嵌入式技术是21世纪最具生命力的新兴技术之一，IEEE曾将嵌入式系统定义为“控制、监视或者辅助装置、设备运行的装置”。可知，嵌入式系统是软件和硬件的综合体。它一般由嵌入式微处理器、外围硬件设备、嵌入式操作系统以及用户的应用程序等四个部分组成，用于实现对其他设备的控制、监视或管理等功能。随着IC技术的发展，CPU处理运算能力越来越强，运行频率也越来越快，这使得集成多功能、多接口的技术成为可能，加之用户对产品应用的需求，使得嵌入式系统从纯硬件实现和通用计算机软件实现之间脱颖而出，充分地展现了它的多功能和便捷性。经过近几年的快速发展，它已经成为电子信息产业中最具增长力的一个分支。随着手机、掌上电脑、机顶盒、数码影音播放器、GPS等新式数码产品的大量应用，嵌入式系统的设计正成为电子工程师们日渐关心的话题，其研究开发也越来越受到追捧。SoC（System on Chip）即片上系统，是一个有专用目标的集成电路，其中包括完整的系统并有嵌入式软件的全部内容。SoC的设计技术始于20世纪90年代中期，随着半导体工艺技术的发展，工程师们能将众多复杂功能集成到单硅片上来，提出一种高度集成化、固件化的系统集成技术，成功地实现了软件、硬件的无缝结合。在传统的电子系统设计中，需要根据设计要求的功能模块对整个系统进行综合，采用的是分布功能综合技术。对于SoC来说，整个系统的设计虽然也是根据功能和参数的要求来设计系统，但是SoC已经不再运用以功能电路为基础的分布式系统综合技术，取而代之的是以IP为基础的系统固件和电路综合技术。这个转变使得SoC出现，并由单纯的集成电路（IC）向集成系统（IS）方向发展，它已逐渐成为目前嵌入式系统应用领域中的一大热点，更有人称其为单芯片的嵌入式系统[1]。SoC芯片的规模一般远大于普通的ASIC，同时由于深亚微米工艺带来的设计困难，如更小的体积、更低的功耗，SoC设计的复杂度也越来越高。为了提高整个系统的可靠性，在SoC的设计中必须有仿真调试环节，这也是设计过程中最复杂、最耗时的环节。因此，采用先进的调试方法，提高调试效率已经成为SoC系统乃至嵌入式系统设计成功的关键。

在开发嵌入式系统的过程中，调试设计是一项不可或缺的工作，一般占到了总体系统开发时间的30%—50%，所以调试工具功能的强大与否直接影响到产品上市时间的快慢[2]。

嵌入式系统开发采用的调试方法为交叉调试（cross developing），即，由宿主机和目标机构成的调试系统。宿主机一般为带有调试软件和调试环境的PC机，目标机则是含有CPU的待调试系统，比如一块SoC。调试的过程中，首先宿主机与目标机通过一定接口和协议确定通信关系，然后将调试程序下载到目标机中，通过宿主机收发命令的控制，最终将调试的结果在宿主机端显示出来。

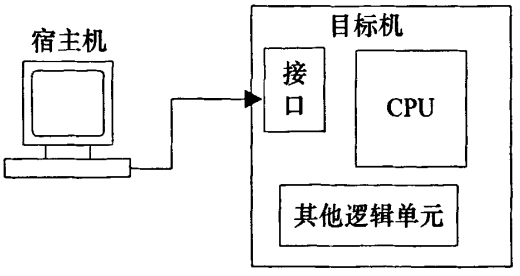

图 1-1 宿主机与目标机的连接

在上面的连接中，目标机必须包括与主机通信的一部分软件，这些软件占有一定内存、中断向量。而主机则应该能做以下工作：1. 将程序下载编程到目标机中；2. 在目标机上启动或停止程序执行；3. 检测内存和CPU寄存器。对系统进行调试时，宿主机处于主导地位，工作大致分为上面的三步。其中第1步是下载程序，交叉编译程序是可以在一种机器上运行却为另一个机器生成代码的编译程序[3]。下载时需先对其进行编译，然后再通过某种接口（如串口、并口、网口或某种特定的通信接口）下载到嵌入式系统中，接着再通过主机-目标调试程序进行调试，完成第2、3步的工作。假定调试系统已确定，那么在上述3步工作中，第1步在调试过程中占用的时间是客观的，即不以人为因素改变。

伴随着IC工艺技术的提高，嵌入式技术也产生了突飞猛进的发展，应用于各领域的SoC系统日趋完善，功能也越来越强大。随之也带来了一些问题，如片上存储器的容量逐渐扩大使得下载程序的时间更长；芯片的高集成度使得芯片电路布局紧凑，单片封装管脚变得非常密集等。这些问题都给嵌入式系统的调试开发带来了诸多不便。Flash属于非易失性存储器件，是目前大多数SoC系统中的关键部件，通常用来存放开机程序引导代码，在SoC系统开发调试过程中，对其进行下载编程是非常频繁的工作，编程方式的繁简与编程速度的快慢直接影响着SoC系统调试效率的高低。如文献[4]采用在系统编程方法，编程2M的Flash需要大约60分钟。过慢的下载编程速度浪费了调试过程中大量的宝贵时间。因此，设法简单、快速地实现Flash的编程，是提高调试效率、减少系统开发时间的一项重要工作。研究高性能的调试编程器，对嵌入式系统，特别是SoC系统的开发有着重要的

意义。

1.2 研究现状及发展趋势

从嵌入式系统产生到发展至今，其具体的调试手段也发生了一系列的变化。这些变化都是伴随着嵌入式软硬件发展而产生的。目前嵌入式处理器的主频越来越高，单位时间内处理的指令条数也越来越大，集成系统的功能复杂度也越来越强，调试开发中原来的一些调试手段便逐渐失去了使用价值，或者无法应对某些特定的调试场合，于是新的调试方法便不断涌现出来。为了更好地说明片上调试编程器的发展趋势，不妨从各类调试手段的研究现状说开去。总的来看，嵌入式系统开发的调试手段包括： ROM Monitor 方法、ROM Emulator 方法、软件仿真调试方法、使用在线仿真器 ICE（In-circuit emulator）方法、片上调试 OCD（On-chip debugging）方法。

ROM Monitor 是用一段驻留在目标机的非易失性存储器件上的小程序，辅助宿主机测试与调试开发者所编写的嵌入式程序。它可以通过对修改寄存器、单步执行等功能的监测实现对程序的简单查错。ROM Emulator 则是一种用于替代目标机上 ROM 芯片的仿真器，通过它，可以在目标机不具备 ROM 芯片的情况下进行调试开发，但是需要目标机支持外部 ROM 地址空间，并且需要占用 CPU、内存等资源。在早期的调试开发中，ROM Monitor 和 ROM Emulator 经常配合使用[5]，目前看来这两种方法已显得比较古老。

软件仿真调试方法即软件模拟器调试，这种方法是采用软件手段模拟真实硬件的工作情况，它需要把断点设置、监测变量等操作通过软件手段先进行模拟，然后再在这个虚拟运行环境中运行所有为目标机编译形成的机器指令，实现无差调试。由于这种调试方法可以脱离实际的硬件操作平台，所以显得较为方便。但是目前的嵌入式系统种类繁多，特别是不同的 SoC 系统可以采用同型号的 CPU，但是根据具体 SoC 功能的不同，其存储器类型、总线结构、接口方式等都存在着差异，这就给软件仿真调试的模拟带来了很大的困难，使其不能真实反映系统层面的问题。

ICE 方法一般是将底座上的 CPU 取下，然后用一个与外部在线仿真器连接的插头代替之，外部的仿真器代替原先 CPU 运行整个系统，这种调试方法被定义为在线仿真方法，即 ICE 方法[6]。采用 ICE 进行仿真调试，主要是用它接替了 CPU 的工作，所以 CPU 的地址线、数据线、控制线都在外部仿真器的监测下。通过 ICE 只要按照各类存储器的读写时序要求就能够实现调试程序的下载，当然所有的下载指令和数据都要通过软件端进行模

拟，牺牲时间就在所难免了。总体来讲，ICE 调试方法中包括了一个新 CPU，这个 CPU 可以接管目标机上的系统资源，因此可以不受限制地完成很多调试工作。但是随着目前高速、多样化 SoC 系统的出现，ICE 的时钟频率和性能支持也必须提高，这就加大了 ICE 的开发难度并且大大提高了产品的成本，加之 ICE 是特定于某种 CPU 而开发的，缺乏一定的通用性，所以 ICE 的技术发展也受到了阻碍[7]。

随着嵌入式技术的进一步发展，调试开发过程中对信号的实时跟踪和运行控制逐渐分离，运行控制逻辑被植入目标机系统的 CPU 核内，由一个专用的调试控制逻辑来实现（如片上仿真模块），然后再给用户开放出一个专用的通信接口，用户通过该接口从宿主机端对 CPU 核内的调试控制逻辑模块进行控制，在 CPU 正常工作模式和调试工作模式间灵活切换，从而实现对目标机上各种资源的跟踪和访问。这种将一部分硬件控制电路转移到片内，并提供大多数 ICE 的调试特性的技术，就是 OCD（On-chip debugging）方法[8]。OCD 方法不占用目标机的资源，也不过分依赖于宿主机端软件，调试环境和片上系统正常运行环境基本一致。并且同事支持软硬断点设置、追踪逻辑（Trace Logic）设置，能够较精确地测量代码执行时间、进行时序分析。因此，OCD 方法已经几乎成为目前嵌入式系统，特别是针对 SoC 系统调试的主流方法。OCD 又大致可分为 BDM 方法、JTAG 方法和 OnCE 方法等。BDM 是由摩托罗拉公司（Motorala）最先提出的调试技术。在这种调试技术中，调试逻辑控制模块直接内嵌于 CPU，通过它还可以直接访问总线上的设备。BDM 和开发系统之间的通信由专门的串行接口来完成，该接口主要包括 3 根信号线：数据输入线（DSI）、数据输出线(DSO)、时钟线(DSCLK)。在通信过程中，所有的调试命令都是由 17bit 的信息流组成，其中的 1bit 用来控制状态，其余的 16bit 都作为调试操作的命令、地址、数据。通过这有限的、固定的数据线进行数据传输显得繁琐、缓慢，完成一个调试操作过程往往需要传送多个数据流。另外，BDM 并不是一项开放的技术，只能限于 Motorola 处理器上，这一点也给它的发展带来了很大的限制。JTAG 源于 20 世纪 80 年代中期，当时联合测试行动组（joint test action group 简写为 JTAG）起草了边界扫描测试规范，该规范于 1990 年被批准为 IEEE 标准 1149.1 规定，并简称为 JTAG 标准[9]，因此，JTAG 既是一个组织的缩写简称，又代表了一种协议规范。从该协议正式发表后又陆续地做过一些修订，如 1993 年和 1995 年的补充，当前最新的修订版本发表于 2001 年[10]。基于 JTAG 的片上调试和 BDM 的方法有些区别，在 BDM 方法中所有的调试指令都是通过串行接口传入后，由内部 CPU 运算逻辑集中完成的；而在 JTAG 调试过程中，所有的外部串行输入都是由一个有限状态机进行控制操作，每一个操作又被分解为若干动作单元，有了状态机对不同

指令的筛选，用户可以根据自行需要，对内部的调试逻辑进行巧妙地设计，从而灵活地实现多种功能。通过它不但可以用于下载编程片内外存储器件的程序，而且可以方便灵活的用于调试，还可以进行系统的互联测试，用来检测电路是否存在连接故障等等。除了 JTAG 功能强大外，它还是个开放的国际标准，因此 JTAG 标准在工业界有着极为广泛的应用，几乎所有面向嵌入式、SoC 领域的微处理器、微控制器、ASIC 定制芯片、FPGA 等可编程逻辑器件等都支持 JTAG 标准。JTAG 已经在软件和硬件的开发和调试中起着举足轻重、甚至不可替代的作用[7]。OnCE 其实是 BDM 和 JTAG 相互融合的一种调试方式，由于它也是一种专有协议，并且是基于 JTAG 和 BDM 之上，此处不作过多赘述。

由以上分类比较可知，目前嵌入式开发中交叉调试的主流方法还是基于片上调试，因此，设法提高调试效率、加快调试下载编程速度的研究应该在此基础上进一步展开。众所周知，SoC 系统中的非易失性存储器件 Flash 一般用来存放开机引导程序，按其所处的位置可以分为片内和片外两种。对于片外 Flash 器件，早先有利用 ICT（in circuit tester）进行编程的。ICT 编程是通过编程夹具把 ICT 设备的测试通道连接到电路板上的 Flash 器件的各个管脚，然后把电路板上和 Flash 器件相连的其他器件进行电气隔离[11]，直接对 Flash 进行编程。ICT 编程需要在 Flash 器件的每个管脚设计 ICT 的测试点，并且需要占用单板的测试时间。由于 ICT 的设备比较昂贵，该方法一般适用于单板量产时。在单板开发阶段，由于无法测试和现场维护，一般不采用该方法。目前较为主流的嵌入式开发调试程序编程下载均是通过 JTAG 接口协议来实现完成的。通过 JTAG 接口进行下载编程可以利用 JTAG 的边界扫描链编程，也可以借助和 JTAG 集成的其他硬件逻辑进行编程，如片上调试模块。

边界扫描链（Boundary Scan）编程是通过对 BS 单元的串行移位操作来完成的，它需要芯片的每个 I/O 引脚都支持 BS 单元，通过 BS 单元的串行，构成一条扫描路径，用户在宿主机端对这条扫描路径进行控制和操作。首先下指令使 BS 器件处于 Extest 状态，然后选中边界扫描链，把对 Flash 操作的地址、数据、控制信号等，通过 JTAG 接口 TDI 管脚串行输入，锁存到对应引脚的 BS 单元，然后再刷新各个引脚状态，把 BS 单元中的信号送到对应管脚的 Flash 中去，再通过 TDO 输出数据。文献[11][12][13][14]均采用此类方法对 Flash 进行编程，其工作示意图如图 1-2 所示。

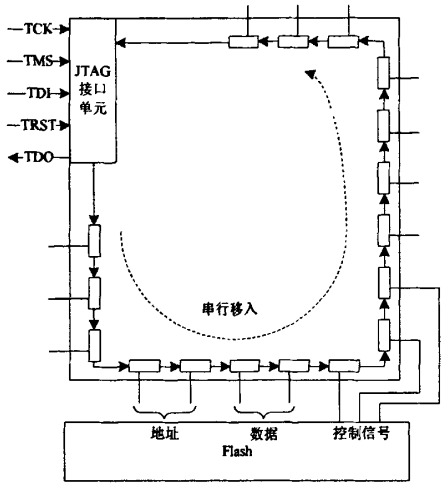

图 1-2 边界扫描链编程 Flash 示意图

利用嵌入式系统片上调试模块编程 Flash 是在片上系统内的 CPU 处于调试模式下，通过 JTAG 接口，利用片上调试逻辑访问 CPU 通用寄存器和内存，完成对 Flash 器件时序操作的加载，进而实现对 Flash 器件的编程。文献[15][16][17]中的 Flash 下载编程均采用此类方法，只是在具体设计上有所变化。利用该方法编程的示意图如图 1-3 所示。

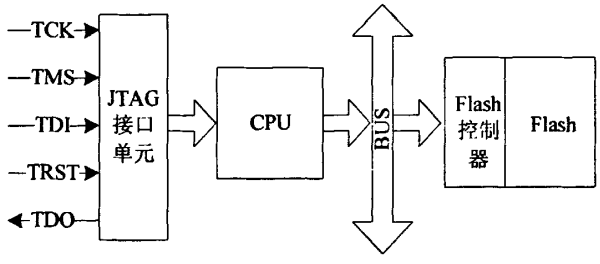

图 1-3 CPU 调试扫描链编程 Flash 示意图

在以上两种较为主流的编程方法中，边界扫描链编程 Flash 需要器件支持 BS 单元，而且需要宿主机模拟 Flash 的操作时序，比较依赖软件，扫描链的长度也限制了编程速度。借助片上调试模块编程 Flash 属于硬件调试范畴，结合硬件设计开发，应用更为广泛，方法更加灵活，比较少地依赖于目标机的软件，很好的适用于 SoC 系统的调试开发过程。文献[11]采用 JTAG 边界扫描链编程 Flash，扫描链的长度就有 224 个 BS 单元，加载一次数据就需要 224 个周期；文献[13]亦采用此法，换算的编程速度仅为 1.56KB/s，非常缓慢；文献[15]借助片上调试模块进行编程，由于编程过程中需要传入很多的控制指令，占用了 JTAG TDI 串行口的带宽，速度也不理想；文献[17]基于 EJTAG，在硬件上进行了改进，提供了一种较为快速的编程方法，但是由于硬件做好后只能产生特定 Flash 器件的编程指

令，不能广泛适用，灵活性略显不足。

针对以上分析，为了提高 SoC 系统的调试效率，顺应目前嵌入式技术的发展趋势，本文研究一种基于片上调试的新型编程器，以实现快速编程多种类型 Flash 的效果。

1.3 论文的研究内容、拟解决的关键问题及创新点

1.3.1 论文的研究内容

本文主要从片上调试模块的基础上设计 Flash 编程方法，故对其他的一些非 JTAG 接口编程方法不做赘述。文章从 JTAG 的基本结构入手，首先阐述 JTAG 接口的逻辑电路结构及工作原理，重点分析 JTAG 调试技术中测试访问端口、测试访问端口控制器、JTAG 指令寄存器和 JTAG 数据寄存器。解释了通过 JTAG 编程 Flash 的方法需满足的条件，阐述了在此条件上产生出的不同编程方法，以及不同方法间的核心区别。然后，在分析总结原有各方法的基础上选择利用片上调试进行 Flash 的下载编程，并提出本文设计的 Flash 编程的新方法，并对该编程方法进行说明论证。接着，文章以国产嵌入式芯片 CK510 的片上调试仿真模块（HAD）为例，分析了 HAD 的使用方法和硬件架构，并在此基础上设计编程器，对整个下载编程流程进行详尽阐述，给出具体的方案设计包括整体电路结构、子模块电路结构、寄存器功能描述以及状态转换图等。最后，在电路设计的基础上搭建验证平台，对设计部分进行仿真验证，给出新设计方案中 Flash 编程的性能指标，与原有方法和其他方法进行对比，说明设计的合理性和可行性。

1.3.2 拟解决的关键问题

由于本文属于新方法的硬件设计研究类，在研究方向上出现的关键问题不妨从原理分析、硬件设计、方法验证三个层面上去归纳。

在原理分析上，本设计是在交叉调试中片上调试方法的基础上展开的，寻求的关键点是原有编程方法的不足，比如编程速度、编程方法适用性问题，以及制约该类方法的瓶颈所在。在硬件设计上，本文的实现目标为快速可重构编程 Flash，其中处理异步时钟域的数据同步、防止数据出现亚稳态、利用硬件控制 CPU 执行编程指令都将成为关注的问题。方法验证上，主要是通过仿真工具对设计硬件进行验证，其中电路功能的时序分析以及下载数据的正确与否是此部分关注的重点。

1.3.3 创新点

本文提出一种可重构的Flash快速编程方法和硬件实现结构。整个设计过程为：首先，通过对不同类型Flash的编程时序要求的分析，总结了编程过程的异同之处，并找到基于JTAG标准的Flash编程过程中影响编程速度的瓶颈。其次，采用程序断点控制和片上内存复用等技术，在CK510片上调试仿真模块中设计片上调试编程器，通过向片上内存加载不同的Flash编程代理汇编程序，利用状态机控制，使CPU能够自动执行汇编编程程序，不但解决了由于JTAG串行端口反复传送指令而导致的编程过程中有效数据带宽占用率低下的问题，而且还支持不同类型Flash器件，避免了固定硬件设计中灵活性不足的问题。最后，仿真测试结果表明，片上调试编程器的编程速度理论分析值和仿真测试值间的误差很小，编程速度较原有方法有所提高，解决了片上调试过程中的编程速度低下的问题；另外，在资源占用方面，综合结果表明本硬件设计面积开销不大，耗用硬件资源很少，具有实际应用价值。

2 基于JTAG标准的片上调试技术原理

2.1 JTAG标准的背景和原理

1985年欧洲联合测试行动小组（Joint European Test Action Group）即JETAG开始制定JTAG标准，次年该工作组加入了来自北美的成员，于是更名为Joint Test Action Group，即JTAG。在接下来的几年里，JTAG技术的附属委员会制定并发布了一系列边界扫描的标准化提案，并于1988年将提案中的JTAG 2.0版本提交给了IEEE可测试总线标准委员会，经过大家对该标准反复的讨论和不断地改进，IEEE 1149.1-1990标准终于在1990年被证实批准。由于该标准是由JTAG组织创始，所以也称之为JTAG标准。JTAG标准制定的初衷是用来对PCB板进行电路功能测试的，包括检查电子元件间的正确互联、单个元件是否正常工作以及整个电路是否完成预期功能等。随着集成电路封装技术的发展，电路板本身越来越小型化，器件的封装体积也随之减小，引脚的排列也越来越密集，通过传统连接电路板上触点进行电路测试的手段变得非常困难，在这种情况下，边界扫描技术产生了。

边界扫描技术需要对边界组件（BS 单元）中的信号进行观察与控制，这就意味着必须设计一种结构实现输入输出信号的隔离，根据JTAG标准，BS单元常用图2-1的方式进行构建。

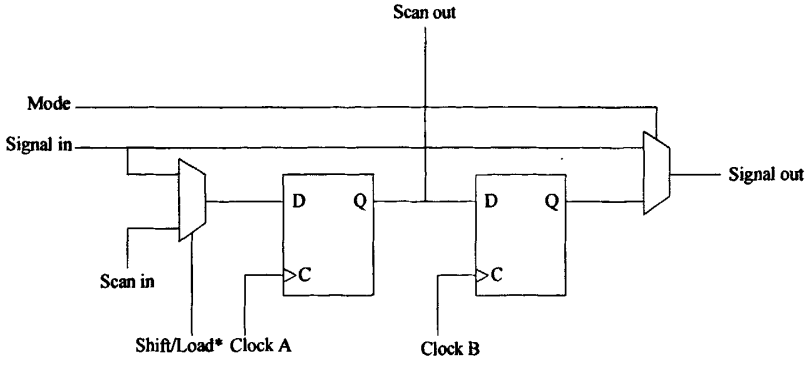

图 2-1 边界扫描寄存器单元

由上图可知，边界扫描寄存器单元由两个选择器和两个触发寄存器组成，它是介于外部引脚和芯片内部逻辑之间的。在芯片正常工作状态下，Mode信号直接选通信号Signal in和信号Signal out，外部输入信号可以直接到达内部，此时边界扫描单元是“透明的”，仿佛并不存在一样。当芯片进入测试模式后，Mode信号选通下路触发器电路，边界扫描单

元就开始工作了。其大致工作过程为：先由 Load*（*表示低电平有效）选择信号 Signal in 和第一个触发器 D 端相连，当 Clock A 时钟脉冲到来时，Signal in 的信号出现在 Q 端，即 Scan out。由于芯片的每个引脚上都有一个 BS 单元，并且连在一起，所以当所有引脚上的 Signal in 都装载到 Signal out 端后，再通过 Load*信号选通 Scan in。此时一条完整的扫描链就已经形成了，前一个 BS 单元的 Scan out 送入下一个 BS 单元的 Scan in，只需提供足够多的 Clock A 时钟脉冲，通过触发器的移位就可以将芯片引脚上的信号状态传送至芯片外部，这就是边界扫描技术的基本思想。以上是从外部读取芯片引脚上的信号，如果想要把外部的信号传入芯片内部也很方便，当第一个触发器将外部信号 Scan in 传入 Q 端时，提供 Clock B 时钟脉冲，将信号寄存移位到第二个触发器的 Q 端，通过多路选择器输出的 Signal out 即为输入的 Scan in 信号。由此可知，不论是芯片的输入引脚还是输出引脚，利用边界扫描链加载或读取芯片的数据原理是一样的，区别只是在于信号的流动方向上。

JTAG 边界扫描链除了用来检查单个芯片引脚的信号，确保单个元件具有正常功能外，还可以用来确保元件之间的正确互联关系。

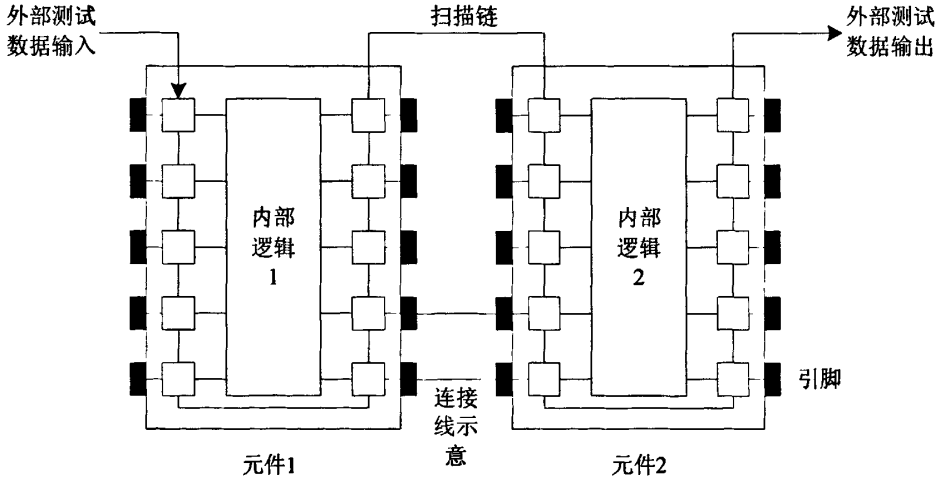

图 2-2 通过边界扫描链检查元件间互联关系示意图

如图 2-2 所示，为了检查元件 1 和元件 2 之间的电路连通性，需要对扫描链输入扫描向量，并进行一次性的移入和移出操作。扫描向量是在扫描测试过程中的一组特定的比特位。具体操作过程为：（1）给元件 1 准备一个扫描向量，将元件 1 与元件 2 相连的引脚处设置为 1，其余引脚设置为 0；（2）将此扫描向量串行移入，由于元件 1 和元件 2 互联，所以如果联通正常的话，元件 2 对应引脚应该出现高电平，否则依旧保持 0；（3）将元件 1 和元件 2 引脚信号串行移出，检查元件 2 对应引脚的电平状态，如果为 1 说明两个元件

的连接是正常的，否则出现异常。

边界扫描还可以用来测试整个电路，用来确保电路能够完成预期的逻辑功能。在这方面的运用还需要待测目标提供测试逻辑和预期测试结果，比较复杂。但还是建立在边界扫描的基本原理上进行的，此处不做赘述。

2.2 JTAG 接口控制器

JTAG 接口控制器由 3 部分组成，即测试端口（TAP，Test Access Port）控制器、指令寄存器（包括指令译码器）、数据寄存器。各部分关系如图 2-3 所示。

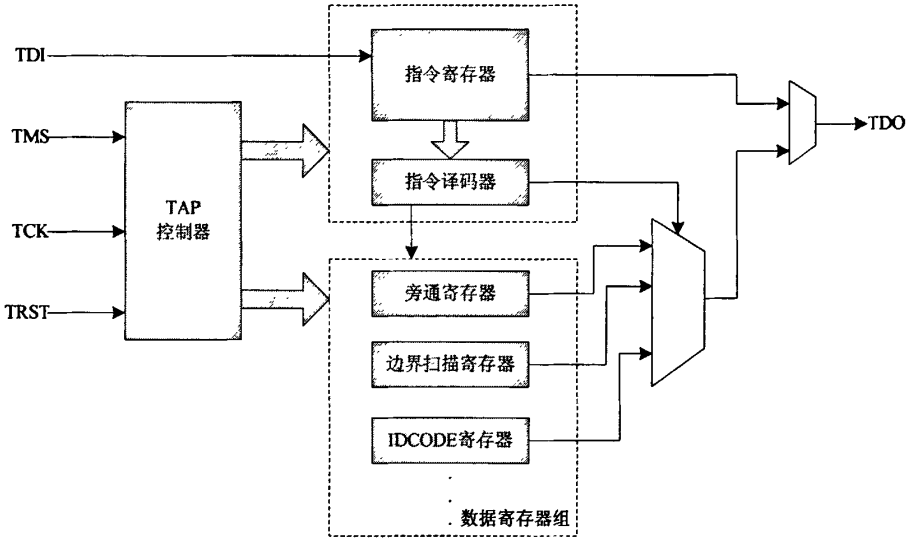

图 2-3 JTAG 接口控制器的结构

在上图中，JTAG 接口共有 5 条信号线，其中 TRST*测试逻辑复位（低电平有效）为可选信号线，其余各信号线为必须具备的。TCK 为测试时钟输入信号、TMS 为测试模式选择输入信号、TDI 为扫描向量串行输入、TDO 为扫描向量串行输出。其中，TCK 和 TMS 作为控制 TAP 控制器的信号，TAP 控制器是 JTAG 接口的核心，它通过一个有限状态机来完成外部信号的串行输入过程。指令寄存器包括指令译码器负责提供地址和控制信号去选择某个数据寄存器，也可以直接选通指令寄存器执行边界扫描测试，这时 TAP 控制器输出的选择信号选择指令寄存器驱动 TDO。上图中数据寄存器给出了旁通寄存器和边界扫描寄存器等。JTAG 标准规定必须具备的两个数据寄存器为旁通（Bypass）寄存器和边界扫描（Boundary Scan）寄存器，其他寄存器任选。指令寄存器和数据寄存器之间是并行的

关系，数据流通的路径由位于 TDI 和 TDO 之间的一个多路选择器进行选择。

2.2.1 测试访问端口及控制器

TAP 是用来访问芯片内部逻辑功能的通用端口，其连接定义如下表:

表 2-1 TAP 管脚连接定义

名称	I/O	功能描述
TCK	I	JTAG 标准强制要求。为芯片测试和 TAP 控制器提供独立的时钟信号，在板测试的各元件时钟均由它提供。
TMS	I	JTAG 标准强制要求。该信号在 TCK 的上升沿被采样，用来控制 TAP 控制器在各状态间的转换。
TDI	I	JTAG 标准强制要求。以 TCK 为驱动，将数据串行输入指令或者数据寄存器中。
TDO	O	JTAG 标准强制要求。以 TCK 为驱动，将数据从指令或者数据寄存器中串行输出。
TRST*	I	JTAG 标准中可选。低电平时有效，芯片进入正常工作状态，JTAG 测试逻辑无效。

表 2-1 给出了 5 条信号线的功能说明，这里为了便于理解，将对上述信号做进一步的备注说明。

- TCK

通常情况下，TCK 是个占空比为 50%的方波信号。在一个多功能系统中，存在着多种元件，为了使各元件串联成一条扫描链并统一工作，JTAG 标准强制要求了提供时钟 TCK。该信号确保了测试数据在不改变片上逻辑状态的情况下移入和移出芯片，它一般是经宿主机的并口引出，所以频率并不是很高。

- TMS

TAP 控制器将在 TCK 的上升沿采样 TMS 信号，经解码后用于控制操作。TMS 用于一个系统中多个元件的测试驱动，所以该引脚上的负载连接不能太多。JTAG 标准规定即使在 TMS 没有信号连通时，JTAG 控制器的测试逻辑状态不变。这就说明即使 TMS 引脚悬空，测试逻辑状态还是保持在工作前的初始状态，即 Test-Logic-Reset 状态，在点评兼容的电路设计中，TMS 引脚上经常加上一个上拉电阻以保证高电平。

➢ TDI

TDI 在 TCK 上升沿被采样进入内部指令或数据寄存器，JTAG 标准要求在数据由 TDI 传送至 TDO 过程中不得发生跳变翻转，这就保证了串行输入、输出的一致性。

➢ TDO

TDO 作为串行输出同样是借助 TCK 的驱动，为了避免操作时产生冲突，该信号是在 TCK 的下降沿被采样的。

➢ TRST*

该信号主要提供测试逻辑复位的辅助功能。它可以在测试逻辑处于工作状态时异步初始化 TAP 控制器，使其回到初始等待状态。因此，该信号可以使测试逻辑脱离芯片系统逻辑而独立复位。

TAP Controller 主要用来实现对边界扫描链的控制，具体是通过一个同步有限状态机实现的。状态机中每一个状态都有其特定的功能和意义，它们之间的转换是随着 TCK 和 TMS 信号变化而产生的。因此，宿主机端的软件发出一个特定的时序序列就可以产生一个对应的操作。该状态机是 TAP 控制器的核心，共有 16 个状态组成，如图 2-4 所示。图中的每个圆角矩形代表一个状态，各个状态间的转换由 TCK 驱动，经 TMS 控制。按照图中各状态所处位置，可从左到右按照纵向划分为三列。其中第二列和第三列的状态除了 DR 和 IR 的区别外，其余几乎是对应的，包括转换条件都是一样的。所以该状态机并不复杂。在这 16 个状态中，只有 6 个状态是稳定状态，分别是测试逻辑复位（Test Logic Reset）、测试/空闲状态（Run Test Idle）、数据寄存器移位（Shift DR）、指令寄存器移位（Shift IR）、数据寄存器暂停（Pause DR）、指令寄存器暂停（Pause IR）。其他状态都是不稳定的状态。在系统重启或正常运行时，应提供 TMS 信号 5 个 TCK 周期以上的高电平，使测试逻辑复位。此时的 TAP 将不影响系统其他各元件的工作状态。若要进入边界扫描或调试状态，可以根据 TMS 和 TCK 的配合使 TAP 进入工作状态。具体说明如下:

➢ Test Logic Reset

该状态是系统上电后自动进入的状态。此状态下的测试逻辑功能全部禁用，用以确保芯片核心逻辑的正常工作。采用 TRST*低电平和保持 TMS5 个以上高电平都可以让 TAP 控制器进入该状态。该状态属于稳定状态，当 TMS 由于干扰或者毛刺导致出现一个低电

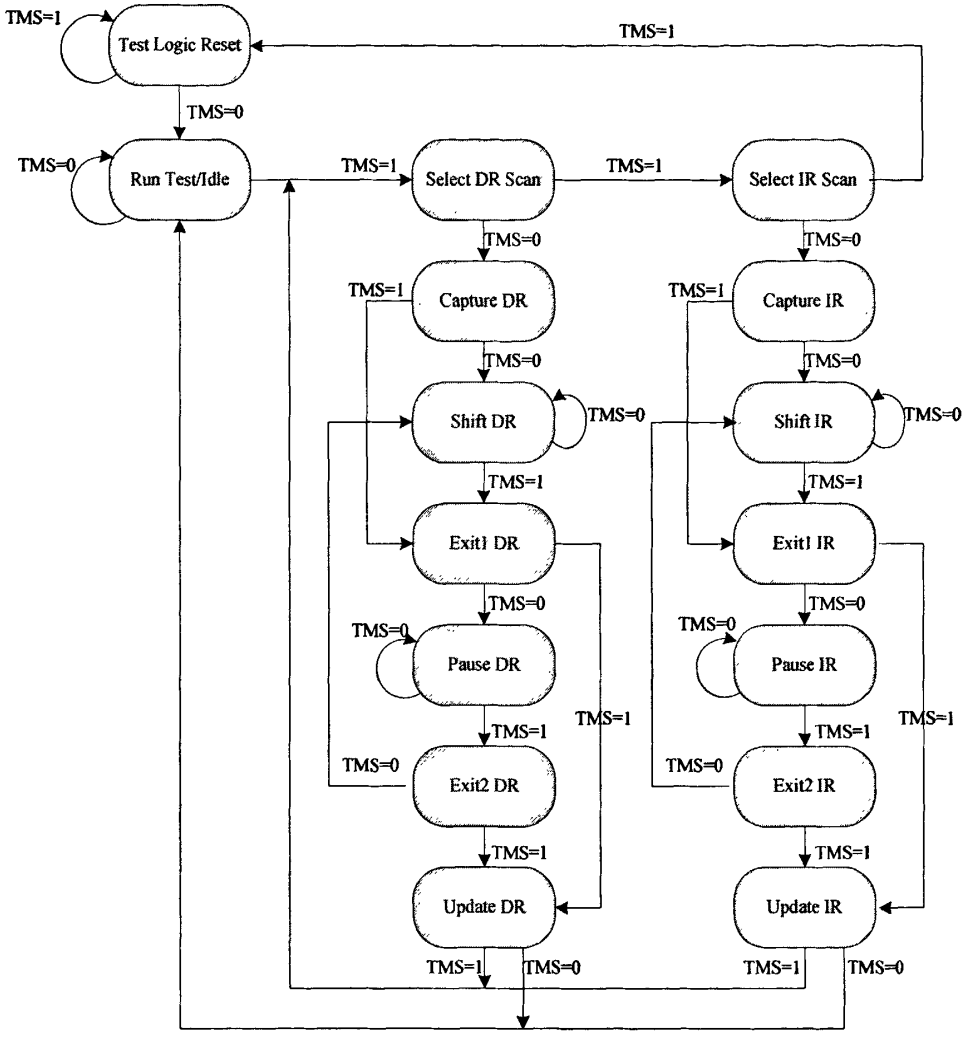

图 2-4 TAP 控制器状态机状态转换图

平后，只要 TMS 能继续保持原先高电平，TAP 控制器将在 3 个 TCK 周期后回到 Test Logic Reset 状态。

- Run Test/Idle

此状态是一个中间状态。根据指令寄存器中的指令不同，该状态下可能会执行一些操作，也可能什么操作都不进行，这正好符合了该状态的命名。当 TMS 信号变 1 后，经 TCK 上升沿采样，TAP 将进入 Select DR Scan 状态。

- Select DR Scan

此状态是一个中间状态。当 TMS 为 0，状态机进入 Capture DR 状态，后续的操作都建立在数据寄存器上；当 TMS 为 1，状态机进入 Select IR Scan 状态。

➢ Select IR Scan

此状态是一个中间状态。当 TMS 为 0，状态机进入 Capture IR 状态，后续的操作都建立在指令寄存器上；当 TMS 为 1，状态机进入 Test Logic Reset 状态。

➢ Capture DR

此状态是一个中间状态。在 TCK 上升沿，芯片输出管脚上的信号被捕获。当 TMS 为 0，状态机进入 Shift DR 状态，TMS 为 1 则进入 Exit1 DR 状态。

➢ Shift DR

此状态为稳定状态。在 TCK 的驱动下，每过一个时钟周期，就有一位 TDI 管脚的数据串行输入，如果 TMS 保持 0，则一直串行移入，当然输入数据的长度也取决于寄存器的长度；如果 TMS 变为 1，状态机跳入 Exit1 DR 状态。

➢ Exit1 DR

此状态是一个中间状态。当 TMS 为 1 并且 TCK 上升沿，状态机进入 Update DR 状态并结束扫描过程；当 TMS 为 0，则状态机进入 Pause DR 状态，并保持指令寄存器中的指令不变。

➢ Pause DR

此状态为稳定状态。串行输入的数据在此状态下将停止移位，并且被当前指令选中的数据寄存器将保持原先状态不变。直到 TMS 为 1 并且被 TCK 上升沿采样后进入 Exit2 DR 状态。

➢ Exit2 DR

此状态是一个中间状态。当 TMS 为 1 经 TCK 上升沿采样，状态机将进入 Update DR 状态，否则进入 Shift DR 状态。

➢ Update DR

此状态是一个中间状态。由 TCK 的上升沿驱动，移位寄存器中的数据将被加载到对应的数据寄存器中或芯片对应的管脚上。有的数据寄存器还带有锁存输出，为了避免移位数据并行输出时的跳变。这种做法往往是在 TCK 的下降沿将移位寄存器中的数据先锁存到数据寄存器中，然后等 TCK 上升沿到来时再加载到相应的元件引脚上。

➢ Capture IR

此状态是一个中间状态。在 TCK 的上升沿，一个特定的逻辑序列将被装载进指令寄存器中去。此状态下，被选中的数据寄存器将保持原先状态不变。

➢ Shift IR

此状态为稳定状态。具体工作过程如 Shift DR 状态。只是具体移位输入的内容为指令而已。

- Exit1 IR

此状态是一个中间状态。当 TMS 为 1 并且 TCK 上升沿，状态机进入 Update IR 状态并结束扫描过程；当 TMS 为 0，则状态机进入 Pause IR 状态。

- Pause IR

此状态为稳定状态。指令寄存器的移位操作被暂停，直到 TMS 为 1 并且 TCK 为上升沿时，状态机进入 Exit2 IR 状态。

- Exit2 IR

此状态是一个中间状态。当 TMS 为 1 经 TCK 上升沿采样，状态机将进入 Update IR 状态，否则进入 Shift IR 状态。

- Update IR

此状态是一个中间状态。在 TCK 的下降沿到来时，串行输入的指令将被锁存更新到并行输出上。一旦新的指令锁存成功，该指令即成为当前指令。之后，若 TMS 为 1 并且 TCK 上升沿采样，状态机将进入 Select DR Scan 状态；否则进入 Run Test/Idle 状态。

以上是对 TAP 控制器状态机各个状态的详细说明。关于指令寄存器和数据寄存器的介绍和访问过程将在下面展开。

2.2.2 指令寄存器

每个 JTAG 接口控制器中都含有一个指令寄存器，通过它可以选择要执行的命令操作，也可以用来选择某个数据寄存器。指令寄存器是基于串行移位寄存器设计的，当指令移入完毕后将被锁存更新。在输入的指令中，一些是 JTAG 标准给出的公共指令（public），另一些是用户厂商根据自己的需求定义的私有指令（private）。在公共指令中，JTAG 标准还规定四条强制指令，即所有遵循 JTAG 标准的元件中必须包括的指令。它们是 BYPASS、SAMPLE、PRELOAD 和 EXTEST 指令。

- BYPASS 指令

BYPASS 指令主要有两个作用。一个就是用来将 BYPASS 寄存器接入 TDI 和 TDO 之间，从而使测试数据可以快速地从扫描链中通过。这样做的好处是减少由于不必要的串行扫描所耽误的时间。JTAG 的时钟 TCK 一般由宿主机提供，频率也不是很高，假设多个元

件一起接在扫描链上，共有上百个 BS 单元，那么传送 1bit 的数据到 TDO 就要移位 100 个周期，如果是对存储性器件（如 Flash）进行编程，那么将耗用大量的时间，效率低下。而 BYPASS 正是可以将不必要的链路旁通，“加快”移动速度。当然也可以定义几条子扫描链，操作时直接选中即可。BYPASS 指令的另一个作用是确保芯片核心逻辑的正常工作状态。在芯片没有提供 ID 寄存器的情况下，BYPASS 指令会在 Test Logic Reset 状态被强制锁存到指令寄存器中，这样就保证了芯片在任何工作状态时扫描链均是完整的。

- SAMPLE 指令

该指令主要用来捕获芯片正常工作时引脚上的状态，如需查看此时状态，只需将捕获的数据串行输出至 TDO 即可。当 SAMPLE 为当前指令时，其他的数据寄存器不能和边界扫描寄存器串联，从芯片内部传输的信号和外部经过芯片引脚的信号都将在 Capture DR 状态下装载进边界扫描链。

- PRELOAD 指令

该指令主要用来将移位寄存器中的数据在 Update DR 状态时并行锁存到对应的寄存器中去。可见该指令的操作正好和 SAMPLE 指令相反。当 PRELOAD 为当前指令时，同样也只能将边界扫描寄存器串行地接入 TDI 和 TDO 之间。

- EXTEST 指令

该指令用来实现板级的互联测试。一般地，在选择该指令前都需要通过指令 PRELOAD 将数据加载到寄存器的并行输出上。测试时，输入、输出引脚的 BS 单元分别用于捕捉测试结果和产生测试激励。

上面就是 JTAG 标准强制要求的指令，除了这些指令外，JTAG 标准还提供了一些可选指令，比如 INTEST 指令（内建静态测试指令）、RUNBIST（执行元件自检指令）、IDCODE 指令（设备码指令）、CLAMP 指令（扫描链决定外驱动信号的指令）等等。这里不做详细的介绍。

2.2.3 数据寄存器

JTAG 标准规定必须具备的两个数据寄存器是边界扫描寄存器（Boundary Scan Register）和旁通寄存器（BYPASS Register），其他寄存器任选。测试、调试时由指令寄存器选定某个数据寄存器作为扫描测试寄存器，未被选中的数据寄存器处于高阻状态。

- 边界扫描寄存器（Boundary Scan Register）

它是由围绕IC管脚的一系列BS单元组成的，通过这条串行地扫描链来实现测试管脚信号的输入和输出。边界扫描链是JTAG标准中最复杂的部分，因为它要完成对芯片核心逻辑的访问，响应外部指令并做出相应操作，同时还要兼顾各个元件间的联系，避免冲突。在设计边界扫描链时要考虑信号的缓冲、隔离、以及驱动能力等方面的问题，有的甚至已经超出了逻辑电路的范畴。

- 旁通寄存器（BYPASS Register）

旁通寄存器是由一个扫描寄存器位组成，当选择旁通寄存器后，TDI和TDO之间只有一位寄存器，此时并不是执行扫描测试。对于一个系统而言，连接在扫描链上的元件可能有很多，但有时测试只需要对其中某个元件进行测试，此时可以利用旁通寄存器配合BYPASS指令将不需要进行测试的芯片旁通，缩短扫描路径，提高测试效率。

- 可选寄存器

除了JTAG标准强制规定的数据寄存器外，用户可以根据需要，在指令不冲突的情况下自行定制新的数据寄存器，并不做公开说明。这就给JTAG标准的扩充和广泛的应用提供了极大的可能。比如ID寄存器的使用，使得通过TAP控制器访问芯片厂商、元件编号、版本成为可能。事实上，这部分应用已经相当的广泛，尤其是在嵌入式硬件调试方面得到了很好的发展。根据特定处理器的设计，以及整体SoC系统功能的需要，功能多样的调试扫描链应运而生。

2.3 CK510的片上调试技术

CK510是一款拥有自主知识产权的高性能国产32位嵌入式处理器，具有可扩展指令，可配置硬件资源，可重新综合，易于集成等优点，已较广地应用于嵌入式控制系统和电池供电的便携式产品之中。和其他32位嵌入式处理器一样，CK510拥有片上调试逻辑，其各项功能是通过片内仿真模块HAD(Hardware Assisted Debug)来具体实现的。

2.3.1 HAD的功能介绍

片上仿真模块（HAD）可以使外围设备在不干扰处理器执行的情况下了解CK510处理器的当前状态信息。用户通过它可以了解CK510的寄存器和存储器的内容，以及其他片内设备，方便用户在CK510处理器的基础上进行硬件/软件的开发。HAD通过一个标准的工业JTAG接口来读取片上调试逻辑内定义的寄存器。HAD模块是完全可以集成的，

CK510 的 JTAG TAP 接口和辅助逻辑电路和标准的 JTAG 电路时兼容的，可以同已有的 JTAG 部件或独立设计额外功能的控制器集成在一起。HAD 模块具有以下特点：

- 通过标准的 JTAG 接口进行调试
- 可以设置 2 个内存硬断点，也可以设置软断点
- 检查改变寄存器值
- 检查改变内存值
- 贮存 8 条可供读取的已执行的跳转指令的地址
- 指令单步或多步执行
- 可以在复位后或在普通用户模式进入调试模式
- 通过 JTAG 读写 HAD 寄存器控制 HAD 完成相应操作

HAD 控制逻辑通过扫描链输入/输出指令和数据，对 CK510 的寄存器和存储地址中的内容进行存取；通过处理器执行扫描输入的指令来获取需要的数据；通过扫描输入装载指令，执行装载指令获取指令所指定具体地址的内容，然后再通过扫描输出。HAD 模块中的逻辑电路不要求处理器中断正常的运行来执行 HAD，更不会和 CK510 的正常运行发生冲突。其中对处理器之外的特定逻辑电路进行存取，如 PC 指针缓存器和计数寄存器可能会要求处理器停止执行以避免同步的冒险竞争。调试程序通过读取状态寄存器来获知处理器所处的状态来确保安全性，其余所有的调试操作都在处理器处于调试模式下进行。

HAD 模块不另外设立外部数据输入/输出的外部管脚，它的功能是通过 JTAG 的外部接口向 HAD 的控制逻辑传送指令和数据实现的。如果一些功能的实现需要用到处理器中的资源，HAD 控制逻辑会向处理器发出一个调试请求，处理器在执行完当前指令后，保存流水线的信息，然后进入调试模式等待下一次命令。HAD 模块与外部通信的接口信号中，除了 JTAG 接口信号外，还有外部调试请求（i_jdb_req_b）、处理器调试请求（i_cpu_dbgrq_b）、处理器调试响应（iu_yy_xx_dbgon）和处理器断点设置请求（i_cpu_brkrq_b[1:0]）。

- 外部调试请求（i_jdb_req_b）

该信号在被送到处理器之前要同调试模式下的处理器的时钟同步，调试请求要至少保持两个系统时钟周期。

- 处理器调试请求（i_cpu_dbgrq_b）

该信号是调试模块和处理器之间的内部信号，当它有效时，HAD 模块要求处理器进入调试模式，根据不同情况可有多种方法来驱动该信号。

➢ 处理器调试响应（iu_yy_xx_dbgon）

该信号是处理器内部信号，属于调试控制逻辑和处理器之间握手信号的一部分，当CPU进入调试模式前使该信号有效。

➢ 处理器断点设置请求（i_cpu_brkrq_b[1:0]）

该信号位可由外部控制逻辑来驱动，表示当前处理器总线的存取操作中包括了断点。此时处理器进入调试模式或者初始化断点异常处理程序。

2.3.2 HAD的控制逻辑

HAD控制逻辑由命令寄存器，译码器和状态/控制寄存器组成。工作时，HAD的命令寄存器HACR作为TAP控制器的指令寄存器（IR），其他各类功能的调试寄存器都作为数据寄存器（DR）。当TAP控制器在Shift IR状态时，指令通过扫描链串行输入，当处于Update IR时，指令载入指令寄存器。当TAP控制器处于Capture DR，Shift DR和Update DR时，命令寄存器将选择HAD中的某个数据寄存器作为对象进行存取操作。

➢ 命令寄存器HACR

命令寄存器中保存了HAD中数据寄存器对象列表，可以控制存取的对象，同时也可以控制执行单步操作和退出调试模式。虽然HACR在Update IR状态进行指令更新，但是只有在TAP控制器状态转换到对数据寄存器（DR）进行扫描时才对选中的具体对象进行存取，然后在Update DR时完成存取。此外，即使仅执行了退出调试模式的时候或者没有要求具体的存取对象时也必须让TAP控制器转到Update DR状态。该寄存器是个10位的寄存器，如表2-2所示：

表2-2 HAD命令寄存器

9	8	7	6	5	4 0
DEBUG	MBRUN	R/W	GO	EX	RS4-RS0

DEBUG-内存BIST的调试指令。该指令用来确认移出从内存BIST得来的错误数据和地址，并移入新的地址用以进行新的测试。

MBRUN-运行内存BIST指令。用来使能用户选择的BIST算法。

R/W-读写命令位，用来表明当前数据流向。若为0，则数据写入RS[4:0]指定的寄存器中；若为1，则将其中数据读出。

GO-执行命令位。若该位被设置为1，则执行CPU扫描链寄存器中IR的指令。

EX-退出调试模式控制位。0 保持调试模式；1 退出调试模式。

RS[4:0]-寄存器选择位。它指定了读/写操作所要对应的寄存器，就好像一本书的目录一样，将每个数据寄存器都对应到相应的地址“页”。其中与调试器编程操作相关的数据寄存器以及新增的数据寄存器将在后续硬件设计章节介绍。

➢ 控制寄存器 HCR

HCR 是 32 位的寄存器用来选择驱动事件使 CK510 进入调试模式并对 HAD 中各逻辑功能进行控制，该寄存器是可读可写的。

表 2-3 HAD 控制寄存器

31 20	19	18	17 16	15	14	13	12	11	10 6	5	4 0
Reserved	WPB	WPA	SQC[1:0]	DR	IDRE	TME	FRZC	RCB	BCB[4:0]	RCA	BCA[4:0]

WPA,WPB-观察点 A,B 控制位。用来决定是否有一个观察点或断点产生给内存断点 A 或 B 逻辑。0 申请断点；1 申请观察点。

SQC[1:0]-连续操作控制位。允许存储器断点和追踪发生的状态被保持直到某个特定的时间产生。该控制位为两位可以产生 4 种控制逻辑。

DR-处理器调试请求控制位。用来要求处理器无条件进入调试模式。

IDRE-内部调试请求允许位。用来有效允许内部产生调试请求。

TME-追踪模式控制位。用来允许 HAD 进行追踪操作。

FRZC-冻结控制位。若为 0 则断点 B 发出断点请求；若为 1 则是 PC 缓存器冻结，不再更新发出新的请求。

RCA,RCB-存储器断点范围控制位。用来控制存储器断点的设置地址范围。

BCA[4:0],BCB[4:0]-存储器断点控制位。这些控制位控制允许断点的设置，限定对于断点存取的特性，判断对于读，写和指令译码的存取操作是否会产生断点吻合的信号。

➢ 状态寄存器 HSR

HSR 是一个 16 位的寄存器，用来显示驱动进入调试模式的事件来源和当前处理器所在的操作模式，各状态位均为只读的。

表 2-4 HAD 状态寄存器

15 11	10	9	8	7	6	5	4	3	2	1 0
Reserved	EBRO	HDRO	DRO	MBO	SWO	TO	FRZO	SQB	SQA	PM[1:0]

EBRO-外部断点请求状态位。响应有无外部断点请求。

HDRO-硬件调试请求状态位。响应硬件有无调试请求。

DRO-调试请求发生状态位。响应 DR 位有效调试请求进入调试模式时被设置的状态。

MBO-存储器断点请求状态位。响应是否产生了内存断点调试请求。

SWO-软件调试请求状态位。响应是否有断点调试指令被执行。

TO-追踪调试请求状态位。响应追踪计数器的值是否为 0。

FRZO-FIFO 缓存冻结发生状态位。响应是否有 FIFO 缓存发生冻结。

SQA,SQB-断点连续发生 A，B 状态位。响应是否有内存断点 A 或 B 的产生。

PM[1:0]-处理器模式位。用来显示当前处理所处的模式，包括正常模式、休眠模式、调试模式。

2.3.3 断点逻辑和追踪逻辑

➢ 断点逻辑

存储器的断点可以设置为指定的一个地址，或者是一个地址范围内。断点控制逻辑包括一个输入的锁存器，该锁存器保存着起始地址和掩膜地址，以及地址比较器，断点计数器，存取特性判断等逻辑。如图 2-5 所示。

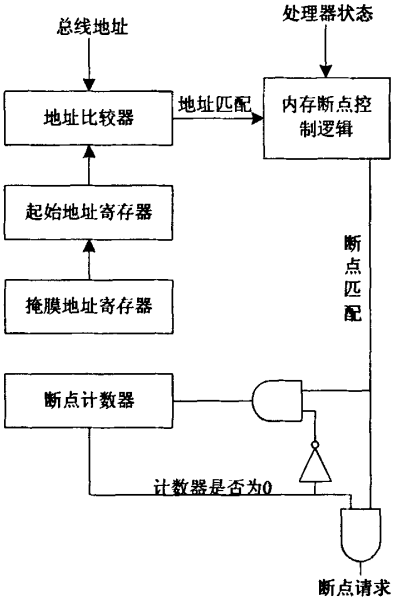

图 2-5 存储器断点逻辑结构

如图 2-5，当总线上的地址和保存在起始地址寄存器中的地址经过掩膜寄存器中的掩

膜位比较得到的地址吻合，地址比较器会产生一个地址匹配的信号。随后，内存断点控制逻辑中的特定位产生控制存取的信号。如果断点计数器的当前值不为 0 时，地址吻合信号用来控制断点计数器的计数，如果计数器的值为 0，产生断点调试请求。

➢ 追踪逻辑

HAD 追踪逻辑允许用户在返回调试模式前单步或者多步执行指令，或者等待外部接口传送的执行命令。它可以帮助用户即使调试一段关键程序，或者统计一段程序中执行的指令条数。追踪逻辑是独立于处理器之外的，由状态寄存器 HSR 中的追踪模式位来控制。其逻辑结构图如图 2-6 所示。

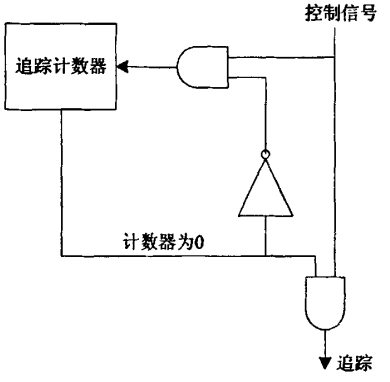

图 2-6 追踪逻辑结构

追踪逻辑使用方法是在追踪程序初始化时，将一个非零的数据写入追踪计数器，并且将当前要执行的指令以及操作地址存入 CPU 扫描链寄存器，然后令 CPU 退出调试模式执行。此后，每执行一条指令（允许中断发生），计数器的值就减 1，当计数器的值减到 0 时，HAD 的控制逻辑会发出调试请求，要求重新回到调试模式。

2.3.4 流水线信息和回写总线寄存器

CPU 的流水线信息保存在内部的寄存器里，一般通过调试扫描链读写这些信息。在 HAD 中 CPUSCR（CPU 扫描链寄存器）存储着处理器跳入调试模式时的信息，以便于处理器返回正常模式时能够继续正常的操作。同时 CPUSCR 也是软件模拟调试时读写处理器和内存的物理媒介。其结构图如图 2-7 所示。

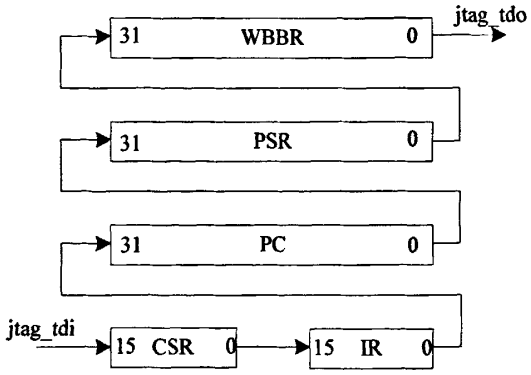

图 2-7 CPU 扫描链寄存器

➢ 指令寄存器 IR

该寄存器主要是为调试模式的 CPU 服务的，用户可以输入任意的 CPU 指令，并控制这些指令的执行，进而用来检测、改变内存和处理器内部寄存器的内容，这种调试方式是单步执行的。退出调试模式后，指令寄存器中的指令不再被执行。

➢ 控制状态寄存器 CSR

CSR 是 16 位寄存器，保存着 CPU 进入调试模式时部分内部状态变量的值。在调试过程中，CSR 的值是可变的，但返回正常工作模式后，必须先把保存好的 CSR 的值重新赋值回去。除了保存内部状态位，CSR 还设计了调试时专用的两个控制位 FDB,FFY 位。分别为是否允许进入调试模式和是否将 WBBR 中的内容作为某操作数的值。

➢ 处理器状态寄存器 PSR

PSR 是 32 位的读写寄存器。当外部控制器需要保存或者修改处理器状态的时候，就可以用该寄存器来实现。调试时，该寄存器的内容会发生变化，如需跳出调试模式后用某值重置该寄存器，应事先做好备份工作。

➢ 程序指针寄存器 PC

PC 是 32 位的寄存器，存储着进入调试模式时 CPU 下一条还没有处理执行的指令对应的地址，在调试模式时可以改变该寄存器的值，所以如果要返回原来正常工作模式的状态，需要将事先保存的 PC 原值再次存储进 PC 寄存器。

➢ 回写总线寄存器 WBBR

WBBR 是外部控制器传送操作数信息给 CPU 的媒介。如果需要读/写内部寄存器和存储器（如 Flash）的值，那么就可以通过执行指令把该值送到 WBBR。比如编程 Flash 过程中更新内部寄存器的环节，可以先赋值给 WBBR，然后执行一条 mov r_x,r_x，那么 WBBR

的值就会作为操作数送到 mov 指向的寄存器中。

2.3.5 调试环境下的控制操作

在任何调试操作前，HAD 必须待 CPU 进入调试模式的确认信号产生后才发送指令和数据。HAD 命令可归类为：读命令、写命令和没有数据读写的命令三类；数据的长度由 16 到 128 位不等。命令和数据都是从低位开始传送的，当 HAD 传送一条命令或数据完毕后，必须等到 CPU 发回确定执行的信号后再送一条命令和数据。所以，对于一个确定的目标系统，HAD 就成了该系统和宿主机之间的媒介，用来控制串行接口驱动、通信并解析宿主机的命令。为了很好的控制目标系统，用户通过 HAD 需要执行下列操作，当然这也是一般嵌入式系统调试过程中的必备操作。

- 通过复位信号或用户定义模式进入调试模式
- 读写片上调试仿真器内寄存器
- 读写 CPU 内部寄存器
- 读写片内存储器
- 设置硬断点/软断点
- 执行单步/多步指令
- 退出调试模式
- 检测系统是否进入调试模式

3 高性能片上调试编程器的设计原理

本章以 JTAG 接口技术为基础，分析通过 JTAG 接口实现 Flash 编程需满足的条件，指出不同编程方法间的区别。在分析总结原有编程方法的基础上，提出快速可重构的 Flash 编程原理。

3.1 Flash 编程过程分析

一般地，通过 JTAG 边界扫描进行 Flash 编程要满足两个基本条件：1. 需要 Flash 器件支持边界扫描功能；2. 数据串行移位操作需要满足 Flash 读写时序的设计。所以对 Flash 进行编程需要 Flash 本身支持边界扫描测试，并且带有 BS 单元，或者将 Flash 的地址线、数据线、控制线间接地连接到带有 BS 器件的芯片上。条件 2 说明当 BS 单元装载好数据后，需要通过特定的指令将数据传送给 Flash 完成编程。每次移入一位数据所需的周期数（TCK）和具有的边界扫描单元数相当。同时，不同的 Flash 器件的操作时序均有所不同，这导致了控制时序的指令传输的周期数有所不同，对于某些具有较复杂时序的 Flash 器件，完成每个地址单元的编程操作必须经过多次的扫描刷新，其编程时所需的周期数（TCK）也相应增多。此外，编程的数据宽度也影响着编程。比如对内容大小相同的两段数据进行下载编程，8bit 的数据宽度要比 16bit 的数据宽度多耗用一倍的编程时间。由此可知，边界扫描链的长度、Flash 器件的操作时序的复杂程度、编程数据宽度以及 TCK 的频率高低，都不同程度地影响了 Flash 编程所需的时间，进而影响到调试效率。

片上调试模块可以使外围设备在不干扰处理器执行的情况下了解处理器内部当前的状态信息，以及其他片内设备信息。借助片上仿真模块编程 Flash 属于调试编程的范畴，它主要借助对片内 CPU 通用寄存器的数据加载和对 CPU 的指令控制来实现对 SRAM、Flash 等器件的编程。一般对于单个芯片而言，其调试扫描链的长度要小于整个板级系统的边界扫描链的长度，数据流通路径的缩短使得编程时间有所减短。另外，对于特定嵌入式系统的 CPU，都有固定的指令集，这样就可以通过外部向系统输入指令灵活地控制 CPU 的运行，完成对数据的编程工作。通过片上调试模块编程 Flash 的基本工作就是写内存。通过片上调试模块向内存地址写一个字的流程如图 3-1（R0，R1 为 CPU 的通用寄存器）：

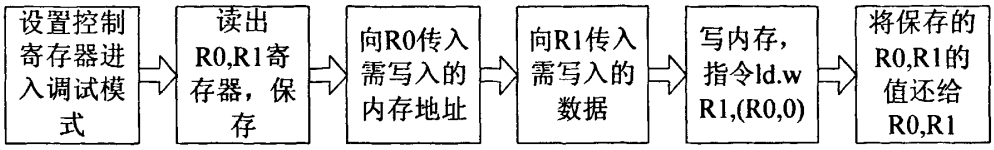

图 3-1 调试模式下写内存

一般对于初始编程，步骤 1 只需执行一次即可，步骤 2，6 也不需要，所以通过片上仿真器编写内存主要就是 3、4、5 三步，即传地址、传数据、执行 CPU。编程 Flash 又是通过多次编写内存来完成的。不同类型 Flash 的编程原理、编程方式、时序操作均有所不同，这里选取具有代表性的两类 Nor Flash 和 Nand Flash 进行分析。

Nor Flash 方面，以 SPANSION 公司的 S29GL-N 系列为例[18]，该类 Flash 支持两种编程模式，分别为按字节编程（×8）和按字编程（×16），其编程命令序列如表 3-1 所示：

表 3-1 编程命令序列

Command Sequence (Note)	Cycles	Bus Cycles							
		First		Second		Third		Fourth	
		Addr	Data	Addr	Data	Addr	Data	Addr	Data
Program（×8）	4	AAA	AA	555	55	AAA	A0	PA	PD
Program（×16）	4	555	AA	2AA	55	555	A0	PA	PD

表 3-1 中，Cycles 代表单次编程所需周期数，Addr 代表地址，Data 代表数据，PA 代表编程地址动作、PD 为编程数据动作，一共经历 4 个总线周期完成一次数据编程操作。其中 First、Second 是 Flash 解锁命令周期，Third 是 Flash 编程建立周期，Fourth 则是写入要编程的目标地址和数据，至此完成一次编程操作。可见，无论是×8 模式还是×16 模式编程，都需要 4 个编程命令周期的过程，只是每个周期中操作的地址和数据不同而已。并且前三个周期均是向固定的地址写入固定的数据，目的是完成 Flash 编程的初始化，只有第四周期编写数据才是真正有效数据。以×16 模式为例，其最后两个周期具体写入数据的时序图如图 3-2 所示：

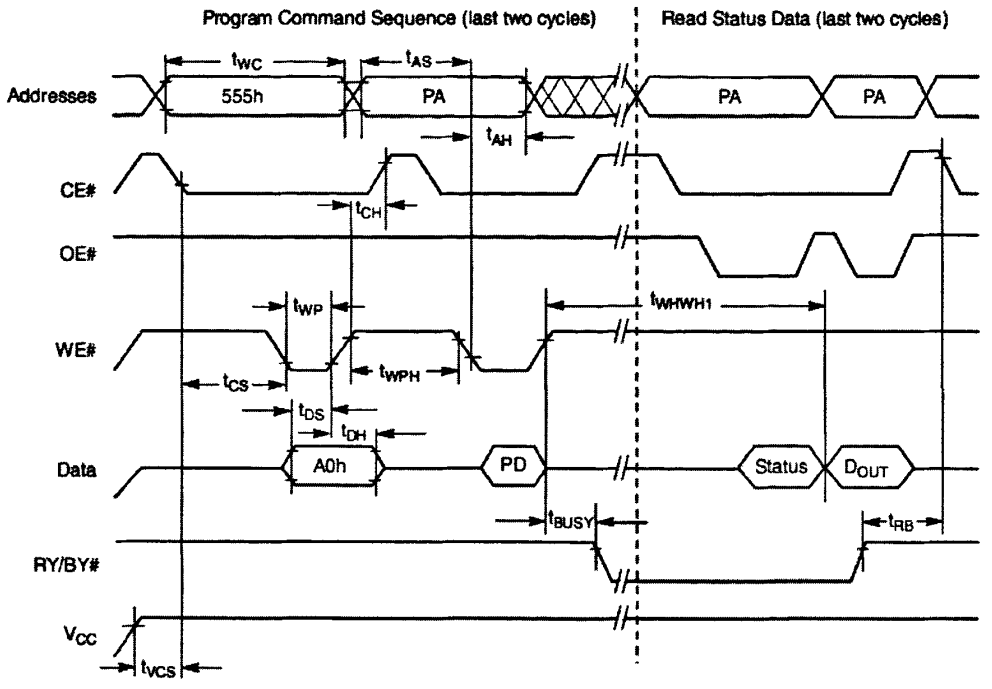

图 3-2 ×16 模式编程操作时序

在上图中，通过 CE#和 WE#两根信号线控制地址和数据的锁存。当编程地址时，地址锁存发生在 WE#的下降沿或者是当一个 CE#脉冲到来的时候锁存，取决于最迟到达的信号。如上图中 PA 在 CE#脉冲到来时继续保持，直到 WE#下降沿到来完成锁存，开始进行数据编程 PD。数据的锁存同样受这两根信号线的控制。不同的是，数据锁存发生在 WE#的上升沿或者 CE#出现脉冲时，并且取决于谁先发生。如图中 PD 在 WE#出现上升沿后即被锁存。经过一段时间，真实的数据 D_{OUT} 出现在实际对应的编程地址上，实现了数据的编程载入。其编程操作流程图如图 3-3[18]所示。

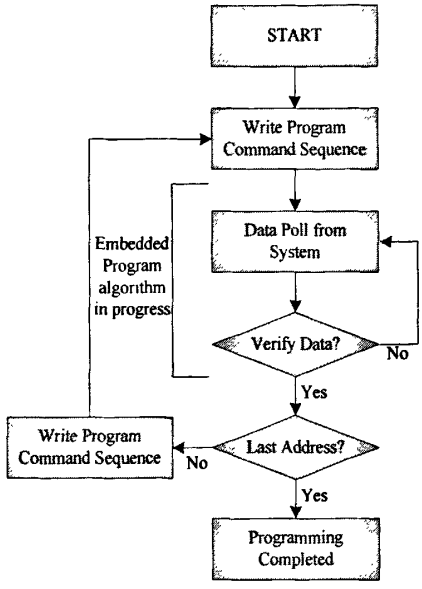

图 3-3 S29GL-N Nor Flash 编程操作

利用 CK510 片上仿真模块实现上述编程过程的核心汇编程序如下:

```
        lrw     r4,start_address        //1 设置首地址
        lrw     r6,end_address          //2 设置末地址
        lrw     r1,0x2000aaaa           //3 命令寄存器地址 1
        lrw     r2,0x20005555           //4 命令寄存器地址 2
        lrw     r11,0xaa                //5 第 1 编程周期命令
        lrw     r9,0x55                 //6 第 2 编程周期命令
        lrw     r12,0xa0                //7 第 3 编程周期命令
PRO:    sth     r11,(r1)                //8 写第 1 周期命名
        sth     r9,(r2)                 //9 写第 2 周期命令
        sth     r12,(r1)                //10 写第 3 周期命令
        lrw     r5,data                 //11 将数据存入 r5 寄存器
        sth     r5,(r4)                 //12 写入数据
        addi    r4,0x2                  //13 地址加 2
        cmpne   r4,r6                   //14 判断是否到达末地址
```

当然，设计 Flash 编程的程序可以有多种，并且可以根据用户的需要在程序中穿插比较、判断和监视等不同指令。为了说明问题，这里给出了 Flash 编程的核心程序，包括调试过程中保存 R0、R1 寄存器值的步骤也省略了。程序的前部分为编程的初始化过程，将

Flash 编程中时序操作需求的地址和数据先暂存在 CPU 特定的通用寄存器中。PRO 部分为 Flash 编程的主循环程序，主要是执行 CPU 完成 Flash 时序操作要求中对特定地址寄存器的加载以及对真正有效数据的编程。在这其中只有第 11、12 行和数据传输有关，为每次都更新内容的“有效”指令，其余均为重复的冗余指令。

Nand Flash 方面，以 SAMSUNG 的 K9F3208W0A-TCS0 型号[19]为例。和 Nor Flash 一样，Nand Flash 也需要先做时序操作，不同于 Nor Flash 的按字或字节的编程方式，Nand Flash 是以页为单位进行读和编程操作的，单页的数据量为 512 字节（目前还有单页数据量达到 2KB，甚至 8KB 的 Nand Flash）；以块为单位进行擦除操作，一块为 4KB、8KB 或者随着单页数据量的增大而增大。编程时，先对单页地址进行数据编程，结束后再进入下一个地址页进行编程。Nand Flash 的容量较大，一般用于大容量静态数据存储，其写入速度也较 Nor Flash 快。图 3-4 是 Nand Flash 的页编程操作时序图。

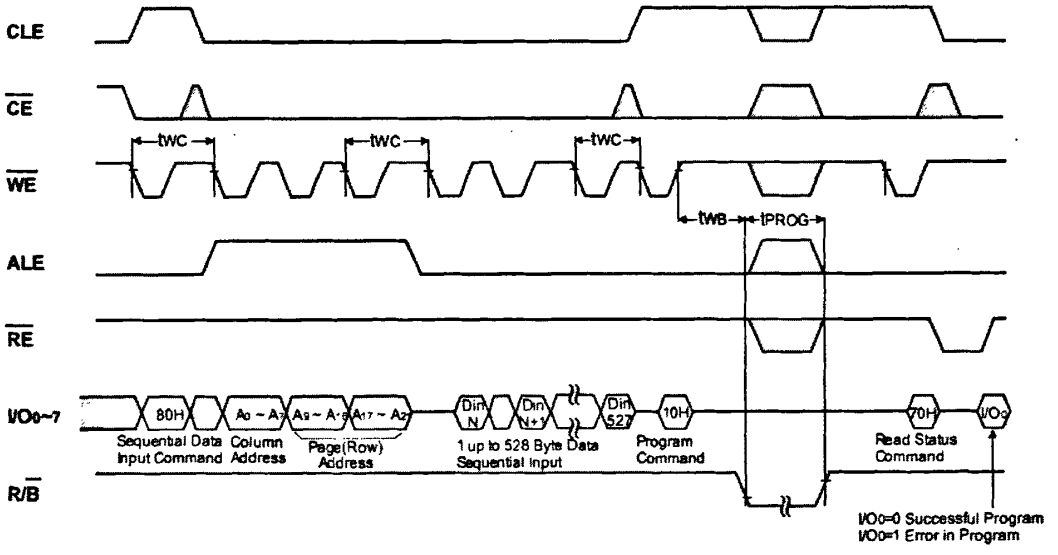

图 3-4 Nand Flash 页编程操作

上图中 CLE 是命令锁存使能位，用于控制命令到达命令寄存器的路径激活。在高电平时经 R/$\overline{B}$ 上升沿采样，将命令锁存到命令寄存器中，如上图中的串行数据输入命令 80H 和编程命令 10H 就是通过这两个信号的配合完成锁存。

$\overline{WE}$，写使能信号，命令、地址、数据均在 $\overline{WE}$ 上升沿时被采样，如上图中地址 A0~A27 的采样，以及 Din 数据的采样。

$\overline{CE}$为片选信号。当处于读操作时，如果$\overline{CE}$变为高电平则 Flash 进入待机模式。如果 Flash 处于编程或擦鞋状态时，将忽略该信号。

ALE，地址锁存使能位。用于控制地址和输入数据锁存到内部对应寄存器中。当 ALE 高电平电平，并且 WE 上升沿时锁存地址；ALE 低电平，并且 WE 上升沿时锁存数据。

$\overline{RE}$，读使能信号。该信号用于串行数据输出控制，高电平时数据在 I/O 总线上传送，直到一个下降沿出现用以确认编程数据真实有效结束，同时将内部列地址计数器加一。

I/O_{0-7}。I/O_{0-7}共有 8 只引脚，用于输入命令、地址和数据，并且在读操作时向外传出数据。当没有片选信号或者输出无效时，I/O_{0-7}处于高阻状态。

$R/\overline{B}$，这个输出信号显示当前 Flash 的状态信息。当处于低电平时，表示 Flash 正在编程、擦除或者进行读取操作。当返回高电平时，意味着上述操作的完成。当没有片选信号或输出无效时，该信号没有高阻状态。所以 Nand Flash 单页编程完成的标志并不是对 10H 命令的锁存，而是等编程时间 tPROG 结束后$R/\overline{B}$上升沿的到来。

利用 CK510 片上仿真模块实现上述 Nand Flash 编程过程的核心汇编程序如下:

```
lrw     r1,0x2000000      //1 设置 Nand Flash 控制器的命令寄存器
lrw     r2,0x2000004      //2 设置 Nand Flash 控制器的地址寄存器
lrw     r4,start_address   //3 设置页编程首地址
lrw     r6,end_address    //4 设置页编程的末地址
lrw     r10,0x80          //5 Nand Flash 的页编程命令 80H
lrw     r11,0x10          //6 Nand Flash 的页编程命令 10H
sth     r10,(r1)          //7 锁存编程命令 80H
lrw     r9,Addr1          //8 目标页地址 1
sth     r9,(r2)           //9
lrw     r9,Addr2          //10 目标页地址 2
sth     r9,(r2)           //11
lrw     r9,Addr3          //12 目标页地址 3
sth     r9,(r2)           //13
lrw     r9,Addr4          //14 目标页地址 4
sth     r9,(r2)           //15
```

```
PRO: lrw    r5,Data          //16 存数据
     stw    r5,(r4)          //17 写入数据，按字大小写入
     addi   r4,0x4           //18 地址加 4
     cmpne  r4,r6            //19 判断是否到达末地址
```

在上述编程过程中，PRO 过程上面的部分为编程的准备阶段，首先完成命令寄存器的配置工作，接着按照 8 位/次的大小锁存地址，锁存 4 次完成后进入 PRO 主循环程序。和 Nor Flash 类似，在主循环程序中进行存数据，写数据和递增地址判断等操作。在单页编程的整个过程中，除了第 16、17 行为编程数据操作外，其余均为编程指令、时序操作。

综上可知，通过边界扫描连编程 Flash 需要元件支持 BS 单元，编程时间受到扫描链长度的制约，即使通过 BYPASS 逻辑实现不必要扫描链的旁通，编程速度也还是受限于具有复杂时序逻辑的 Flash 编程。此外，该类编程还较依赖 PC 软件端的设计支持，对于实现片内 Flash 编程较为困难。总体来说，硬件灵活性较低，速度也不快。利用 JTAG 的片上调试逻辑进行 Flash 编程，较少依赖于目标机的软件，给芯片在软/硬件开发阶段提供了很大的便利。通过调试扫描链，可以在调试模式下控制 CPU 的运行，进而间接访问 FC（Flash Controller）来实现 Flash 的编程。这样就可以借助硬件来解决 Flash 的时序操作，由于 CPU 运行的系统时钟频率远大于 TCK 的频率，所以此类编程不但减轻了目标机软件端的工作，而且一定程度上提高了 Flash 的编程速度。

3.2 快速编程原理

通过上面的编程分析可知，现有通过 JTAG 接口利用片上调试扫描链进行 Flash 编程的方法，其汇编指令都是通过 JTAG 接口的 TDI 串行传入的，不管是何种 Flash，操作时序如何复杂，由于 JTAG 的 TCK 时钟频率的限制，串行输入的速度都不会很快。而且每次编程过程中都需要传输大量的控制指令，导致 JTAG 串口有效数据传送效率非常低下。表 2 显示上述 Nand Flash 批量编程数据时，JTAG 带宽有效利用率为 49.6%，而 Nor Flash 的编程效率则更低，仅为 28.6%，编程控制指令的传输、时序操作指令的传输和地址加载指令的传输耗费了大量的 JTAG 串口传输资源，导致了编程速度缓慢，使得整个系统的调试效率大打折扣。

表 3-2 不同类型 Flash 编程中 JTAG 带宽占用率

	Flash 类型	编程类型	初始化指令	主循环指令	冗余指令	有效指令
带宽占用率%	Nor Flash	双字	50	50	85.7	14.3
		批量	≈0	≈100	71.4	28.6
	Nand Flash	页（2KB）	0.7	99.3	50.4	49.6
		批量	0.7	99.3	50.4	49.6

为了加快编程速度，提高调试效率，目前利用 JTAG 片上调试扫描链进行 Flash 编程的方法都是在传统的扫描链传输方式下进行一定的修改和扩充。如采用了重新设计指令寄存器的方法[17]，将原来一条调试扫描链上的控制位/状态位和数据分开，通过指令译码器产生相关的选择信号，同时制定 TAP 状态机的数据移位周期数，使得每次串行移位周期和相关的寄存器长度一致，“减少了”扫描链的传输长度，提高了 JTAG 口的数据传输效率。另外在 MIPS 芯片支持的 EJTAG 调试扫描编程中，也是从提高传输数据效率的角度进行新的设计[16]。该设计的基本思想是在 CPU 中引入一种文件传输模式，并设计一个指令产生引擎模块，让该模块和 CPU 单独交互，产生编程数据所需要的指令序列，并交给 CPU 核心逻辑执行。由于指令产生引擎模块工作在系统时钟下，所以通过该模块去执行指令的时间可以忽略不计，从而提高了编程速度。除此之外还有一些文献是从提高 TCK 的频率入手的，如文献[20]设计了一个 USB-JTAG 仿真器，通过 SPI（高速同步串行口）接口来替代 GPIO（通用输入/输出）接口用以提高串行输入 TCK 的时钟频率，从而提高编程速度，当然这并没有改变传输过程中的带宽利用率。

由上述改进可知，虽然大家采用了不同的设计方法，性能上也是存在着各自的优劣，但就设计思想本身而言都是考虑了减少数据编程时间和提高数据传输效率，这说明通过 JTAG 片上调试扫描链编程 Flash 速度制约的瓶颈主要在于主机端传送指令所浪费的时间以及较慢的 TCK 时钟频率。而对于片上调试模块而言，串行时钟 TCK 属于外部输入信号，不属于片上调试本身制约编程速度的因素，可在外部采取其他办法提高输入频率。就片上调试模块本身，在完成一次写内存的操作中，多余指令的传输影响了编程数据传送的效率和 JTAG 端口有效数据的利用率，如何削减这些冗余指令，或者设法快速传输这些指令成为在片上调试模块基础上设计高速编程 Flash 的主要方法。

进行快速编程 Flash 的基本原理是：在 Flash 数据下载编程之前，先将 Flash 编程对应的汇编程序下载至片上 SRAM，进一步由硬件自动控制 JTAG 外部接口上的数据接收，并且向 CPU 发出指令控制，令 CPU 在正常工作模式和调试工作模式切换的过程中，执行片上 SRAM 中的汇编程序，完成对数据的编程工作。这样编程使得原先需要从外部不断传入的控制指令和地址变为内部直接加载，在系统时钟频率下，通过设计控制流程使 CPU 全速运行 Flash 编程的汇编程序，同时 JTAG 串行接口接收需要下载编程的目标数据。这样，既可有效减少 JTAG 传输汇编程序中 Flash 编程时序操作的冗余指令，又可以借助高速系统时钟频率提高数据编程下载的速度。进行快速 Flash 编程的大致流程如图 3-5 所示。

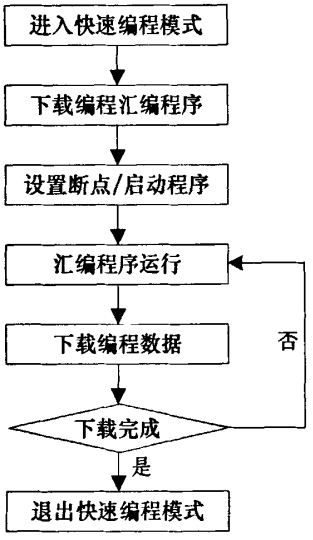

图 3-5 快速编程模式的工作流程

利用片上调试扫描链快速编程的总体工作流程分为两个部分，即快速编程模式初始化和快速编程过程。在快速 Flash 编程之前，主机端需要首先让 CPU 片上调试编程器进入快速编程模式，并通过 JTAG 接口，把 Flash 编程所需的汇编程序下载到片上 SRAM 中去。下载完汇编程序后，CPU 尚不能开始执行，软件端服务程序需要设置汇编控制流程序执行的起始地址和停顿等待的地址。运用片上调试器中存储器断点资源，灵活地设置断点，从而使 CPU 在调试模式和正常工作模式间切换，实现对汇编控制流程序的运行与数据下载编程。具体表现为，主机端服务程序将软件断点设置于 Flash 写入数据的程序上，CPU 会在完成运行部分 Flash 编程控制流程（如初始化流程）之后，遇断点进入调试模式，等待调试接口将数据写入到 CPU 通用寄存器中。等调试接口控制逻辑将准备好的数据写入到 CPU 通用寄存器后，向 CPU 发送继续执行程序的指令，使 Flash 编程控制流程从断点设

置处继续被 CPU 执行，完成后即可实现 CPU 对 Flash 的单次数据编程。在当前数据下载完成之后，CPU 仍然会继续执行下一个循环控制过程，直到执行过程中遇到断点再次停下，进而进行下一次数据的下载编程，如此往复。在整个下载过程中，CPU 等待数据与执行命令是在调试模式和正常工作模式间切换进行的，Flash 编程控制流程序的执行与 JTAG 接口上编程数据的载入也是完全并列执行的，加上 CPU 运行的频率很高，因此数据下载的理论极限速度基本可以接近 JTAG 的传输速度。

3.3 可重构编程思想

由上一节快速编程原理可知，编程 Flash 的汇编控制程序是事先下载到片上 SRAM 中去的。对于不同类型的 Flash，此编程汇编程序必然有所不同，如前面分析的 Nor Flash 和 Nand Flash 在编程方式上、编程命令上都有着很大的区别。仔细分析其整个编程过程，我们不难发现，不论编程过程如何变换，只要从片上 SRAM 内读取的汇编指令为 CPU 指令集指令，便可以完成对该指令的执行操作。如 lrw 指令、sth 指令、addi 指令、cmpne 指令等，只需通过硬件设置 CPU 执行指令的起始地址，CPU 均会自动执行相应指令，只是每次操作执行的地址、数据和选用的寄存器有所不同。因此，对于不同类型 Flash 编程过程中的指令长短、次序变化等问题都迎刃而解。另一方面，在循环编程过程中，每次编入的数据都是不同的，处于动态变化中。为了配合 CPU 在运行程序中提取到预编程的有效数据，可以在 CPU 运行汇编指令至编程数据前设置断点，令 CPU 进入调试模式等待数据进入 CPU 扫描链寄存器中的回写总线寄存器（WBBR），并在 CPU 调试模式下设置相应的 IR 指令，使 CPU 完成对内部通用寄存器的更新。然后，令 CPU 重新跳回正常工作模式，执行原先断点处未完成的指令，便可以实现对 Flash 的一次编程。随着程序的循环往复，CPU 便可以完成对目标 Flash 数据的加载工作。

因此，不论何种类型 Flash 的编程，其汇编控制程序都是先通过原先调试扫描链编程方法下载到片上 SRAM 中去，并在 SRAM 中执行，其程序执行首地址由 PC 指针提供或者通过配置 IR 指令寄存器实现。Flash 编程的初始化过程的时序操作以及后续的数据编程操作都是由 CPU 在系统时钟下完成。整个下载过程中，需要事先下载汇编控制程序，由于该过程只进行一次，并且和编入 Flash 的数据相比，该程序占用空间有限，下载时间也很短暂。这样就可以在提高编程速度的同时，兼顾不同类型 Flash 编程的通用性，体现了可重构思想。

4 高性能片上调试编程器的硬件实现

4.1 总体设计框架

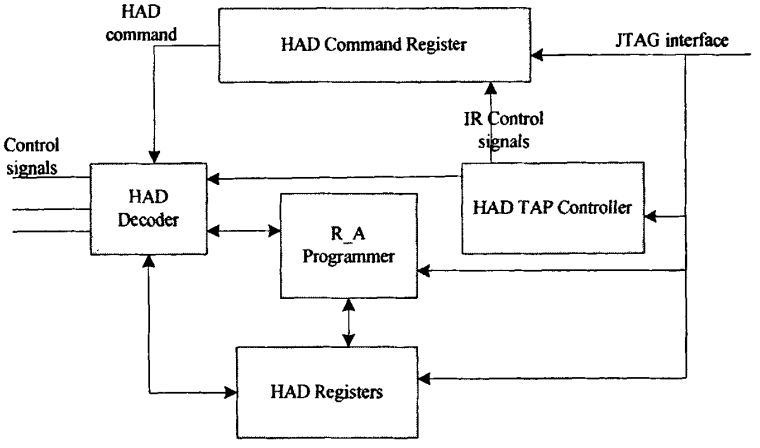

图 4-1 总体框架

本章以 CK510 的片上调试器为对象，在原有片上调试功能的基础上，设计快速可重构编程器硬件单元，配合以原有的断点控制逻辑、CPU 扫描链寄存器等逻辑单元，共同实现快速可重构的下载编程功能。图 4-1 为带有可重构快速编程器的 HAD 框图，显示了编程器在 HAD 中所处的位置以及和主要模块间的互联关系。

HAD Command Register-命令寄存器。如前述的 HACR，寄存器内部保存了 HAD 中数据寄存器对象列表，可以控制存取的对象，同时也可以控制执行单步操作和退出调试模式。

HAD Decoder-译码器。该寄存器从 HACR 中接受 8 位的命令，从处理器中接受当前的状态信息，产生对 HAD 中寄存器读写信号的掩膜信号。

HAD Registers-数据寄存器。HAD 中所有的数据寄存器资源。

TAP Controller-TAP 控制器。完成 JTAG 串行数据通信控制。

R_A Programmer-可重构快速编程器。高速下载编程数据，支持不同类型 Flash 编程。

图 4-2 是 R_A Programmer 模块结构的示意图，显示了该模块与 HAD 间的信息交互关系。

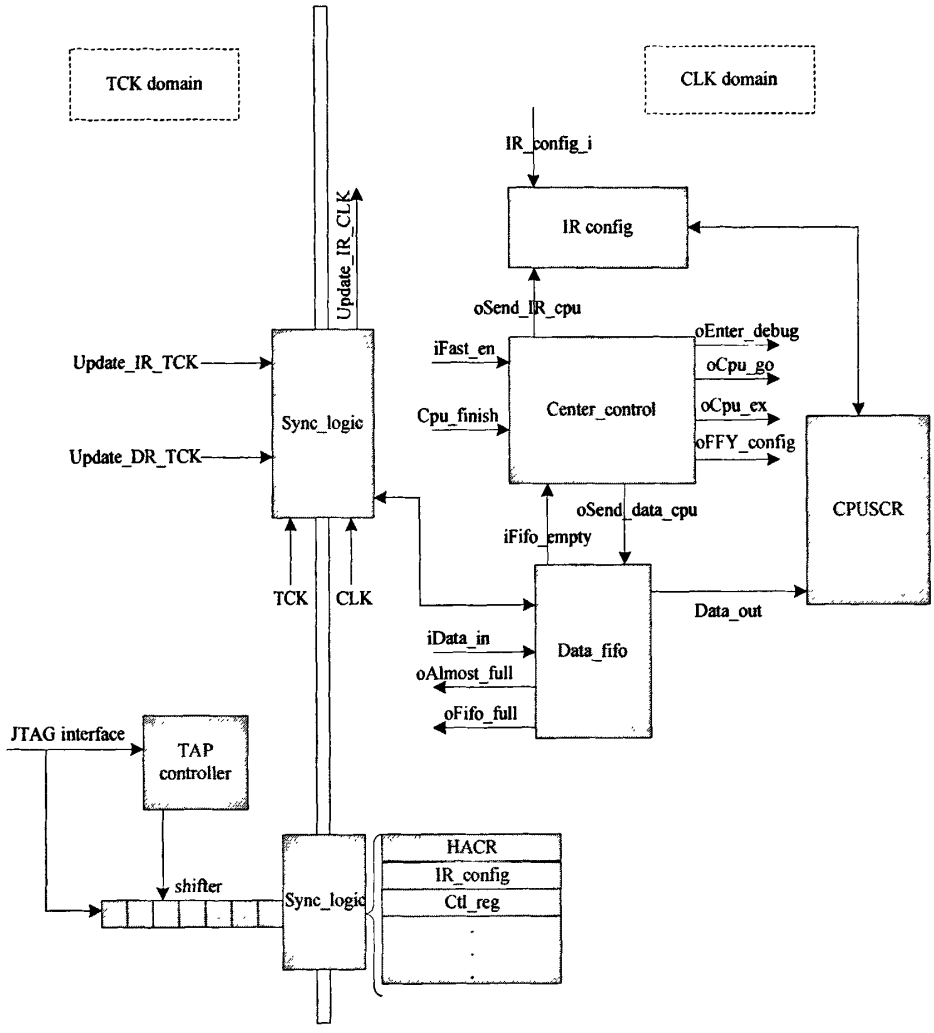

图 4-2 可重构快速编程模块

4.2 寄存器定义

根据硬件逻辑需要，HACR 中的 RS[4: 0]寄存器选择位应添加支持快速编程的数据寄存器，具体定义如下表:

表 4-1 RS[4:0]寄存器定义

RS[4:0]	寄存器名称
00000-00001	保留
00010	ID
00011	追踪计数器

00100	内存断点寄存器 A
00101	内存断点寄存器 B
00110	程序 FIFO 计数器
00111	断点起始地址 A
01000	断点起始地址 B
01001	断点掩膜地址 A
01010	断点掩膜地址 B
01011	处理器扫描链寄存器(CPUSCR)
01100	没有选中任何寄存器(Bypass)
01101	控制寄存器 (HCR)
01110	状态寄存器 (HSR)
01111	存储模式寄存器
10000	记录调试方式寄存器
10001-10110	记录出错信息寄存器
10111	保留
11000	编程器数据寄存器
11001	编程器控制寄存器
11010	编程器状态寄存器
11011	IR 指令配置寄存器
11100-11110	保留
11111	旁路寄存器

在上述寄存器“目录”中，11000-11011 为新增数据寄存器，具体描述如下:

- 11000: Data_fifo-编程器数据寄存器。该寄存器为 33 位数据寄存器，对应 fifo 的数据入口，其中低 32 位是数据有效位，最高位为状态位。
- 11001 “Ctr_reg-编程器控制寄存器。该寄存器为 6 位寄存器

5	4	3	2	1	0
Fast_en	Enter_debug	IR_choice	Cpu_go	Cpu_ex	FFY_config

Fast_en。该位由外部进行配置，用来决定下载器是否开始工作。“0” 不开启快速下载模式；“1” 开启快速下载模式。

Enter_debug。该位由全局控制逻辑发出信号，也可由外部配置。“0”不开启调试模式；“1”进入调试模式工作。

IR_choice。该位由外部进行配置，用来判断选择 IR_config 中的 IR。“0”选择高 16 位 IR，“1”选择低 16 位 IR。

Cpu_go。如果设置该位的话，CPU 会执行保存在 CPUSCR 中 IR 的指令，该位需要和 HAD 中 CSR 寄存器中的 FFY 位配合使用。FFY 位控制 WBBR 寄存器的值作为 CPUSCR 更新后执行的第一条指令的操作数的值。比如更新 R1 的值，其过程为：将数据写入 WBBR 中，设置 FFY 位为“1”，将指令写入 IR 中，设置 Cpu_go 为“1”，CPU 会自动进入正常工作模式，执行 IR 中的指令完毕后，跳回调试模式。Cpu_go 为“0”，不执行 IR 中的指令，为“1”执行 IR 中的指令。

Cpu-ex。该控制位用来控制 CPU 的工作模式状态。“0”保持调试模式，“1”退出调试模式。当且仅当出现 Cpu_go 命令时，处理器将退出调试模式工作状态进入正常工作模式，直到另一个调试请求出现，该位才被再次设置。

FFY_config。该控制位在 HAD 中控制状态寄存器 CSR 中。本编程器需要 CPU 产生相同的动作，故保留并援引该信号来控制 CPUSCR 中 WBBR 值的更新，“0”无操作，“1”WBBR 中的数据作为 CPU 某一操作数的值。

- 11010:State_reg-编程器状态寄存器。

2	1	0
Cpu_finish	Fifo_full	Almost_full

Cpu_finish。从外部来看该位是只读的，“0”CPU 没有将 WBBR 内的值存入 cpu 通用寄存器；“1”CPU 已经将数据存入通用寄存器。

Fifo_full。该位只读，用来显示编程时 fifo 内数据是否已满。

Almost_full。该位只读，用来显示编程时 fifo 内数据深度为“1”时的状态。

- 11011: IR_config-IR 指令配置寄存器。该寄存器为 32 位寄存器，可以配置两条 CPU 调试指令。

4.3 外部 TAP 接口模块

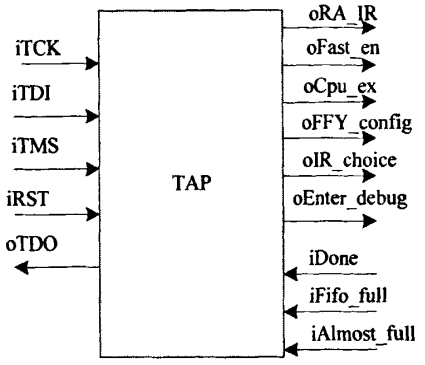

图 4-3 外部 TAP 接口模块

该模块主要用来完成编程器与目标系统外部信息的交互，它包括了 HAD 中 TAP 控制逻辑、JTAG 接口，以及定义的全部数据寄存器。该模块工作在测试时钟域 TCK 下，实现对目标系统外部的指令分类与解析。信号定义如下:

表 4-2 外部 TAP 接口模块信号定义

名称	功能
iTCK	测试时钟输入
iTDI	外部串行数据输入
iTMS	测试模式输入
iRST	复位信号输入
oTDO	测试数据输出
oRA_IR	可重构指令配置输出
oFast_en	快速下载使能输出
oCpu_ex	CPU 退出调试模式
oFFY_config	操作数执行控制输出
oIR_choice	选择 IR 配置寄存器输出
oEnter_debug	进入调试模式输出
iDone	编程数据完成
iFifo_full	数据 FIFO 已满
iAlmost_full	数据 FIFO 将满

4.4 FIFO 功能模块

它是存放编程数据的缓冲区，主要用来平衡 JTAG 接口时钟与 CPU 系统时钟之间的速度，解决异步时钟域的数据同步问题。当 update_DR 信号来临时，该 FIFO 通过对外部控制信号同步处理后，接收外部的数据。示意图如图 4-4 所示。

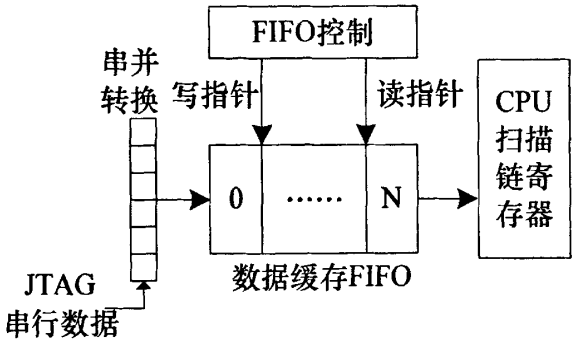

图 4-4 FIFO 缓存设计示意图

根据 FIFO 深度经验公式：$N = \frac{D}{\max(S_{in}, S_{out})}|S_{in} - S_{out}|$，其中 D 为输入数据量，N 为 FIFO 深度，S_{in}，S_{out} 为输入输出速度。公式表明在给定端口的速度下，FIFO 深度 N 和数据量 D 是等价的。由于普通 JTAG 串口时钟 TCK 的频率比系统时钟频率慢的多，所以理论上 $S_{out} >> S_{in}$，故而 FIFO 中不大会出现数据堆砌的情况。考虑不同系统时钟频率不同，兼顾支持快速 TCK 时钟输入的 JTAG 接口类型，此处采用宽度为 32 位，深度为 8 的 FIFO 用来暂存外部数据，通过控制再送往 CPUSCR 进行编程。其具体设计如下：

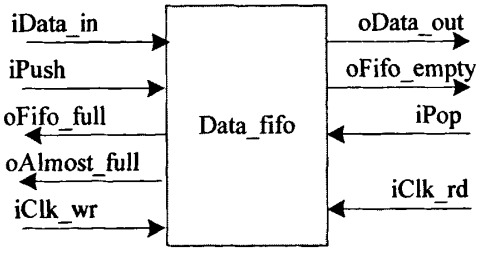

图 4-5 FIFO 模块接口

表 4-3 FIFO 信号定义：

名称	功能
iClk_rd	系统时钟下读时钟信号

iPop	Fifo 数据读出使能信号，高电平有效
oData_out [31:0]	Fifo 读出数据
oFifo_empty	Fifo 空标志，高电平有效
iClk_wr	测试时钟 TCK 下写时钟信号
iPush	Fifo 写入使能信号，高电平有效
iData_in	Fifo 输入数据
oFifo_full	Fifo 满标志，高电平有效
oAlmost_full	Fifo 将近满标志，高电平有效，当 fifo 的空闲深度只剩下“1”时，置位将近满标志位

此 FIFO 属于异步 FIFO，同时工作在读写两个时钟域下。在设计中，除了 32 位的数据存放外，还增加了一位查询位，如图 4-6 所示。

图 4-6 FIFO 内数据结构

设定数据寄存器链为 33 位，其中低 32 位是数据，最高位是 oAlmost_full 查询近满的标志位。之所以这样设计，是因为通过 JTAG 接口向内部 FIFO 串行写入数据的同时，首先移出 oAlmost_full 标志位，如果 oAlmost_full 为“0”，表明 FIFO 未满，可以继续写入下一批次数据；如果 oAlmost_full 为“1”，表明上一次数据写入后，FIFO 中未被占用的存储单元深度只剩下 1，只要再写入一批次数据 FIFO 就满了，所以在当前数据写入后将造成 FIFO 填满，从而可以准确预测出 FIFO 的满状态，避免因 FIFO 突然变满无法接收数据而导致的数据丢失。这样做似乎是给数据传送的时候增添了冗余指令，占用了数据传输的带宽，但事实上采用状态位和数据相结合的链结构，可以边传送数据边查询关键状态，免去了查询状态链所需的时间开销，反而更有利于提高数据的吞吐率。

4.5 全局控制逻辑单元

此单元用来控制硬件模块的整体逻辑，功能包括开启快速下载模式，控制数据缓存区与 CPUSCR 的通信，配置 IR 指令，设置 CPU 工作状态等。其接口模块如图 4-7 所示。

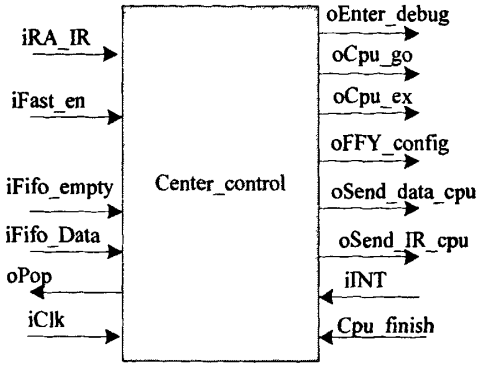

图 4-7 全局控制模块接口

由于全局控制模块是和编程器内其他模块互联一起工作，故该模块的外部接口很多信号只是和其他模块在输入、输出关系上做了互换，功能上在前面都已做过介绍说明，此处不作过多赘述。值得提到的是信号 iINT 和 oPop，前者是 CPU 中断信号的发出，后者为 FIFO 读数据使能信号。该模块主要用来实现整个编程器的全局控制，大部分结构为组合逻辑单元，根据前面讲到的下载编程过程，其整体工作状态机如图 4-8 所示。

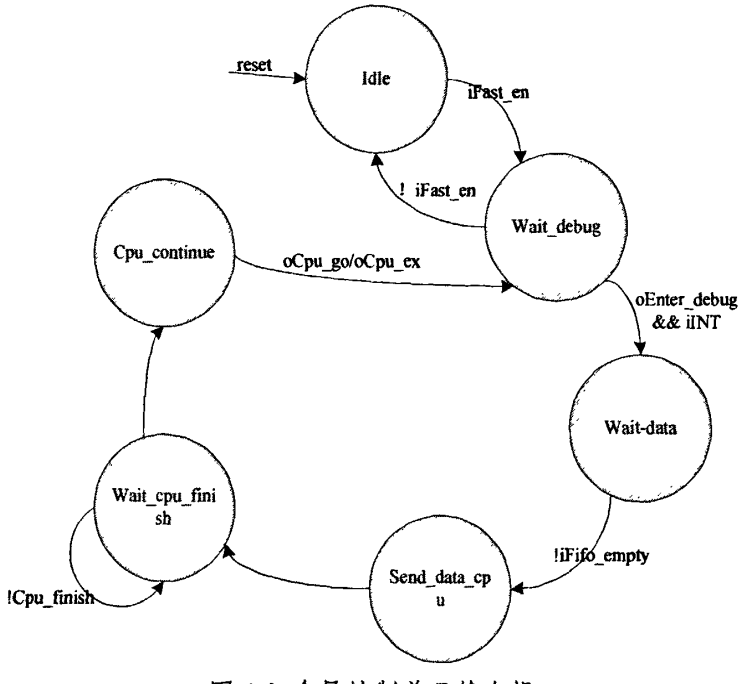

图 4-8 全局控制单元状态机

系统复位后，编程器进入 Idle 空闲状态；当 iFast_en 置高，开启快速下载模式进入 Wait_debug 状态。此时需要使 HAD 进入调试模式，并将控制 Flash 下载编程的汇编程序下载到片内 SRAM 中，同时设置 CPU 开始执行编程初始化程序，完成对 Flash 时序操作

指令的执行，等待预先设置的断点重新跳入调试模式。当 CPU 中断信号产生并进入调试模式后等待 FIFO 中数据是否准备完毕，如果有数据，即 Fifo_empty 非空，则进入 Send_data_cpu 状态，将数据送入 CPUSCR 寄存器链中的回写总线寄存器 WBBR，同时设置 CPUSCR 中 IR 的指令，使得 WBBR 中的数据加载到 CPU 通用寄存器中去，进入 Wait_cpu_finish 状态。完成后跳入 Cpu_Continue 状态，等待设置 oCpu_ex/oCpu_go 控制位，令 CPU 返回正常工作模式，从断点处接着执行指令。根据汇编程序设计，此时 CPU 会根据指令将对应通用寄存器中的数据编程到相应的地址空间上，完成一次编程，继续进入 Wait_debug 状态，等待下一次编程，当所有编程任务结束后退出，进入空闲状态。

4.6 同步逻辑单元

同步逻辑单元是为了解决异步时钟域信号之间的传输导致的亚稳态产生而设计的逻辑单元。所谓亚稳态，是指不同时钟域信号采样时发生的同步逻辑错误[21]。图 4-9 显示了亚稳态发生时的情况。

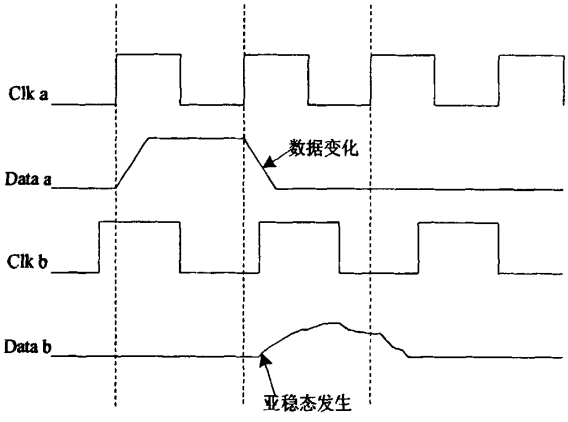

图 4-9 亚稳态的产生

上图中，假设时钟域 Clk a 下的信号 Data a 传入时钟域 Clk b 中。由于 a、b 为两个不同的时钟域，它们之间的频率、相位都有可能不同。当 Clk b 来采样信号 Data a 时，Data a 正好在 Clk b 域的建立时间（setup time）被采样。此时采到的数据 Data b 正好是介入逻辑 1 和逻辑 0 的中间状态，这种保持中间状态的时间就是亚稳态时间，如果亚稳态时间超过一个时钟周期，这种不确定的状态就会传入下一级寄存器，进而导致连锁反应，影响到整个系统的正常功能。因此，异步时钟域的信号传递必须保证在信号的保持期（hold time）进行采样，只有这样采到的信号才是稳定的。

通过同步逻辑解决亚稳态的方法很多，较常用的方法是采用寄存器级联的方式来消除亚稳态。由图 4-9 可知亚稳态发生的时间点为信号采样在建立时间内，是一个不为 1 的概率事件。MTBF（mean time between failures）故障间隔时间[22]，可以预测两次错误发生时之间间隔的时间。通过MTBF来检测亚稳态的公式为[21]: 单寄存器同步 $MTBF(t_r)=\frac{e^{(t_r/\tau)}}{T_0 \bullet f_s \bullet f}$（1）；双寄存器同步 $MTBF(t_r)=\frac{e^{(t_r/\tau)}}{T_0 \bullet f_s \bullet f} \times \frac{e^{(t_r/\tau)}}{T_0 \bullet f_s}$（2）。上述公式中，$t_r$是信号的建立时间，$f_s$是采样时钟频率，$f$ 为异步时钟域频率，T_0和τ为保持时间和建立时间，具体值和采用的元件有关。文献[23]给出了 0.25 微米工艺下双寄存器同步的 MTBF=9.57×10^{10}年。因此，选用双寄存器同步逻辑可以解决亚稳态的问题，如图 4-10。

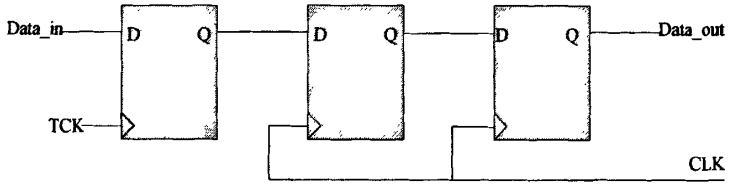

图 4-10 双寄存器级联同步逻辑

在 Center_control 模块中中，同步器用于同步来自 TAP 控制器的 IR 数据以及 iFast_en 等控制信号，将其从 TCK 时钟域同步到 CLK 系统时钟域下。在 FIFO 模块中，同步器用于同步不同时钟域的指针信号，比如在读时钟域，需要将写时钟域的写指针同步到读时钟域，然后和读指针进行比较，从而判断当前的 FIFO 状态。

5 测试验证及性能评估

5.1 测试平台

所谓的测试平台，即 testbench，通常指一段测试仿真代码，用来设计产生特定的输入序列，同时用来观测设计输出的响应。图 5-1 示意了一个测试平台和待测试设计（DUT）之间的相互关系。

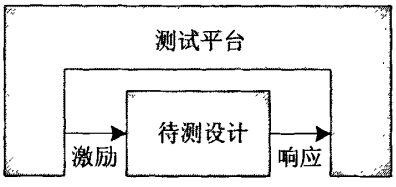

图 5-1 测试平台和待测设计的结构

测试平台一般是一个封闭的系统，它为设计提供输入信号，并监视设计的输出信号，它是整个测试系统的控制核心[24]。基于测试平台再产生具体的测试验证任务，用来确定产生特定的输入模式，以及期望的设计输出。因此，整个测试验证的过程就是通过特定的方法核对设计规范和输出结果一致性的过程，其具体的测试目标视具体的设计规范而定。

本文设计的片上调试编程器，根据硬件实现目标，具体的测试内容应包括：各模块逻辑功能是否按要求正确工作；最终写入 Flash 的数据预编程数据是否一致；编程速度是否符合预期设计等。另外，还需要从硬件角度对设计面积、成本等方面进行可行性评估。

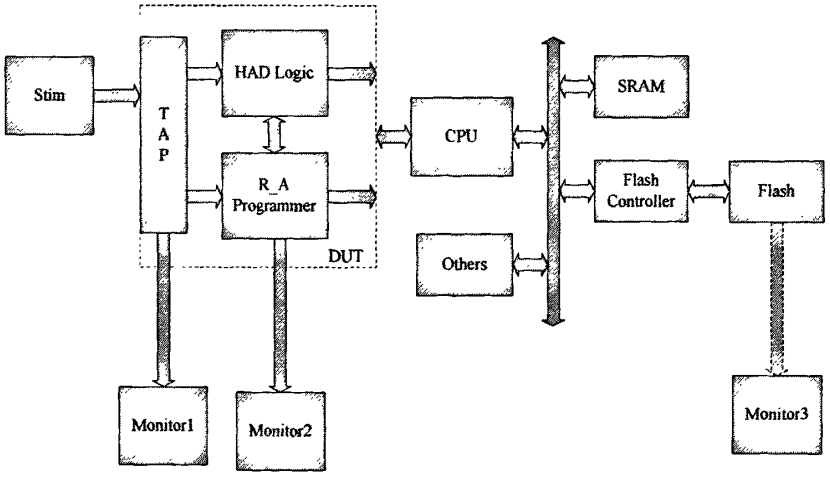

图 5-2 整体测试框架

本文采用 Verilog 硬件描述语言编写测试平台，其测试框架如图 5-2 所示。图中虚线

框内为待测 DUT 模块，包括 TAP 接口、HAD 调试逻辑以及编程器单元。CPU 和 AMBA 总线部分采用行为级模型替代，可以接收执行来自 DUT 的指令、数据，并将其转译为标准总线信号供其他模块收发。Stim 为测试激励源，根据 JTAG 传输标准以及 DUT 的功能定义，发送特定的命令序列和具体的编程数据。Monitor1、Monitor2、Monitor3 为具体的监测点。其中 Monitor1 用来监测 TAP 接口是否能够按照 JTAG 标准正确串行传入测试序列，包括对命令寄存器 HACR 的读写、R_A Programmer 控制寄存器 Ctr_reg 的读写以及状态寄存器 State_reg 等的读写；Monitor2 用来监测 R_A Programmer 模块中信号和数据的变换，其中对 FIFO 的监测验证包括对 FIFO 控制逻辑的验证，如写满状态验证、读空状态验证，以及数据传输的验证。对 Center_control 模块的监测验证包括对全局控制逻辑状态机的测试验证，对断点判断的逻辑跟踪以及各信号间的时序关系等。Monitor3 用来监测 Flash 内编程数据是否和预编程数据一致。由于不能对 Flash 直接进行数据监测，所以在验证数据匹配性的问题上必须先进行编程操作，然后再通过对事先编程的地址上数据的读取来验证数据是否一致。图中 Monitor3 的数据监测需要通过外部发送读取命令序列等操作来完成。此外，除了对各个模块的逻辑功能和数据编程的正确性进行测试验证外，还需要在设置固定时钟频率的基础上对整个仿真编程过程的时间进行测量计算，进而对编程器编程速度进行评估分析。

5.2 测试激励产生

产生激励的过程是一个为待测试验证模块提供输入信号的过程，从激励发生器的角度来看，设计的每一个输入信号都是激励发生器的输出。本设计模块的信号输入接口是基于 JTAG 测试标准的串行输入接口，因此激励发生器产生的信号需要模拟实际中宿主机调试目标系统的时序信号，并严格按照 JTAG 中 TMS 序列转换的顺序发送命令或数据。具体设计的验证 case 应该涵盖对 TAP 接口的测试验证、对 R_A Programmer 内逻辑功能的测试验证以及数据编程结果正确性的验证。下面给出一些关键激励的设计:

- case1 对 HAD 命令寄存器的访问

该激励发送的前提是 CK510 已经进入调试模式，JTAG 控制状态机复位后进入 Run Test/Idle 状态，具体操作为写 HACR 中的 RS[4:0]寄存器，选中编程器控制寄存器。

表 5-1 写 HACR 中编程器控制寄存器序列

步骤	TMS	JTAG 状态	对应操作

1	0	Run-Test/Idle	
2	1	Select-DR-Scan	
3	1	Select-IR-Scan	
4	0	Capture-IR	捕获 CK510 HAD 命令
5	0	Shift-IR	HACR 命令（10'h19）
9 个 JTAG 周期			
7	1	Exit1-IR	命令的最后一位
8	1	Update-IR	更新 HACR 的值
9	0	Run-Test/Idle	

➢ case2 对 R_A Programmer 内数据寄存器的访问

该激励发送的前提是 CK510 进入调试模式，JTAG 控制状态机处于 Run Test/Idle 状态，并且已经通过命令寄存器写入选数据寄存器命令 10'h18，具体的操作为对选中的专用数据寄存器进行写数据测试。

表 5-2 写 R_A Programmer 内数据寄存器

步骤	TMS	JTAG 状态	对应操作
1	0	Run-Test/Idle	
9	1	Select-DR-Scan	
10	0	Capture-DR	
11	0	Shift-DR	写编程器数据寄存器
32 个 JTAG 周期			
13	1	Exit1-DR	最后写入的位
14	1	Update-DR	更新 DR
15	0	Run-Test/Idle	

➢ case3 对 R_A Programmer 内控制寄存器的访问

该激励发送的前提是 CK510 进入调试模式，JTAG 控制状态机处于 Run Test/Idle 状态，并且已经通过命令寄存器选择 R_A Programmer 内控制寄存器，写入命令 10'h19，具体的操作为对选中的控制寄存器写入控制命令。

表 5-3 写 R_A Programmer 内控制寄存器

步骤	TMS	JTAG 状态	对应操作

1	0	Run-Test/Idle	
9	1	Select-DR-Scan	
10	0	Capture-DR	
11	0	Shift-DR	写编程器控制命令
5 个 JTAG 周期			
13	1	Exit1-DR	最后写入的位
14	1	Update-DR	更新 DR
15	0	Run-Test/Idle	

➢ case4 对 R_A Programmer 内状态寄存器的访问

该激励的发送主要是为了能够对 R_A Programmer 内状态寄存器内的数据进行查询和监测，发送的前提是 CK510 进入调试模式，JTAG 控制状态机处于 Run Test/Idle 状态，并且已经选中该寄存器。具体的操作是对 State_reg 进行读操作。

表 5-4 读 R_A Programmer 内状态寄存器

步骤	TMS	JTAG 状态	对应操作
1	0	Run-Test/Idle	
9	1	Select-DR-Scan	
10	0	Capture-DR	捕获 State_reg 中的数据
11	0	Shift-DR	读出 3 位状态数据
2 个 JTAG 周期			
13	1	Exit1-DR	最后读出的位
14	1	Update-DR	
15	0	Run-Test/Idle	

以上是个别测试激励范例。除此之外，激励发生器还包括很多测试序列，比如从外部进入调试模式的测试序列、对 CPUSCR 中 IR 指令配置的测试序列，以及调试模式下通过外部直接读/写寄存器的测试序列等，此处不再一一列举。值得提到的是，在编程 Flash 数据的时候，需要多次用到 case2 的测试序列，如编程 2KB 的数据，需要发送 32 次这样的序列，可见序列中有效数据所占比重的大小，直接影响到 JTAG 串行接口的带宽利用率。本设计使指令从硬件单元发送并执行，大大地减少了 JTAG 串行端口的工作量，这也反映了该硬件设计的实用性。

5.3 测试响应检测

VCS 是编译型 Verilog 模拟器，它支持 OVI 标准的 Verilog HDL 语言、PLI 和 SDF。VCS 具有目前行业中最高的模拟性能，其出色的内存管理能力足以支持千万门级的 ASIC 设计，而其模拟精度也完全满足深亚微米 ASIC Sign-Off 的要求[25]。本节以测试平台设计和测试激励产生为基础进行测试，设定系统时钟工作频率为 100MHz，JTAG 串行时钟输入为 2.5MHz，借助 VCS 仿真工具通过波形等说明设计各模块及关键监测点的数据正确性。

- 写 HACR 中的编程器控制寄存器

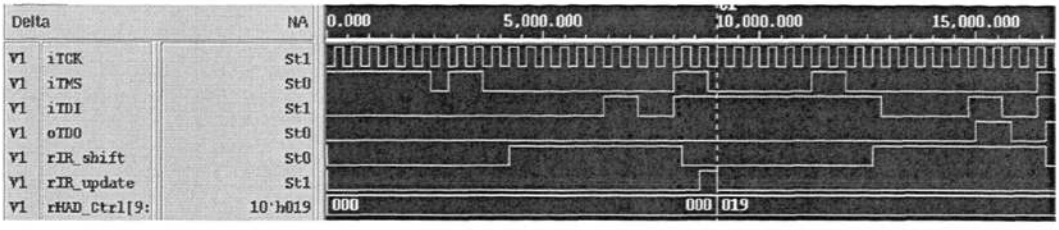

图 5-3 写 HAD 命令寄存器

图中标线处说明经过 Shift IR 和 Update IR 后，数据 10'h19 写入 HACR。

- 写 R_A Programmer 中控制寄存器

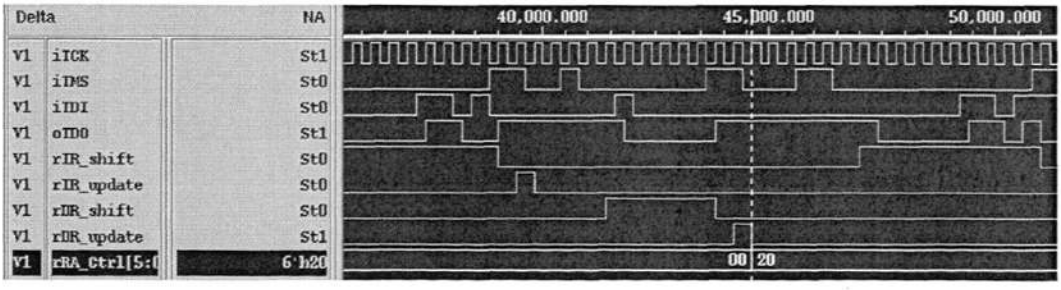

图 5-4 写 R_A Programmer 模块控制寄存器

图中标线处说明经过 Shift DR 和 Update DR 后，控制数据 6'h20 写入控制寄存器。

- 写 R_A Programmer 中数据寄存器

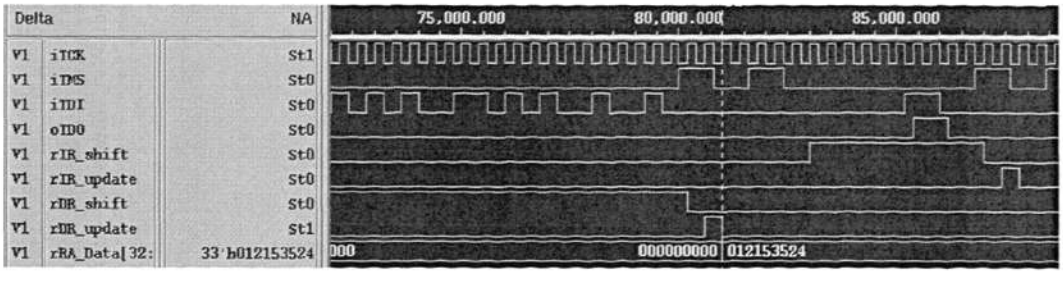

图 5-5 写 R_A Programmer 模块数据寄存器

图中标线处说明经过 Shift DR 和 Update DR 后，数据 33'h012153524 写入数据寄存器，其中最高位为 FIFO 中的判断位。

- 读 R_A Programmer 中状态寄存器

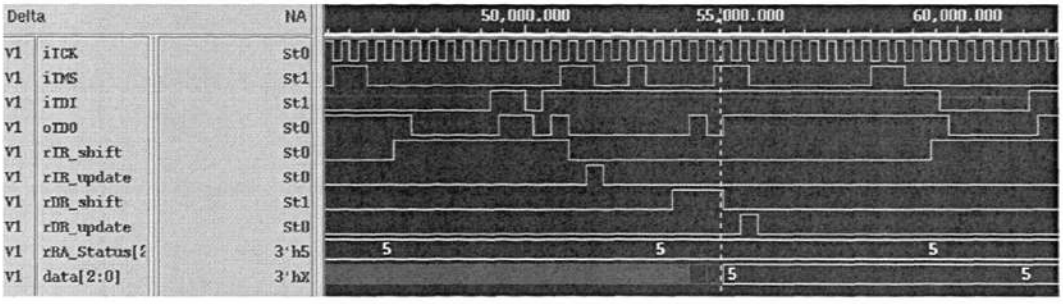

图 5-6 读 R_A Programmer 状态寄存器

图中标线处显示读状态寄存器后从 TDO 串行输出收到的状态数据 3'b101。

- Fifo 读写控制仿真

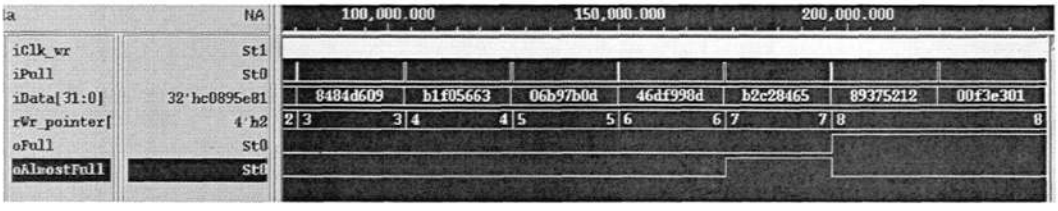

图 5-7 Fifo 写满状态验证

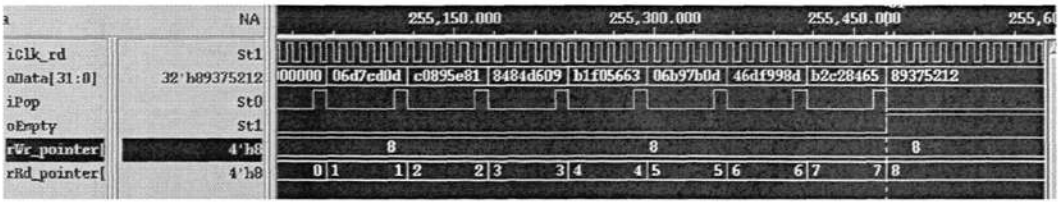

图 5-8 Fifo 读空状态验证

Fifo 读写验证，从下图中可以看出，由于 2.5MHz 的 TCK 时钟频率远远慢于 100MHz 的系统时钟频率，所以读指针 rRd_pointer 一直紧跟着写指针 rWr_pointer，说明 R_A Programmer 能将外部传入的数据快速的传给 CPU 通用寄存器，进而编程至 Flash。

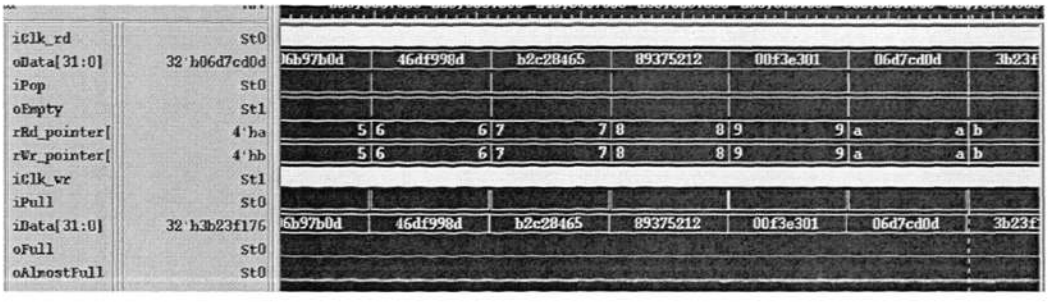

图 5-9 Fifo 读写验证

➢ 对 Center_control 模块的验证

Center_control 模块的主要控制信号在编程工作模式下的仿真波形，当 ifast_en 信号为高时，Center_control 先产生一个 oStart 信号，启动 CPU 执行代理汇编程序，当遇到断点时，CPU 发出一个进入断点的标志信号 iInterrupt 给 Center_control，Center_control 接收到该信号后从 FIFO 中取数据，然后将其放到 WBBR 上，并设置 IR 寄存器和 oExe 控制信号，让 CPU 执行发送过来的 mov r1，r1 指令，完成数据的传递。

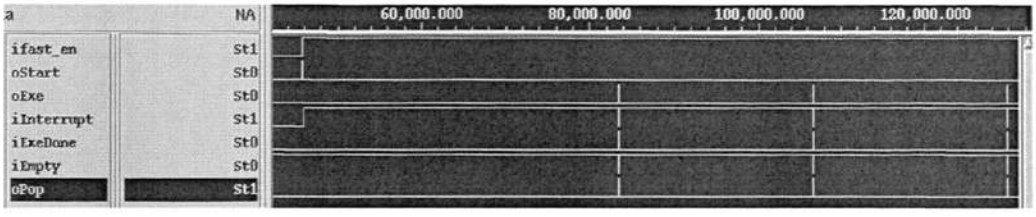

图 5-9 Center_control 模块的验证

➢ 写 Nand Flash 控制器的验证。

对于 Flash 编程的测试验证，包括对 Flash 控制器读写的验证以及对最终数据编程的测试验证，针对不同类型的 Flash，需要选用不同的控制器进行匹配。以 hynix 公司的 HY27us08121m 的 512Mbit Nand Flash[26]为例，配以相应的 Flash 控制器进行编程操作，快速下载编程的测试验证如下。

图 5-10 中，地址 202 对应 NAND Flash 控制器的 CMD 寄存器，地址 203 对应 CONF 寄存器。开始编程之前，代理程序先对 NAND Flash 控制器的命令寄存器（CMD）和配置寄存器（CONF）进行设置，然后从 000 地址开始，向 NAND Flash 的数据缓冲区写数据如图 5-11，当写完最后一比特数据（1ff），向控制寄存器（地址 201）的启动位写“1”，启动编程，NAND Flash 控制器将自动对目标 Flash 进行编程，通过 dio 数据口向 NAND Flash 发送编程命令 80 如图 5-12。

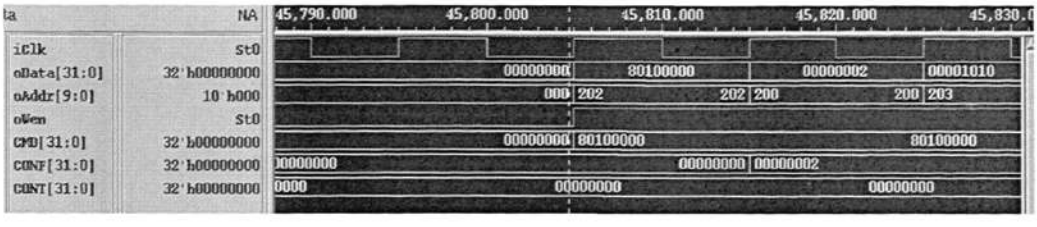

图 5-10 写 Nand Flash 控制器的验证 1

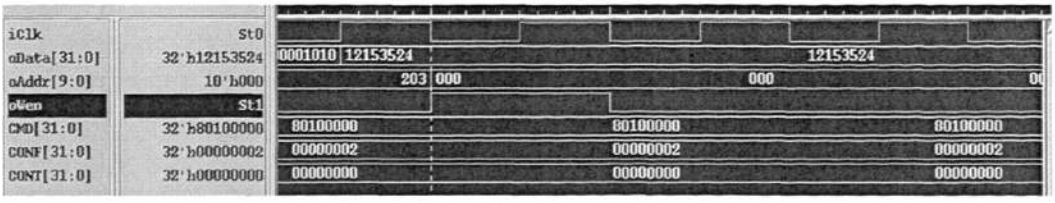

图 5-11 写 Nand Flash 控制器的验证 2

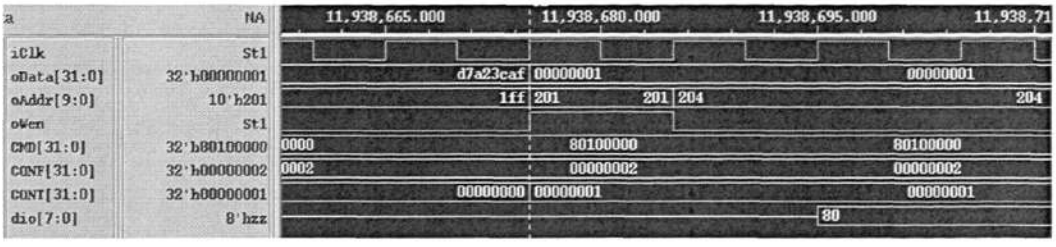

图 5-12 写 Nand Flash 控制器的验证 3

➢ Flash 编程测试验证

按照 Nand Flash 编程时序操作，首先发送编程命令 80H，然后从低到高输送 32 位地址 32'h00001010 给 Nand Flash，接着传送 2KB 的数据到 Nand Flash 的页寄存器 Page Registers，最后发送编程启动命令 10H，这时 Nand Flash 状态信号 R 被拉低，表示 Nand Flash 已经进入编程工作状态，直到 R 重新被拉高时编程完成。编程完成时，Nand Flash 主控的状态寄存器的完成标志位 STATUS[0]将被置高，如图 5-15 所示。

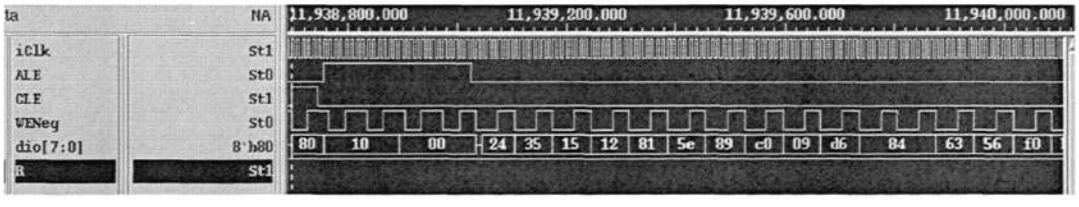

图 5-13 Flash 编程测试验证 1

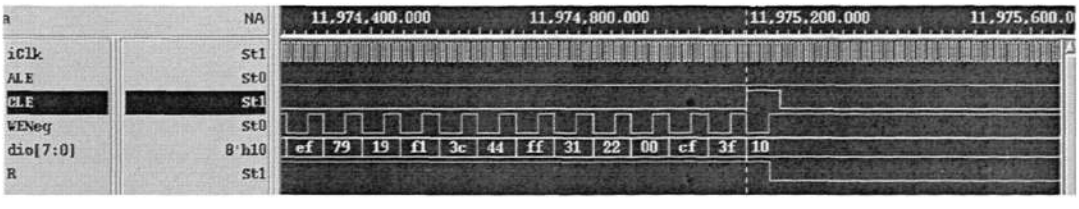

图 5-14 Flash 编程测试验证 2

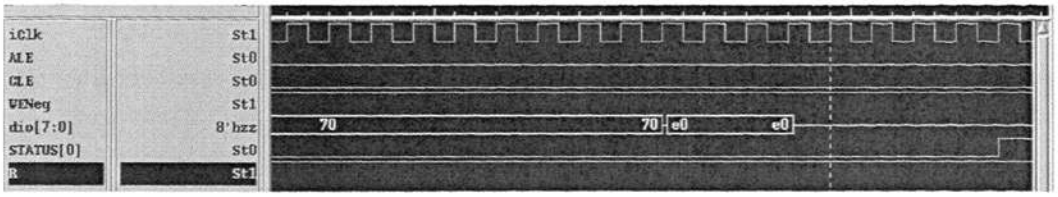

图 5-15 Flash 编程测试验证 3

对于数据一致性的验证，可将预编程的 2KB 随机数据写入文件 write.txt 中，待编程结束后，将编好的数据通过外部读取命令读出存入文件 read.txt 中，对两个文件内的数据采用数组分组，按位异或的方法进行验证。实验结果表明两组数据是完全相同的，证明了编程数据的正确性。

5.4 性能评估

对片上调试编程器的性能评估主要从编程器的编程速度和硬件开销两方面进行评估。在编程速度方面，先看 HAD 未改进前的 Flash 编程。原先方法是先通过 HACR 选中 CPUSCR 128 位寄存器，然后通过外部交叉输送指令和数据完成编程下载的。假设系统已处于调试模式下的 Run Test/Idle 状态。1）完成对指令寄存器的访问更新需要 14 个 TCK；2）完成对数据寄存器 CPUSCR 的访问更新需要 133 个 TCK，总计 147 个 TCK。根据 Nor Flash 的时序操作，CK510 总共需要 6 次这样的操作最多实现编程 Nor Flash 的 32 位数据。标准并行接口（SPP）最高传输速度为 150Kbps[27]，用来模拟产生时钟 TCK，频率可达 75KHz，在大批量数据编程时，忽略掉代理汇编程序执行过程中初始化编程的时间，得到的编程速度为：

$$rate_1 = \frac{4B}{147 \times 6 \times 1/75KHz} \approx 0.34KB/s \quad (1)$$

本设计采用专用数据寄存器传送数据至数据缓存 FIFO，再由编程器写 CPUSCR，其余指令传送均在片内由硬件自动完成，所以外部 JTAG 端传送 32 位数据（此处还应当包括最高位的控制位，总计 33 位数据）只需要 14+38=52 个 TCK，考虑到系统时钟频率远大于 TCK 时钟频率，忽略掉系统内部执行汇编指令的时间和其他在该时钟域下操作的时间，理论编程速度可达：

$$rate_2 = \frac{4B}{52 \times 1/75KHz} \approx 5.8KB/s \quad (2)$$

通过比较可知，编程器的理论编程速度约为普通 HAD 编程速度的 17 倍。在仿真过程中，本文将 TCK 时钟频率设定为 2.5MHz，系统时钟频率为 100MHz，设计测试验证的编程程序大小为 2KB，初始编程时间起点为 0，此时 TAP 控制器开始工作，接收指令和数据。到发送 Nand Flash 编程起始命令 80H 时，已耗时 11.5ms，如图 5-16 所示。

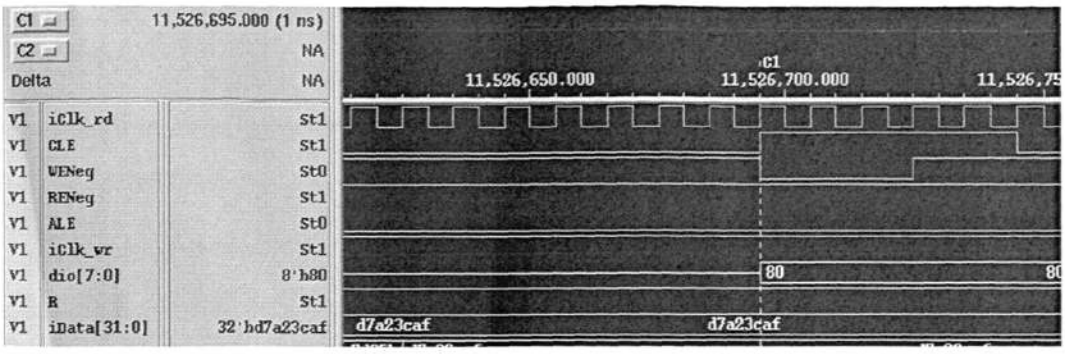

图 5-16 编程命令 80H 发出

当读状态命令 70H 发送后，数据被编程到 Flash 上，又耗时约 0.5ms，如图 5-17 中 Delta 所示。可知总共编程时间约为 12ms，对应的编程速度应该为：

$$rate_3^{'} = \frac{2048}{12} = 170.67KB/s \quad (3)$$

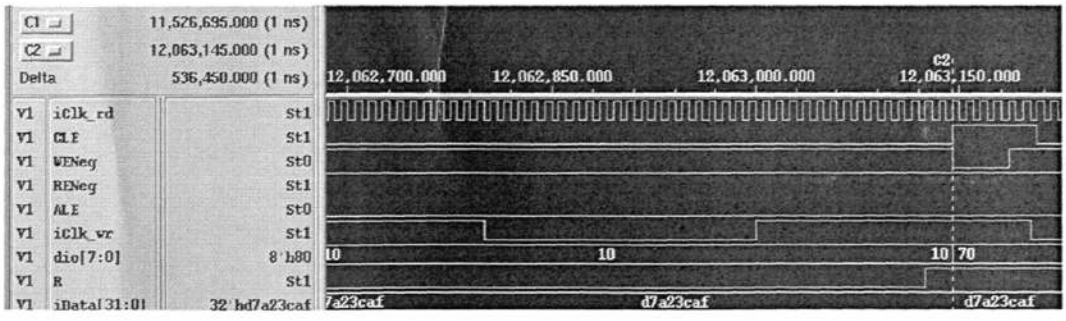

图 5-17 编程命令 70H 发出

式（3）中的速度是 TCK 频率为 2.5MHz 时的编程速度，折算到时钟频率 75KHz 的速度为：

$$rate_4^{'} = 170.67 \times \frac{75}{2500} = 5.12KB/s \quad (4)$$

由于计算的理论速度没有考虑到 Flash 编程过程耗用的时间，所以实际上通过 JTAG 串行编程耗用的时间为 11.5ms。对应的编程速度为：

$$rate_3^{'} = \frac{2KB}{0.0115} = 178KB/s \quad (5)$$

折算到 TCK 频率为 75KHz 的速度为：

$$rate_4^{'} = 178 \times \frac{75}{2500} = 5.34KB/s \quad (6)$$

对比式（2）和式（6）的速度不难发现，仿真实测速度和理论速度相差 0.46KB/s，误差范围不大，证明了硬件支持快速编程的可行性。

表 5-5 列举了本设计和 HAD 原始编程，以及另外两种片上调试编程设计的性能比较，其中文献[13]为利用边界扫描链 BS 单元编程 Flash 方法，文献[17]为利用片上调试扫描链改进型 Flash 编程，其编程速度均折算到普通并行接口提供 TCK 峰值频率下的理想编程速度。可以看出，本设计在编程速度和编程通用性两方面都存在着优势。

表 5-5 性能比较

	编程速度	设计通用性
本文	5.12KB/s	支持不同类型 Flash 编程
原 HAD	0.34KB/s	支持不同类型 Flash 编程
文献[2]	1.56KB/s	支持不同类型 Flash 编程
文献[4]	5.6KB/s	支持特定类型 Flash 编程

在硬件资源耗用方面，主要通过 Synopsys 公司的逻辑综合优化工具 Design Compiler（DC）对设计的硬件进行综合分析。DC 是把硬件语言描述的电路综合为跟工艺相关的门级电路，并且根据用户的设计要求在时序、面积功耗上取得最佳的效果。根据具体设计，综合采用 smic.18 工艺，有两个时钟源 TCK 和系统时钟 CLK，设定系统时钟工作在 100MHz 频率下，对应设置 CLK 的周期为 10ns，而 TCK 频率较慢，设置 TCK 的周期为 400ns。

```
create_clock  –period 10 -name myclk [get_ports clk]
create_clock  –period 400 -name mytck [get_ports tck]

set_dont_touch_network [get_clocks myclk]
set_dont_touch_network [get_clocks mytck]
set_dont_touch_network [get_ports rst]
```

由于该设计涉及到两个时钟域，在设置时序约束的时候，还需要将跨时钟域的组合路径设置成虚假路径，跨时钟域的虚假路径约束设置如下：

```
set_false_path -from [get_clock myclk] -to [get_clock mytck]
set_false_path -from [get_clock mytck] -to [get_clock myclk]
```

在面积综合方面约束其为最小面积，设置为：

Set_max_area 0

其他约束条件可以参考 DC 使用手册，这里不再列出。综合后的时序结果如下所示，包括 TCK 时钟域和 CLK 时钟域两个时序报告，结果表明均符合设计要求。综合报告如下，面积单位是平方微米。

```
****************************************
Report : qor
Design : RA_programmer
Version: W-2004.12-SP2
Date   : Mon Dec 25 23:22:13 2009
****************************************
  Timing Path Group 'myclk'
  -----------------------------------
  Levels of Logic:                12.00
  Critical Path Length:            9.56
  Critical Path Slack:             0.18
  Total Negative Slack:            0.00
  No. of Violating Paths:          0.00
  -----------------------------------
  Timing Path Group 'mytck'
  -----------------------------------
  Levels of Logic:                 4.00
  Critical Path Length:            7.89
  Critical Path Slack:           391.94
  Total Negative Slack:            0.00
  No. of Violating Paths:          0.00
  -----------------------------------
  Area
  -----------------------------------
  Combinational Area:    15683.807617
  Noncombinational Area: 27732.070312
  Net Area:             193247.687500
  -----------------------------------
  Cell Area:             43416.171875
  Design Area:          236663.859375
```

由上可知，该设计满足时序设定要求，在 TCK 时域中，由关键路径信息可以看出执行的预留时间还很充裕，因此可以通过提高 TCK 的时钟频率来进一步地加快编程速度；从面积报告中看出，该硬件模块综合后的面积不大，在大幅提高片上调试编程效率的情况下，仅耗用了很少的硬件资源，性能材耗比很高。

6 结束语

电子科学技术的发展使得嵌入式 SoC 开发技术已逐渐成为电子信息产业中最具增长力的一个分支。随着各种多功能高集成度、高复杂度的 SoC 的出现，系统开发人员面临的是更为复杂、繁琐的调试工作。而调试设计一般占到了总体系统开发时间的 30%—50%，所以调试工具功能的强大与否直接影响到产品上市时间的快慢。而在 SoC 系统开发调试过程中，对 Flash 编程速度的快慢直接影响着 SoC 系统调试效率的高低。因此，设法快速地实现 Flash 编程，节约调试工作本身外不必要的时间，是提高调试效率的一项重要工作。围绕着这个基本问题，本论文主要做了以下工作：

1. 认真分析概括了嵌入式调试技术的发展、分类以及各自的特点，提出调试设计是嵌入式开发中的一项重要工作，总结出片上调试技术是目前嵌入式调试技术的主流。并在此基础上总结了基于片上调试技术编程的各类方法，提出一种新的基于片上调试技术的高性能编程方法。

2. 详细的介绍分析了 JTAG 技术原理，并以 CK510 芯片为例，分析了基于 JTAG 标准的片上调试技术及硬件结构。

3. 在总结片上调试编程 Flash 技术的基础上，通过对 Flash 编程过程的详细分析，给出高性能片上调试编程器的设计原理，并加以说明论证。

4. 对片上调试编程器进行硬件设计，给出整体设计结构、各子模块的设计结构、寄存器定义和各模块间通信关系以及具体的工作流程。

5. 对设计的硬件进行系统性的测试验证，通过仿真测试证明了硬件设计的合理性和实用性。

快速编程 Flash，提高调试效率是当今 SoC 系统开发中一项重要工作。本文提出了一种基于 JTAG 快速可重构编程 Flash 的设计方法，不但适用于不同类型 Flash 的在线编程，而且很好地提高了编程速度，该设计具有一定的参考价值。

参考文献

[1] 赵岩 SoC 中嵌入式微处理器调试技术的研究和实现 湖南省长沙市 中南大学 2005.

[2] 桑楠 嵌入式系统原理及应用开发技术. 北京航空航天大学出版社 2002-04.

[3] 孙玉芳等译 嵌入式计算系统设计原理. 机械工业出版社, 2002-02, 142-143.

[4] 戴卫彬 基于 JTAG 的在系统编程和硬件调试研究与应用 云南昆明 昆明理工大学 2005-12-05.

[5] 朱梅 基于 JTAG 标准的通用交叉调试代理的设计和实现 四川成都 电子科技大学 2007-04-23.

[6] 毛德操 胡希明 嵌入式系统——采用公开源代码和 StrongARM/Xscale 处理器 浙江大学出版社 2003.

[7] 罗克露 陈云川 嵌入式软件调试技术 电子工业出版社 2009-01.

[8] Arnold Berger, Michael Barr, Introduction to On-Chip Debug. http://www.embedded.com/story/OEG20030205S0032.

[9] IEEE. IEEE1149.1-2001 IEEEE Standard Test Access Port and Boundary-scan Architecture[S]. 2001-09:3.

[10]Test Technology Standards Committee of the IEEE Computer Society. IEEE Standard Test Access Port and Boundary-Scan Architecture. 2001.

[11]郑先刚 张学斌 基于 JTAG 技术的 Flash 加载 现代电子技术 2004 11 5-7.

[12]赵海舰 甘萌 嵌入式系统中的 Flash 编程技术研究 计算机工程与设计 2005 26（11）3006-3009.

[13]张琳 周拥军 武飞 Flash 在线数据加载的 JTAG 通用方法研究 电光与控制 2009 16（3）95-97.

[14]赖欣 胡泽 赖晓斌 汪春浦 TMS320LF240XA 系列 DSP 片内 FLASH 编程技术 国外电子测量技术 2005 24（9）20-23.

[15]许琼 基于 JTAG 的 ARM7TDMI 调试系统 计算机工程 2008 34（15）252-254

[16]游海量 葛海通 严晓浪 一种基于 JTAG 协议的嵌入式调试接口设计方法 江南大学学报（自然科学版） 2007 6（5） 523-527.

[17]董相晖 张志敏 一种基于 EJTAG 快速在线烧写 Flash 的设计 微电子学与计算机 2007

24（12） 106-108.

[18]SPANSION. S29GL-N, MirrorBit® Flash Family. 2008-05-30.

[19]SAMSUNG Electronics. K9F3208W0A-TCS0, 4M×8Bit NAND Flash Memory. 1999-09-15.

[20]Xuhui Chen, Dengyi Zhang, Hongyun Yang. Design and Implementation of a Single-chip ARM-based USB Interface JTAG Emulator. The Fifth IEEE International Symposium on Embedded Computing[C]. SEC 2008, 2008-10: 272-275.

[21]William J Dally and John W. Poulton, Digital Systems Engineering, Cambridge University Press, 1998:462-513.

[22]Mean time between failures. http://en.wikipedia.org/wiki/MTBF.

[23]杜旭 左剑 夏晓菲 何建华 ASIC系统中跨时钟域配置模块的设计与实现 微电子学与计算机 2004 21（6） 173-177.

[24]Janick Bergeron. Writing Testbenches-Functional Verification of HDL Models. Springer. 2000-1-1.

[25]VCS. http://baike.baidu.com/view/1808584.htm?fr=ala0_1.

[26]Hynix Electronics. HY27SS(08/16)121M Series 512Mbit NAND Flash Memory. 2004-10

[27]IEEE. IEEE1284 Parallel Ports. Lava Computer MFG Inc. 2002-05-29.

[28]D.A Bonnett, “Design for In-System Programming” ,in Proc. Int. Test Conference(ITC’99), Atlantic City, NJ, USA, Sept 28-30, 1999, pp. 252-259.

[29]JTAG Interface: Simple Introduction. http://www.amontec.com/

[30]MIPS Technologies. EJTAG Specification [OL]. Revision 2.6 http://www.mips.com.

[31]C. Ingeol and L. Chaedeok, “ES-debugger: the flexible embedded system debugger based on JTAG technology”, Advanced Communication Technology, ICACT 2005,2005.

[32]A.Jutman,“At-SpeedOn-ChipDiagnosisofBoard-LevelInterconnectFaults”,in Formal Proc .of 9th IEEE EuropeanTestSymposium(ETS’04),France,2004,pp.2-7.

[33]成杏梅 刘鹏 钟耿 等 嵌入式MPSoC的调试功能实现[J] 计算机辅助设计与图形学学报 vol.20, 2008 438-445.

[34]Sergei Devadze, Artur Jutman, Igor Aleksejev, Raimund Ubar. Turning JTAG Inside Out for Fast Extended Test Access. Test Workshop, 2009. LATW '09. 10th Latin American. 2009-3, 1-6.

[35]严晓浪 孟建熠 葛海通 嵌入式处理器高速在线下载直通通道的设计方法[P] 中国发明专利 专利号：CN200710156914.7

[36]张伟 李兆麟 张闯 一种基于JTAG的嵌入式微处理器片上可调试系统[J] 计算机工程与应用 2004 1-5.

[37]K. P. Parker. The Boundary-Scan Handbook. Kluwer Academic Publishers, Boston, MA, USA, 2003, 373p.

[38]B. Nadeau-Dostie, et.al, "An Embedded Technique for At-Speed Interconnect Testing," in Proc. Int. Test Conf.(ITC'99), Atlantic City, NJ, USA, Sept28-30,1999, pp. 431-438.

[39]蒲石 异步多时钟域系统的同步设计研究[M] 西安：西安电子科技大学，2007.

[40]徐世伟 刘严严 刘红侠 异步时钟亚稳态及FIFO标志位的产生口[J] 电子技术应用 2006(11)：99-102.

[41]夏宇闻 Verilog 数字系统设计教程 北京航空航天大学出版社 2003-7

作者简介

陈超，男，1983 年出生。2002-2006 就读于浙江大学电气工程学院电气工程及其自动化专业；2006-2007 加入浙江大学第八届研究生支教团，赴四川省凉山州昭觉县支教一年；2007-今就读于浙江大学电气工程学院电路与系统专业。攻读硕士学位期间，主要从事数字电路设计及信息安全方面的研究。

攻读硕士学位期间发表的论文

《一种基于 JTAG 的可重构 Flash 快速编程方法》，已被《计算机工程》录用，拟刊登于 2010 年第 18 期。