硬件实验课设报告

兼容 ARM9 的软核 CPU（基于 FPGA）

姓名：马逸君

学号：17300180070

2019 年 12 月

本书内容综述

第一章 数字电路设计模型

1.1 数字电路的最简抽象模型

任何复杂的数字电路都可以抽象成带有输入输出的黑箱模块或其组合。

1.2 组合逻辑

组合逻辑电路的输入变化都会立即导致输出重新计算，但输出的变化(如有)会有延迟。

1.3 时序逻辑

时序逻辑最基本的器件是寄存器。
D 寄存器在时钟上升沿赋值为输入信号(有延迟)，在其它时刻保持现有状态。

1.4 同步电路

异步电路中寄存器的时钟输入来自不同的时钟源。
异步电路中采用同一时钟源的电路可以作为同一种类型来分析，而它们之间的连接必须单独考虑。

1.5 同步电路的时序路径

同步电路的时序路径分为四类：
外部输入端口输入到寄存器；寄存器输出端口输入到其它寄存器(包括自身)输入端口；寄存器输出到外部输出端口；输入端口通过组合逻辑直接到输出端口。
寄存器到寄存器之间组合逻辑消耗的时间不能超过一个时钟周期。
通常是最长的组合逻辑串决定时钟周期/频率。

1.6 RTL 描述

一个数字电路可以采用 RTL（寄存器传输级）设计描述：
第一步，确定需要用到多少寄存器；第二步，对每一个寄存器的输入端口进行描述；
第三步，从时钟频率、组合逻辑等方面确保寄存器之间正常连接。
一个寄存器最重要的属性就是其输入端口连接的组合逻辑串。
该输入描述可能是该电路中所有外部输入端口和所有寄存器输出端口的函数。
只要给出了所有寄存器的输入描述，就确定了电路所有的组合逻辑。

1.7 综合生成电路

设计数字电路时不必在硬件程序中给出确定的实现方案，我们可以用行为级或更高抽象级别的描述，而综合软件会将其综合生成我们需要的电路。

第二章 Verilog RTL 编程

2.1 Verilog 语言

assign 语句用于描述组合逻辑串。
always @ (...)的含义是，当且仅当"..."发生时重新计算语句内的表达式。
<=符号专用于寄存器描述，它的含义是符号右边的值在时钟上升沿结束后存入左边，同一个 always 语句下的所有"<="语句是并行的。

2.2 Verilog 设计的基本单元

always @ (...)语句一般用于描述寄存器的行为，可以在该语句中描述寄存器本身和它专属的一段组合逻辑。
assign 语句用于描述简单的组合逻辑，但要求在一个语句内完成描述。
always_comb 用于描述复杂的组合逻辑，系统将自动在接下来的语句中寻找敏感信号（等效于 always @ (*) ）。
function ... endfunction 语句用于描述多次调用的组合逻辑，该语句也是可综合的。

2.3 RTL 设计原则

任何编程语言都实现对信息的处理。在进行 RTL 描述时信息以寄存器的形式存在，就像 C 语言中的变量。
设计完电路后需要检验寄存器之间的相互关系是否符合预期，这通过仿真来实现。

2.4 RTL 设计要点

作者建议，描述设计时，寄存器需要与实体明确地一一对应，组合逻辑则不必完全描述出具体构造，它往往附属于寄存器。
综合软件会标出电路中最长的路径（关键路径），如果关键路径超标，则需要调整描述风格，让综合器使用较少的逻辑串来描述关键路径。

2.5 RTL 设计实例：UART 串口通信

（代码见附录）

第三章 仿真

3.1 仿真的意义

仿真就是对设计的试用过程：给予 RTL 设计一定的激励，观察响应是否正确。
RTL 设计的完整流程要求完成设计后立即检验正确性，这样才能验证当前设计能否符合对应的需求，如果可以，则在下次碰到相同需求的时候可以照搬这种设计套路，从而提高熟练度；
而不是仅仅写完 RTL 代码，然后采取哪里有问题修复哪里的策略。

3.2 testbench 文件

testbench 文件是这样一类 Verilog 程序，它给待测试的 Verilog 程序送入指定的信号，接收其输出信号(并进行一定的测试)。
一个 testbench 文件是封闭的，只为一个特定的 RTL 文件服务，一般无外部输入输出端口。

3.3 Modelsim 仿真

仿真软件默认情况下会认为信号传递没有延时。我们可以手动为每一个寄存器赋值语句给予延时，但不超过一个时钟周期。可以用`define 语句定义常量。

3.4 仿真实例：UART 串口仿真

task 语句和 function 类似，但不必返回数据。可以在仿真文件中用 task 语句表示频繁调用的代码。
可以调用系统函数，如$end 表示结束仿真，$display 表示输出调试信息。
（代码见附录）

第四章 FPGA 开发板原型验证

4.1 FPGA 内部结构

FPGA 与 CPU 的异同：都采用二进制配置文件来控制执行效果；
CPU 芯片只有一个“大脑”，包干所有“活计”，但也只能一件接一件地做事情；
FPGA 则依靠一个虚拟的“团队”，并发执行效率可能高于 CPU。
CPU 芯片的二进制配置文件由 C 语言编译而成，FPGA 则由 Verilog RTL 语言编译而成。
FPGA 称为“可编程”芯片，编译器将 Verilog 编译成二进制编程文件下载到 FPGA 中，使 FPGA 被“编程”为 Verilog RTL 功能的芯片。
Xilinx Spartan-3 系列的芯片内部构造：
四周是 IOB(Input/Output Block)，其引脚可以由用户配置成输入、输出或同时可输入输出的，电压也可由用户配置。
特殊功能部件有 DCM（生成时钟信号）、block RAM（专门用于 RAM 的实现）、Multiplier（专门用于乘法运算）。
最基本的部件是 CLB(Configurable Logic Block)。一个 CLB 内部是 4 个 Slice，分为两个 SliceL 和两个 SliceM。
每个 Slice 含有两个寄存器，用于实现 RTL 编程中的寄存器单元，其他部分（LUT、进位、MUX、运算逻辑等）实现组合逻辑。
实现组合逻辑的核心是 LUT4，它有 4 个输入端口、1 个输出端口，每种输入对应的输出可任意设定，是根据用户配置来设定的。用于实现各种各样复杂的组合逻辑。一个 LUT4 可以描述任何四输入一输出的组合逻辑。
SliceL 和 SliceM 的区别是内部存放这 16 个输出值的载体不同，SliceL 是一次写入、使用时不可更改，配置后只能作为组合逻辑；SliceM 使用寄存器，所以它可以充当 RAM，也可以用于移位寄存器。

4.2 FPGA 开发板

FPGA 开发板是以 FPGA 为核心、围绕其放置了各种外设的电路板。用户的设计可以通过和外设连接的接口，驱动外设工作或获取外设数据。
FPGA 开发板上的接口一般分为以下几类：
(1)简单电平(switch)/脉冲输入(button，按下时产生一个脉冲信号)。
(2)简单信号显示。包括 LED、七段数码管、LED 显示屏。
(3)时钟信号输入。可以直接利用焊接在开发板上的晶振，也可以接入外部时钟。
(4)存储单元。包括板上焊接的 Flash、SRAM、SDRAM 和有些开发板配备的 SD 卡槽等。
(5)专用输入输出接口。如 UART 串口、VGA 接口、USB 接口。
(6)扩展接口。用于在 FPGA 开发板上增加专用于某种应用的子板。

4.3 FPGA 设计开发流程

第一步，评估 FPGA 的片外资源和片内资源。
片外资源指 FPGA 引脚连接的资源，它是通过明确 FPGA 各个引脚具有何种功能确定的；
片内资源指 FPGA 内部结构中的资源，包括 DCM(Digital Clock Manager)、Multiplier、Block RAM，有的可以通过 Verilog RTL 描述自动调用，有的则必须通过配置 IP 的形式来进行。
第二步，围绕这些资源进行 Verilog RTL 编程。
第三步，使用仿真工具对设计进行调试。
片外资源是在 testbench 文件中“虚拟”出来的，片内资源如通过 Verilog RTL 描述的方式调用，则无需改动，如通过配置 IP 的方式，则需要调用 FPGA 生产商提供的仿真模型（当然也能手动编写仿真模型）。
第四步，FPGA 下载执行。
这一步又分为四个子步骤：
综合优化。将 Verilog RTL 描述中抽象的组合逻辑描述具体化成门级（基本逻辑单元组成的逻辑连接关系图）。
翻译(translate)。将由基本逻辑单元联系而成的网表映射成 FPGA 底层元件的逻辑结构，如把以与门、或门、非门连接的逻辑变成以 LUT4 为主体的描述。
翻译过程可以由用户的约束指定方向，比如要求面积小则会把多个逻辑门融合到尽量少的 LUT4 上，要求时间短则会优化关键路径使其经过尽量少的 LUT4。
映射(map)。把由 FPGA 底层元件连接的网表对应到该 FPGA 内部的 CLB 和 IOB 上。但仍然不是一一对应的映射，只是确定某些 CLB 分配给某些种类逻辑。
布线(place&route)。根据上一步的布局对具体 CLB 和 IOB 进行可编程连线，形成真正能够下载到 FPGA 的二进制 bit 文件。布局布线工具会进行优化，避免一个 Slice 和其它太多 Slice 连接在一起形成拥塞。

4.4 FPGA 设计内部单元

在进行 RTL 编程时，如果了解 FPGA 的构造，针对其结构进行编程，则 FPGA 的综合器可以生成非常高效的网表，在 FPGA 中执行的效率也会很高。
FPGA 内部有一些基本单元(primitive)和宏单元(macro)，在 FPGA 供应商提供的帮助文档中会列出。如果想优化 FPGA 设计，则可以在设计中直接例化，也可以通过行为相符的 HDL 描述来调用它。
以 Xilinx Spartan-3E 为例：
(1)算术功能单元。可例化调用，综合器也会直接把 HDL 中的整数乘法用这些单元级联实现。
(2)时钟单元。一般自动调用即可，也可手动干预。
(3)输入输出端口单元。一般情况下 FPGA 顶层输入输出引脚会自动根据情况应用这些单元。
(4)RAM/ROM 存储单元。欲使综合工具将相关语句例化为 Block RAM，描述风格必须和 Block RAM 的行为一致，具体参看综合器帮助文档。而在 HDL 中描述的大型 ROM 可以采用 IP 配置向导生成以 Block RAM 为基础的 ROM。
(5)寄存器和锁存器单元。几乎总是自动调用，但了解这些寄存器单元有助于我们写出风格更加相符的 HDL 描述。
(6)移位寄存器单元。不带复位信号，所以我们在 RTL 描述中描述移位寄存器时一定不要出现复位信号。
(7)Slice/CLB。
我们可以根据这些基本单元的特性，有意识地写出风格相符的 RTL 描述，引导综合工具识别为其认可的基本单元的连接。

4.5/4.6 UART 设计在 Xilinx FPGA 的下载执行

首先确定使用到的外部资源：时钟、复位按键、串口通信端口。考虑到 FPGA 开发板自带时钟和我们需要的时钟频

率不符，还需要使用片内资源 PLL，通过配置 IP 的方式来进行。
接下来设计实验，并设计出对应的顶层文件。
本例中一种可行的实验设计是，串口接收到 rx_vld/rx_data 以后，直接引入 tx_vld/tx_data 上；
也可以新增读缓冲和发送按键，用一个 1KB 的 Block RAM 作为缓冲，将 rx 读到的数据按顺序写入其中，在按下发送按键后从该 RAM 的 0 地址取数据并通过 tx 端口发送出去。
最后设计约束文件（可以通过制造商提供的软件进行，也可手动编写），指定你设计的模块的端口和 FPGA 引脚的对应关系。（如使用 FPGA 制造商提供的现成约束文件则无此必要）
完成以后，经过综合、翻译、映射、布线（这三步统称为实现，implementation）、生成码流的过程，即可下载到 FPGA 执行。
这些过程每一步都有报告供设计者查阅，如综合报告中可以检查自己的设计是否被综合成了理想的底层元件(e.g. 上面第二种实验设计中的读缓冲是否真正综合成了 block RAM)。
使用串口通信线将 FPGA 开发板和 PC 等其它设备连接，然后通过这些设备上的对应工具（如 PC 上的“迷你终端”软件）进行测试。

第五章 ARM9 微处理器编程模型

5.1 ARM 公司历史 （略）

5.2 ARM 的处理器架构概况 （略）

5.3 微处理器基本模型

微处理器只需会简单的基本操作，而简单操作组合形成的一整套动作的意义不是微处理器的责任范围，而是由计算机程序决定。
微处理器向指令池发出取指令需求、得到指令、进行执行、然后接着取下一条，如果没有中断则会一直重复这个过程。
“中断”本身不提供指令给处理器执行，而是一种提醒处理器暂时中断当前指令执行流程、切换到某段固定指令段的机制，且 CPU 有权屏蔽。
RISC 指令对 CISC 指令集中的很多指令作了分解，如将 mov (%eax), (%ecx)分解为两条指令：先将(%eax)中的数据读入 CPU 内部的暂存寄存器，再将暂存寄存器的内容写入(%ecx)。
有了内部的暂存寄存器组，RISC 指令的操作可以分解为两个处理过程：(1)数据池与寄存器组数据交换、(2)取寄存器组的数据进行处理后写回寄存器组。处理数据池的数据需要循环进行 1-2-1 的流程。

5.4 ARMv4 架构模式

ARMv4 架构微处理器有 7 种运行模式：
用户模式 USR，限制访问；系统模式 SYS，用于运行特权级操作系统；特权模式 SVC，仅供操作系统使用的一种保护模式；
快速中断模式 FIQ，用于快速高优先级中断处理；外部中断模式 IRQ，用于一般的中断处理；
数据访问异常模式 ABT，取指或取数据发生异常时进入；未定义指令异常模式 UND，执行的指令未定义时进入。
处理器启动时处于 SVC，用于对各种资源进行初始化，然后进入 USR 或 SYS。
USR 模式对资源的访问是受限的，而且它无权修改 CPU 的模式，只有借助软中断指令进入 SVC 来修改。

SYS 和 SVC 一样是访问不受限制的。
如果处在上述三种模式时发生了 IRQ 中断或 FIQ 中断而且没有被屏蔽，则系统会进入相应的中断模式，中断处理完毕后会自动恢复到先前的模式。
如果出现上述两种异常，则进入相应的异常模式，进行异常处理，然后恢复到先前的模式。（一般在进入异常模式后、异常处理前，OS 会找到发生异常的相关指令重新执行一遍，确保不是误报。）

5.5 ARMv4 架构内部寄存器 （略）

5.6 ARMv4 架构的异常中断

ARMv4 架构处理器正常情况下工作在 USR 或 SYS 模式，在出现异常中断时进入相应的模式，并修改 PC 指向相应的值。
7 种异常中断：复位，进入 SVC；FIQ 快速中断，进入 FIQ；IRQ 中断，进入 IRQ；数据处理异常，进入 ABT；取指异常，进入 ABT；未定义指令异常，进入 UND；SWI 软件中断异常，进入 SVC。

5.7/5.8 ARMv4 架构的指令集和中断 （略）

第六章 兼容 ARM9 微处理器的 Verilog RTL 设计

6.1 确定 RTL 设计的输入输出端口

显而易见进行 RTL 设计首先需要根据功能确定输入输出端口。兼容 ARM9 微处理器的端口连接到指令池和数据池，按一定法则获取指令，然后按照指令操控存储在数据池的数据。
指令池和数据池的端口也是按需设计。微处理器发出读指令使能和指令地址后，在下一周期得到指令；指令池模型需要产生取指异常中断，所以需要在给出指令的同时指出本次取指令操作是否正常。故输入信号有时钟、读使能、读地址，输出信号有指令、取指异常标志。
数据池则需要读数据、写数据，而且还分为字操作、半字操作、字节操作，所有这些操作行为都通过同一套总线发送给数据池。为此，输入端口有：时钟、使能、写使能、字节使能、地址、写数据，输出端口有：数据输出、取数据异常标志。
考虑到很多 cache 并不能如理想状况那样在一个时钟周期内释放读写结果，增加一个端口 cpu_en 以使处理器能够暂停，等待 cache 读写。
CPU 需要实现 ARMv4 架构的 7 种中断和 20 条指令，故在之前的基础上再增加 3 个中断源输入（reset、IRQ、FIQ）。至此我们就完成了输入输出端口的设计。

6.2 经典三级流水线架构

在进行微处理器的 RTL 设计之前，我们必须确定整个微处理器的流水线架构。采用流水线架构是因为，指令从取出到执行完毕是一个相对复杂的过程，为了不让一条指令占用太多 CPU 时间，我们把它分成 3 或 5 步执行。
一般的三级流水线架构：取指、译码、执行。但对于数据池相关指令，把数据池数据载入寄存器无法在一周期内完成，因为数据池也需要一个时钟周期才能释放读数据结果。这样一来，这类 LDR 指令的执行必须花费 5 周期：取指、译码、算地址、访存、回写。
我们可以看见一条 LDR 指令可以使三级流水线停顿两拍。不幸的是，LDR 指令一般是相当常见的，为了防止其严重降低效率，我们从结构上改变流水线，使其影响消失。

6.3 经典五级流水线架构

五级流水线解决了 LDR 指令延时的问题，甚至可以认为，正是因为 LDR 需要 5 时钟周期，人们才修改流水线为五级的。一般的五级流水线架构：取指、译码、算地址、访存、回写。

但在两条相关（前一条指令的输出是后一条指令的输入）的 MOV 指令相连的时候，仍然必须在五级流水线的基础上延时 2 时钟周期（否则后一条指令将得到“过时”的结果）。这个问题可以修正，我们可以把前一条指令的回写结果提前使用，即后一条指令进入执行阶段时直接取出在流水线上尚未回写的上一条指令的结果（而非寄存器内过时的结果）。但当一条 LDR 连接一条相关的 MOV 时，该方法无效（因为需要使用的数据尚未取出），仍然必须延时 2 拍。

6.4 三级流水线改进架构

为了给数据池访问指令留出足够的时间，同时又不让寄存器处理类指令出现不必要的延时，可以设计出这样的一种三级流水线：

寄存器类指令和数据池的写指令同经典三级流水线，数据池读指令 LDR 延迟一拍变为四级——取指、译码、执行、回写。这里“执行”同时完成算地址和发出访存信号两个操作。

如果遇到正处于第三级的读指令的输出和处于第二级的指令的输入相等的时候，必须将后者暂缓执行，并在第三级插入一条空指令。

6.5 适用于兼容 ARM9 微处理器的三级架构

接下来我们考察具体如何用这套流水线实现 ARM9 指令。ARM9 指令分为两大类：数据池读写指令、寄存器处理指令。

数据池读写指令本身不涉及复杂的数据操作，相对复杂的是生成读写地址，读写地址一般是 Rn 加第二操作数生成，而最复杂的第二操作数是 LDR0 和 LDR1，是通过 Rm 或立即数移位生成的。所以，数据池读写指令的复杂数据公式为：Rm>>num + Rn。

寄存器处理指令中比较复杂的是乘法 MULT 和长乘法 MULTL，它们处理数据的公式是 Rm * Rs + Rn；此外还有 DP0、DP1、DP2，它们的公式是 Rm>>Rs + Rn 或 Rm>>num + Rn。

由此可见 ARM 指令集最多需要两步操作，第一步是由 Rm 和 Rs 进行移位或乘法操作得到第二操作数，第二步是由第二操作数和 Rn 进行加法为主的运算操作。由于乘法 MULT 和长乘法 MULTL 指令的存在，乘法运算无法避免，所以必须使用乘法器。那么我们需要考虑是否可以通过乘法器实现移位运算。

DP 指令有四种移位方式：逻辑左移、逻辑右移、算术右移、循环右移。乘法器显然能实现逻辑左移；对逻辑右移，Rn 和 1<<num 相乘的 64 位结果中，我们发现高 32 位就等于 Rn<<(32-num)，问题解决；对算术右移，如果符号位为 1 则将 Rm 取反再乘法再取反；对循环右移，将 64 位乘法结果的高 32 位和低 32 位进行或操作即得。

这样一来，每条指令都可以用一个乘法器串联一个加法器来实现。那么流水线的第三级就会是一个乘法器串联一个加法器，决定设计频率的关键路径就是这两者的耗时之和。

再考虑能否打破串联，以减小关键路径。我们发现，可以在第二级译码的时候顺便进行乘法运算得到第二操作数。至于译码操作本身，因为 ARM9 指令集非常简单，只需要 21 条 assign code_is_xxx = ...的语句生成非常简单的组合逻辑即可完成，不必设计专门的译码单元。这样就在不增加流水线级数的前提下打破了串联，缩减了关键路径，从而兼容 ARM9 处理器可以运行在更高频率上。

虽然如此，这一设计会带来数据冲突问题，因为乘法操作会从寄存器组中读取 Rm 和 Rs，如果第二级乘法正在读取 Rm 和 Rs，同时第三级的指令又在改写要读取的寄存器，就会造成数据冲突，与之前讲到的 LDR 的冲突一样。

6.6 影响流水线架构执行的四种状况

为了让每一条指令都可以使用这套流水线实现，需要考虑三级流水线之间所有可能的互动情况。
首先考察三级乘加结构的关键信号，因为有暂停流水线的需求，所以给每一级增加一个使能信号。这样一来，需要考虑的关键信号有rom_en、rom_addr、code、code_flag、cmd(第二级指令传递到第三级后的新名称)、cmd_flag、"Rm"、"Rs"、sec_operand(第二操作数)、Rn、to_vld、to_num(结果写入哪个寄存器及其使能信号)、cha_vld、cha_num(回写信号)、go_vld、go_num(回写信号在第四级的新名称)。
接下来逐个和逐对考察 ARM9 指令集的指令，对于所有可能影响流水线执行的指令或指令序列加以特殊处理。对跳转指令 BX，其判断条件是 to_vld == 1 && to_num == 4'hf，需要确保跳转指令后的其他指令不被执行，故在跳转指令执行到第三级时需要将 rom_en 赋值为 0 以使得处理器不再取其后的指令，然后在下一时钟周期将 cmd_flag 和 code_flag 置为 0 以阻止已被取出的后续指令的执行，并恢复 rom_en 为 1（这一时刻 PC 已被赋给新值)。还有另一种跳转的方式是 LDR PC, [R0]，在指令执行到第三级时也令 rom_en = 0，下一周期（指令执行到第四级）也令 code_flag = cmd_flag = 0，但由于此时 PC 还没有完成赋值，我们仍然需要再令 rom_en = 0，并在下一时刻仍然令 code_flag = cmd_flag = 0，判断条件是 go_vld == 1 && go_num == 4'hf。
当出现 LDR R1, [R0]和 MOV R2, R1 这种前后两条指令的输入输出相互关联的情况，即数据冲突时，必须在它们中间插入一个空指令。判断条件是 Rm 和 Rs 对应的寄存器和 to_vld、cha_vld、go_vld 对应的寄存器中有相同，编程时需要综合多种情况，用'|'连接（例如 code_rm_vld(当前指令使用的"Rm"是一个寄存器) & cha_vld & (cha_num == code_rm_num))，处理方法是在数据冲突的前一条指令执行到 T3 时置位 rom_en = 0 使得下一时刻流水线第二级的内容仍然是数据冲突的后一条指令，然后在下一时刻让 cmd_flag = 0 使得进入第三级的后一指令不执行（执行的是留在第二级的后一指令)。
多周期指令。简单起见先考虑两个周期的指令，以 SWP 为例，若 SWP 位于第三级，则首先让流水线等待（声明一个组合逻辑 hold_en 表示多周期指令正在执行，定义见后)，让 rom_en = 0，执行完第一个周期的工作之后让 cmd 从 SWP 变成 SWPX（SWP 的另外一种格式，表示只进行 SWP 的写操作部分，是单周期)，并且恢复 rom_en 且解除流水线的等待。对于多个周期的指令 LDM，这个指令的[15:0]中有多少位为 1 则表示还需要进行多少个周期的读写操作，最后再进行单周期的地址回写，仿照 SWP 的处理方式，我们置位 rom_en = 0，同时依次对[15:0]中为 1 的那些位对应的寄存器进行读写操作并让相应的位变为 0，到 cmd[15:0]变为全 0 之后就恢复 rom_en，解除 hold_en，并完成地址回写。判断条件 hold_en = cmd_ok & (cmd_is_SWP | cmd_is_MULTL | (cmd_is_LDM & (cmd_sum_m（cmd[15:0]的和） != 5'b0)))。
中断。(未完待续)

第七章 在兼容 ARM9 处理器内核上运行 Hello World 程序

7.1 基于 FPGA 的 SoC 设计流程

在进行具体的 SoC 设计时，处理器的指令池、数据池、中断必须具体化到设计实体中：
指令池一般具体化为 ROM，数据池则在实现不同功能时例化为 ROM、RAM 或者寄存器（寄存器通常连接外设，对外设定义行为规则或交换数据）中的不同种类，IRQ 和 FIQ 中断的功能也应该予以定义。
这样一来，设计过程就从以寄存器为最小单元的逻辑设计转变为以微处理器为核心、带 ROM、RAM 和寄存器的独立嵌入式系统为中心的逻辑设计；
输入端口作用于该嵌入式系统的两个途径是：输入端口连接到微处理器的中断端口，或者连接到寄存器中改变某个寄存器的值；

输出端口连接某个寄存器，嵌入式系统写入这个寄存器时输出改变。
接下来在进行嵌入式软件的编写之前，必须将 ROM、RAM、寄存器分配到具体的地址，这是嵌入式系统运行的内环境；
ROM 和 RAM 直接分配连续的地址范围即可，但 ROM 和 RAM 的不能重叠；
寄存器的地址是离散的，可以直接选用某种成熟的单片机外设寄存器设定情况，也可以自定义访问规则并写在.h/.c 文件中。
在内环境定义完毕后，参照这个内环境进行 SoC 硬件和嵌入式软件的设计工作；
工作结束后，嵌入式软件生成一段.bin 代码，SoC 硬件将这段.bin 代码置入 ROM，即完成了一个全面的 SoC 设计。
欲完成输出"Hello World"的 SoC 设计，首先容易想出，除了实现嵌入式系统之外，还需给 UART 模块配置专用寄存器，以实现对串口收发数据的操作；
第一步是定义兼容 ARM9 内核的工作内环境，第二步是定义上述的专用寄存器（包括对应的行为规则）。

7.2 使用 RealView MDK 编译 Hello World 程序

在嵌入式软件 IDE——RealView MDK-ARM 中建立工程，依次定义好单片机具体型号、编译设置(如不生成 thumb 指令)、其它命令（如从生成的.axf 文件中解析出.bin），开始开发。
示例工程中的文件用途：startup.s，完成配置单片机和设定栈，初始化各个模式下的状态；Hello.c，主函数所在文件；Retarget.c，重定位 printf 依赖的一些函数使其定位到 Serial.c 的相关函数中；Serial.c，描述串口底层函数。

7.3 仿真输出 Hello World

μVision 中编译产生的.bin 文件的内容就是一行一行的 ARM 指令。我们可以用 Verilog 写一个 testbench 文件，直接用$fopen 和$fread 函数打开这个.bin 文件，将其内容读入到 ROM（指令池）中，并解析兼容 ARM 内核，最后通过$write 输出 Hello World 字符。
可以通过在波形中添加 ROM 相关的信号（rom_en/rom_addr[31:0]/rom_data[31:0])来观察指令的提取情况，添加 RAM 相关的信号（ram_cen/ram_wen/ram_flag[3:0]/ram_addr[31:0]/ram_wdata[31:0]/ram_rdata[31:0]）来观察数据池的读写情况，也可添加 r0~re 来观察微处理器内部寄存器组的变化情况。
（代码、运行结果见附录）

7.4 建立 Hello World 的 FPGA 设计工程

兼容 ARM9 处理器可以与嵌入式编译软件 μVision 完美结合：只要从 μVision 编译出.bin 文件，转换成.coe，读入 ROM 中，即可让 CPU 按嵌入式程序的指令工作。
仿照第一节中所说的流程，不难画出以 ARM9 内核为核心的 SoC FPGA 的设计框图，具体包含兼容 ARM9 内核、UART、ROM 和 RAM（对应指令池数据池)、寄存器。采取自顶向下的方式，来完成一个完整的设计。
ROM：生成 IP，读入 hello.bin。IP 的设置根据实际需要设置为双口 ROM，各个端口的宽度、读写方式等也根据 CPU 总线情况而定，深度则根据 μVision 中设定的 ROM 大小除以宽度计算而来，根据需要为每个端口都配备 enable 引脚，并指定 init file（因为只接受.coe 格式的 init file，所以需要在仿真过程中通过$fopen、$fread、$fdisplay 将.bin 转换成.coe)。
RAM：生成 IP。大小由 μVision 中的设定指定，宽度为总线宽度 32 位，因为还涉及到字节使能 ram_flag[3:0]，故勾选 byte write enable。
寄存器：已经在μVision 中设定完成。板上实现是功能模拟（而不一定必须分配到指定的物理地址)，将寄存器对应的地址（ram_addr == 32'he000_0004）写在其涉及的功能语句中即可（如 tx_vld <= ram_cen & ram_wen &

(ram_addr == 32'he000_0004))。
（运行结果见附录）

第八章 兼容 ARM9 处理器内核性能测试 Dhrystone Benchmark

8.1 Dhrystone 2.1 简介

Dhrystone 主要测试微处理器的整数性能，嵌入式处理器由于面积和低功耗限制会将其性能向整数和逻辑运算倾斜，所以用 Dhrystone 测试嵌入式处理器更能体现其测量价值。ARM 公司对其旗下的处理器内核一般采用 Dhrystone 2.1 版本进行测试。
μVision 的 DHRY 例程包含 7 个源码文件：startup.s、retarget.c、serial.c、time.c、dhry.h、dhry_1.c、dhry_2.c。前三个文件同第 7 章的 3 个文件，第 7 章在使用之前修改了这 3 个文件，我们将修改过的文件拷贝过来覆盖原来的 3 个文件。time.c 是一个定时器，用于计算处理器完成 Dhrystone 的耗时；dhry_1.c 和 dhry_2.c 是 Dhrystone 测试的主程序。
值得一提的是，Dhrystone 是以测试经过的时钟周期数作为测试时间的，所以受试处理器 1s 跑完 Dhrystone Benchmark 的次数还需要除以时钟频率(MHz)才能得到最终的性能指标 Dhrystone MIPS。另外，运行次数需要足够大。

8.2 移植 Dhrystone 2.1 进行编译

首先，我们根据实际情况对于在 FPGA 运行上不必要的代码进行删改。然后，为了避免大量修改仿真文件，我们将该工程的器件修改为之前的 LPC2100 系列，但是 LPC2101 的 ROM、RAM 太小以至于无法编译，尝试发现最低需要设为 LPC2104。最后参照上一章的操作将编译生成的.axf 转换成.bin。
当然了，为了获得尽可能高的执行效率，我们需要根据 ARM 公司提供的指导 Dhrystone Benchmark 编译的文档，设置合适的编译参数，如-O3 -Otime --no_inline --no_multifile。

8.3 仿真运行 Dhrystone Benchmark

我们直接拷贝上一章的 testbench 文件进行修改，内容包括修改时钟频率为 1MHz、按 LPC2104 设定扩大 ROM/RAM 大小、定时发送 IRQ 中断（因为定时中断是 time.c 中用到的计时方式）。
接下来还需要将输入仿真次数的语句改为直接赋值，因为仿真环境无法输入数据。再仿照上一章的方法启动仿真，但遇系统提示 Out of heap memory，故将 startup.s 中定义的 Heap Size 扩大，再次运行略多于 3s 的仿真时间（测试需 3s，还需若干毫秒打印完整的测试信息），得到测试结果。
（运行结果见附录）

8.4 在线可编程的 FPGA SoC 设计工程

按上一章的设计，每修改一次 ROM 的内容就必须重新综合一次，时间消耗巨大。我们可以设法通过 UART 将.bin 文件发送到 FPGA 开发板的 ROM 中，解决这个问题。
这样就必须使得 ROM 可以设置为可写状态，从而这个 FPGA SoC 设计工程有两个状态：工作状态和编程状态（处理器内核不工作，ROM 可写）。所以我们将 SW[0]连接到 cpu_en 和 ROM 的 wea(Port A Write Enable)信号。并且需要更改设计，使得可以每从串口接收 4 字节数据就送入 ROM 一次。最后还需要按需添加定时发送中断的逻辑。
还需注意，处理器切换到工作态时必须手动复位一次，以使得 PC 从 0x0 再次开始工作。
这样一来，FPGA 就可以在线编程，从而可以认为是完整地模拟了单片机开发板的功能。

8.5 Dhrystone Benchmark 在开发板中运行

在烧录二进制码流文件后，开发板会按照 ROM 中设定好的初始内容运行。为了在线修改 ROM，我们可以将上节指定的 SW[0]扳到输出为 1'b0 的一端，然后通过串口调试助手发送.bin 文件到开发板，发送完毕后将 SW[0]置为 1 并按下复位按钮，开发板即按照新写入 ROM 的内容运行。
在运行 Dhrystone Benchmark 时，开发板会通过串口询问运行次数，我们通过串口发送运行次数到开发板（注意发送格式），即可进行 Dhrystone 测试。
（运行结果见附录）

第九章 uClinux 仿真——结合 SkyEye 启动不带 MMU 的操作系统

9.1 ARM7TDMI-S 处理器内核

首先了解在单片机和低端手机市场相当活跃的 ARM7 系列。ARM7 系列微处理器包括 ARM7TDMI、ARM7TDMI-S 等四种类型，ARM7TDMI 的使用最为广泛，其名称的含义是：T 代表 thumb 指令集（thumb 指令是 16 位的，所有 thumb 指令都有对应的 32 位 ARM 指令形式，所以使用 thumb 指令集有助于节省 ROM 存储空间），D 表示支持片上 debug（通过 JTAG 并口），M 表示内嵌硬件乘法器（但是为了降低面积和功耗，只是 32×8 的乘法器），I 表示嵌入式 ICE-RT 逻辑（in circuit emulator real time，使处理器支持片上断点和调试点）。ARM7TDMI-S 则是 ARM7TDMI 的可综合的软核形式。
ARM7TDMI-S 采用三级流水线：取指、译码、执行。它采用冯诺依曼结构，指令池和数据池通过同一个 memory interface 访问，这套 memory interface 与 ARM 公司推出的 AMBA 总线高度相似。它有 4 种传输类型：内部周期（通常情况下不传播数据时）、非连续传送周期（传送单个数据）、连续传送周期（用于 LDM 等需连续传输多个数据时）、协处理器寄存器传送周期（处理协处理器指令时）。
ARM7TDMI-S 作为可综合的软核，也拥有我们使用 module 描述时的 input 和 output 端口。它拥有时钟输入、中断接口、大小端配置接口、总线控制信号、调试接口、ICE-RT 访问、memory interface、协处理器接口等一系列端口，可以查询用户手册来获取每一个端口的详细信息。
ARM7TDMI-S 内部主要分为三个部分：Embedded ICE-RT macrocell、Embedded ICE-RT TAP controller、CPU 内核。前两个部分服务于 ICE-RT，接收用户的调试指令并转而对 CPU 内核发出相应的指令，包括暂缓 CPU 内核的执行和对 memory interface 发出读写指令。CPU 内核与本书的兼容内核功能大致相同，也是不断取出指令译码执行，但可以支持连接协处理器。值得一提的是，CPU 内核不能直接参与调试，其支持的调试操作只有暂停执行，类似于对本书的兼容内核的 cpu_en 置 0。
ARM7TDMI-S 的 CPU 内核符合经典三级流水线模型，从 memory interface 中读取指令，通过 instruction pipeline 进行指令译码并将译码结果通过传给 register bank，由 register bank 运算得到第二操作数，第二操作数和 Rn 一起送入 ALU 即可进行地址或数据运算，然后交由 address register 读写内存或传给 register bank 写回寄存器。遇乘法指令时由 32×8 乘法器和 barrel shifter 共同完成运算。ARM9TDMI 和 ARM7TDMI 类似，在 ALU 和 regbank 之间有两个可选部件 barrel shifter 和 multiplier，它们的关键路径是 MUL+ALU 或 shifter+ALU 中的一条。因为它们的关键路径是两个模块的叠加，所以它们在 FPGA 上运行的时钟频率不能达到很高，约 25MHz。
通过本节对 ARM7TDMI-S 内核的分析，可以看出这些处理器内核并不是非常神秘的“天外来客”，它也是具有输入输出端口的 RTL 描述，也是在对处理器内核完成描述后添加各种调试功能接口。只要在设计上下足功夫，读者也能设计出非常优秀的处理器内核。

9.2 以 ARM7TDMI 为核心的单片机

处理器内核需要配以外设才可以完整地实现其功能，就如大脑与手足、眼睛、皮肤等的关系。ATMEL 公司的 AT91 系列单片机就是一款应用 ARM7TDMI 核心的优秀单片机。

这款单片机有 100 个引脚(接口)，包括 EBI(external bus interface，连接片外存储)、AIC（连接外部中断)、TC（定时器)、USART、PIO（并行 IO 引脚)、CLOCK（主时钟输入输出)、WD（Watchdog 溢出输出引脚)、Reset、ICE（在线调试引脚，它通过 JTAG 接口直接与 ARM7TDMI 的 ICE 接口相连)、Power（提供 VDD、GND 引脚）和一些可以由用户自定义的引脚。这些外设通过总线与处理器内核连接，ARM7TDMI 通过预定义的寄存器得知外设的状况并通过写寄存器来操控外设。指令池和数据池除了使用内置 ROM 和 SRAM，也可以通过 EBI 接口使用外部存储空间。

单片机的内部，ARM7TDMI 和单片机 ICE 接口直接连接，ARM7TDMI 通过高速的 ASB 总线管理 RAM、ROM、寄存器外设，ASB 总线通过 AMBA Bridge 桥接低速的 APB 总线，APB 总线连接 AIC（接收片内外中断信息并直接传给 ARM7 内核）等一些简单低速外设和 PIO controller（将可自定义引脚配置为用户定义的形式)。

9.3 μClinux/uClinux 嵌入式操作系统

MMU 给予操作系统使用虚拟地址的能力，对地址空间进行更加灵活的管理。但在很多小型的嵌入式系统中，仅凭物理地址即可满足内存管理需求了，出于减少系统复杂程度、降低硬件及开发成本、降低功耗的考虑，遂在处理器的硬件设计中取消了 MMU 模块。

与之相适应，也产生了经过小型化改造的 Linux 操作系统——uClinux。uClinux 保留了 Linux 的大多数优点：稳定、可移植性、网络功能、对各种文件系统完备的支持、标准丰富的 API。它编译的目标文件不会超过数百 KB 的数量级，特别适用于少内存或 flash 的单片机。它采用的文件系统是 romfs，比 ext 文件系统节约空间，但不支持动态擦写保存，对于系统需要动态保护的数据采用虚拟 RAM 盘的方法处理。

9.4 SkyEye 硬件模拟平台

正如 RTL 设计可以进行 Modelsim 仿真，uClinux 的内核可以在 PC 上进行软件模拟，SkyEye 就是一个这样的模拟平台。SkyEye 的目标是在通用的 Linux 和 Windows 平台上实现一个纯软件的 IDE，模拟运行（指令级模拟）常见的嵌入式计算机系统，对它们和它们的各种系统软件如 TCP/IP、图形子系统、文件子系统进行源码级分析调试。

下载并在 Linux 下安装 SkyEye。下载对应 AT91 系列单片机的 uClinux 内核的 bin 代码 uclinux0（无扩展名，可执行文件格式)、romfs 文件格式的文件系统 boot0.rom，放在同一目录下，并在该目录下创建 skyeye.conf，指定 CPU 类型、单片机平台、ROM/RAM/寄存器空间的地址分配、log 信息。接下来在终端下进入该目录，执行命令"skyeye -e uclinux0"，即可启动一次 uClinux 操作系统在 SkyEye 上的模拟运行，并打印出 log 信息。

uClinux 的启动分为两个阶段，第一个阶段是 bootloader 启动阶段，第二个阶段是 Linux 内核的初始化和启动。log 信息中,起始行到"Linux version ..."之前的部分是 SkyEye 的运行信息;该行起至"POSIX conformance testing by UNIFIX"一行止是第一阶段的运行信息，打印了 Bootloader 的头信息、版本、编译时间，并运行硬件初始化工作；剩下的从"Linux NET4.0 for Linux 2.4"起的部分是第二阶段的运行信息，这一阶段内核初始化，然后打印出 uClinux 的 log 标志，并把控制权交给 Sash command shell。

（运行结果见附录）

9.5 Modelsim 下仿真 uClinux 启动过程

类似 SkyEye，我们可以利用兼容 ARM7/9 的处理器内核，加载 uClinux 的 bin 文件，模拟其启动过程。当然，在仿真过程中，需要不断参照 SkyEye 的运行结果（作为标准输出）调试我们的仿真源码。另外值得一提的是，我们

只有处理器内核，但 uClinux 的运行依赖单片机，它会访问单片机的外设，包括串口、定时器、watchdog、PIO 控制器等，虽然我们可以为每个外设写一个行为模型，但本次仿真只模拟其启动过程，所以我们可以简化，对于不经常访问的寄存器直接返回能使 uClinux 正常启动的期望的返回值，这些数值从 SkyEye 文档或 AR91 系列单片机文档中可以得知。

下面来看仿真源码具体如何设计。首先，从 https://code.google.com/archive/p/arm-cpu-core/下载仿真文件包，其中包含 uclinux0.bin 和 root 文件系统 boot0.bin，在 testbench 中声明两块与文件大小匹配的存储空间（如 reg[7:0] boot_all[383999:0]），将这两个文件读入；并且按照 SkyEye 对 ROM/RAM 空间的定义（共定义了 7 块，但仿真只用到 4 块），初始化 4 块 ROM/RAM 空间，其中 2 块的前一部分加载文件内容，剩余部分和另外 2 块空间全部初始化为 0。

uClinux 的数据池和指令池初始化完毕，现在把兼容 ARM9 处理器内核例化到 testbench 内、生成 100MHz 时钟信号、对处理器内核进行初始化，需要注意的是，因为系统按照 skyeye.conf 中的设定从 0x0100_0000 开始取第一条指令，因此需要修改处理器内核的源码，设定复位地址为 0x0100_0000。

接下来描述指令池和数据池的行为。指令池根据地址高位部分的不同，从 4 块 RAM 空间中的前 3 块送出对应指令。至于数据池，我们把每次对数据池的读写（以字为单位）按字节分为 4 部分各用一个 always 语句描述，按照地址可以分为对 RAM/ROM 的读写和对寄存器的读写，前者与指令池同理实现，对于后者，由于我们只是仿真，因此可以忽略对这些寄存器的写，而对于寄存器的读则直接返回能使 uClinux 正常启动的期望值。但有一个例外，若需要对串口的控制寄存器进行读写，我们需要将读写的内容在屏幕上完整地展现出来，这样才能得到仿真输出信息；AR91 系列单片机通过串口打印的方式是先向指定的寄存器中写入地址，然后向另一指定寄存器中写入字符串的长度，串口会自动从 RAM 中该地址开始取出给定长度的字符串打印，我们在仿真代码中参照实现即可。

最后我们描述 IRQ 中断，这是为了模仿 TC1 定时器的行为，其每计数一定的时间就生成一个 IRQ 中断（但我们不需要研究 TC1 的具体触发原理，只需要在 uClinux 启动过程中应该发出中断的时间点发出中断即可，可以直接用 26 条#xxxxxxx irq = 1; #10 irq = 0;来实现），这些 IRQ 中断用于计算 BogoMIPS，即语句"Calibrating delay loop... 13.00 BogoMIPS 代表的计算过程。

至此我们就完成了 testbench 的设计。仿真 90ms，就可以在 Transcript 窗口中看到 uClinux 的启动信息了。（运行结果见附录）

第十章 Linux 操作系统仿真——仿真 mini2440 模型启动带 MMU 的操作系统

10.1 ARM920T 处理器内核

和前面介绍过的 ARM7TDMI 系列相比，ARM9TDMI 采用数据池和指令池访问分开的哈佛结构，性能更佳。ARM9TDMI 系列采用五级流水线：取指、译码、执行、访存、回写。ARM9TDMI 系列包括三个产品：ARM9TDMI，裸内核；ARM920T，内核+cache+存储器管理单元 MMU；ARM940T，ARM9TDMI+cache+保护单元 MPU。这里的 cache 都是每颗 CPU 两个 cache，分别针对数据和指令。

ARM920T 以 ARM9TDMI 为核心。ARM9TDMI 带有和 ARM7TDMI 类似的调试接口、协处理器接口。ARM920T 中，指令池和数据池模型由自带 cache 充当，cache miss 时通过 AMBA 总线访存；如果是写数据池的操作，则由专门的 write buffer 模块完成回写。服务于指令池和数据池的各有一个 MMU，若软件需要控制 MMU 则通过协处理器 CP15 来完成。CP15 共有 15 个寄存器，软件若想控制 cache 或启动 MMU，则通过协处理器访问指令 MCR 或 MRC 对这 15 个寄存器进行访问，来达到控制的目的。这 15 个寄存器包括 ID code、cache type、cache &

MMU control、TLB operation register 等。
cache 的具体作用是对数据或指令进行缓存。MMU 有两个功能：对数据池或指令池的访问地址进行转换、对访问的权限进行检查。具体而言，为了将虚拟地址转换成物理地址，MMU 从内存中找到该地址域对应的转换页，把虚拟地址的高位部分替换得到物理地址；并且从转换页中可以读到该部分地址域的权限信息，结合 CPU 的状态即可判定这次访问是否合法。

10.2 S3C2440A 32 位微控制器

S3C2440A 微控制器是以 ARM920T 处理器为核心的一种单片机，采用 130nm CMOS 标准工艺制造。三星为其配备了大量外设接口，充分发挥 ARM920T 强大的处理能力。
S3C2440A 通过两条总线连接外设，高速的 AHB 总线和低速的 APB 总线，后者连接对速度要求不高的外设，并通过 AMBA Bridge 和 AHB 总线相连接。外设的种类有：外部 Memory 控制器、LCD 控制器、DMA 控制器、UART 串口、定时器、实时时钟、通用 I/O 端口、外部中断源等。这些外设通过总线连接并将控制权交给处理器内核。

10.3 mini2440 ARM9 开发板

S3C2440A 只是一颗芯片，不能独立工作，为了发挥它的高性能，必须在一块 PCB 板上为其各个外设接口连接上相应的外设。mini2440 ARM9 开发板就是这样的设备，连接了 LCD 显示、100M 以太网 RJ-45 接口、串口、USB host、USB slave B 型接口等。mini2440 ARM9 开发板的“大脑”是三星 S3C2440 微控制器，主频 400MHz，最高可达 533MHz；SDRAM 大小 64MB、工作频率可以高达 100MHz，数据总线宽度 32 位；我们烧录的嵌入式程序主要放在 Flash 存储中，板上有两种 Flash 存储，256MB 的 NAND Flash 和 2MB 的 NOR Flash。它们提供了处理器工作的数据池和指令池模型，是使得微控制器可以工作的最低需求。
为了理解 mini2440 的运行原理，我们按照对处理器运行模型划分的 3 大部分 ROM、RAM 和寄存器的顺序阐述。模型中的 ROM 对应着开发板上的 NOR Flash 或 NAND Flash，两者都可以用于启动整个开发板；不同之处在于 NOR Flash 的每一字节都有地址编号，可以直接在 AMBA 总线上发出地址来访存，因此可以直接用于启动开发板。NAND Flash 存储的数据在总线上没有规定的地址信息，访问它必须通过对一个寄存器写入读写操作，然后写入相对地址信息，才能从另一个寄存器中读出该地址对应的数据；它不能从 AMBA 总线直接访问，也就无法直接从 NAND Flash 中取出指令，所以不能直接用于启动开发板；当然本开发板上的 NAND Flash 进行了改造，包含一个 4KB 的 boot SRAM，默认编址下它位于最高地址处，若需要将 NAND Flash 用于启动，则将其编址到 0x0，这 4KB 的 SRAM 存放 boot loader，它将 NAND Flash 存放的 Linux 内核复制到 SDRAM 内，然后跳转到 SDRAM 开始执行。
模型中的 RAM 是由 64MB 的 SDRAM 来充当的，其每个字节数据都有对应地址。模型中的寄存器则较为复杂，mini2440 的外设很多、且每个外设都有固定的寄存器和行为规则，嵌入式工程师需要查阅用户手册并按这个规定编写程序，当然在开发 mini2440 开发板时也需要按 S3C2440A 的引脚分布图把各引脚链接到相应的外设。

10.4 NAND Flash 仿真模型

mini2440 开发板上，Linux 内核及系统文件都是存放在 256MB NAND Flash 内，因此需建立一个 NAND Flash 的仿真模型，装入 mini2440 开发板中导出的 image，按照输入的访问地址将 image 对应位置的数据送给处理器内核。
S3C2440A 微控制器通过读写寄存器的方式来获取 NAND Flash 的内容。读 NAND Flash 的过程：对控制寄存器发出读操作，对地址寄存器写入读地址，轮询读取状态寄存器，若读取状态寄存器回答成功，则从读数据寄存器取得该地址数据。写 NAND Flash 过程类似。
设计此仿真模型，首先定义它的 RTL 输入输出端口，然后利用 initial 语句对 image 进行装入，最后只需非常简单

的两条 assign 语句（按分区计算真实地址、读数据）即可完成按地址读数据的操作。

10.5 为兼容 ARM9 处理器内核增加协处理器指令

在 ARM920T 中，为了实现对 MMU 和 cache 的管理，处理器会不断对协处理器 CP15 的各个寄存器进行读写。为了能够进行仿真，必须为兼容 ARM9 处理器内核增加协处理器指令的支持。协处理器指令分为两类：MRC，将处理器内部寄存器组(R0-R15)的某个寄存器送入协处理器的寄存器；MCR，将协处理器的某个寄存器送入寄存器组。因为仿真中实际并不存在协处理器 CP15，所以我们忽略所有 MRC 指令，只实现 MCR 指令。

首先，我们参照 MCR 指令的格式在处理器内核中添加 code_is_mcr 和 cmd_is_mcr；然后，修改 all_code 的表达式，向其中添加 MCR 指令；增加一个输出端口 output logic inst_mcr，表示 MCR 指令即将进入第三级流水线、需要协处理器在下一时钟周期给出寄存器值作为应答；增加一个输入端口 input logic[31:0] mcr_data，用于输入 MCR 的寄存器值；修改 to_vld、to_num、to_data，使得 cmd_is_mcr 生效时，亦即 MCR 位指令在第三级时 mcr_data 能够进入对应的寄存器。

10.6 仿真启动 Linux 操作系统

首先在 testbench 文件中例化 ARM9 内核和 NAND Flash 模型，其中 NAND Flash 的设计文件中需要通过 initial 语句装入 mini2440 的 NAND Flash 镜像(bin 文件)。

梳理 Linux 操作系统在 mini2440 上的启动过程，可以分成 3 大步骤：首先，NAND Flash 的前 4KB 作为 bootloader 执行，一些初始化工作结束后，它把 NAND Flash 内存放的 Linux 内核拷贝入 SDRAM 内；bootloader 结束后，跳转入 SDRAM 内执行，64MB SDRAM 同时充当 ROM 和 RAM，此时处理器内核发出的地址是物理地址，可以直接从 SDRAM 内读指令或数据，直至处理器写协处理器寄存器，启动 MMU 功能；MMU 启动后，程序依然在 SDRAM 内执行，地址必须进行转换，只不过转换后的物理地址依然指向 SDRAM。

由此可见，SDRAM 是程序执行的主战场。所以我们接下来对 SDRAM 进行模拟：声明一个 64MB 的 reg 数组 sdram，初值为 0。仿真的时候不需要实现 cache，对于给出的地址 addr 直接返回 sdram[addr]即可。

然后模拟 bootloader，声明 reg[7:0] rom[4095:0]，用读入的 NAND Flash 映像的前 4KB 初始化。然后就可以写出指令池模型，第一个阶段返回 rom 中对应地址的内容，后两个阶段返回 sdram 中的内容，若不属于合法地址则报错显示"rom physical address overrun"。访问 sdram 时的地址应该用 rom_addrm（MMU 转换后的物理地址）而非 rom_addr，因为第三阶段是启用 MMU 的。

接下来模拟 MMU。定义使能信号 mmu_enable = 1'b0，用一个 always 语句描述其更改：如果 CPU 内核执行 MRC 指令对 CP15 的 Register 1: Control 的 bit[0]位写入数值，就将这个数值赋给 mmu_enable。虚拟地址向物理地址的转换通过页表实现，页表的首地址存放在 CP15 的 Register 2: translation table base(TTB) register 中（用一个 always 语句描述处理器内核对该寄存器的写入），有了 TTB 和虚拟地址 ram_addr 即可查找相应的页表项，对应页表项的地址为{ttb, ram_addr[31:20], 2'b0}（每 1MB 虚拟地址空间共用一个页表项），然后再根据得到的页表项的低两位判断查表情况，每种情况对应的页表项都有不同的结构，低两位为 11 时得到直接转换的物理地址{entry_data[31:20], ram_addr[19:0]}，为 00 时发出数据访问异常中断，置位 ram_abort=1，其他两种情况需查二级页表，二级页表查表过程类似。rom_addr 到 rom_addrm 也要经过这种转换。

上面我们完成了启动的 3 大步骤的设计，最后我们还需要模拟外设。因为这个镜像是从 mini2440 开发板中直接复制出来的，所以启动时会检查、初始化板上的各种外设。对每种外设进行行为级模拟代价太高，考虑到我们只需要模拟 boot 过程，所以除了串口以外，我们可以虚拟所有外设，对外设的写操作我们忽略，对外设的读操作我们直接返回能使 Linux 正常启动的期望的返回值，这些数值取决于 mini2440 开发板。具体来说，只需要在设计 ram_rdata 时，若 ram_addrm 指向了外设寄存器的地址，就给 ram_rdata 赋以这些故意设计的返回值。当然也会有少数例外，例如读取定时器的技术状态时应回答真实的值。还需实现串口的打印，若对串口寄存器写入了值，我

们就直接将其按字符打印到屏幕上($write)。
至此我们就完成了 testbench 的设计。执行命令"run -all", 就可以在 Transcript 窗口中看到 Linux 的启动信息了。完整的启动过程大约需要仿真 7~8s 才完成。在仿真的最后阶段，可能由于某些寄存器的设定不到位，会有略微不同，书上的仿真结果仅供参考。
（运行结果见附录）

代码

2.5 RTL 设计实例：UART 串口通信

```
module rxtx(
    input logic clk,
    input logic rst,
    input logic rx, //接收端口
    input logic tx_vld, //发送有效（请求发送）
    input logic[7:0] tx_data, //发送数据

    output logic rx_vld, //接收有效
    output logic[7:0] rx_data, //接收数据
    output logic tx, //发送端口
    output logic tx_rdy //发送端口就绪
);
    //用 2-3 个寄存器同步来消除异步传输的不确定性
    logic rxx;
    logic rx1, rx2, rx3;
    always_ff @ (posedge clk)
    begin
        rx1 <= rx; rx2 <= rx1; rx3 <= rx2; rxx <= rx3;
    end

    //检测 rxx 的变化，该检测结果为计数器服务
    logic rx_change;
    logic rx_dly;
    always_ff @ (posedge clk)
        rx_dly <= rxx;
    assign rx_change = (rxx != rx_dly); //用一个周期之前的值与当前值比较，得到变化与否的标志
```

```
    //时钟计数器，在 rx 保持为同一个值的时候，用这个值持续的时长来计算持续了多少位。
    logic[13:0] rx_cnt;
    always_ff @ (posedge clk)
    begin
        if (rst) rx_cnt <= 14'b0;
        else if (rx_change || (rx_cnt == 14'd2603)) rx_cnt <= 14'b0;
        else rx_cnt <= rx_cnt + 1'b1;
    end

    //采样标志
    logic rx_en;
    assign rx_en = (rx_cnt == 14'd1301); //半周期时刻，该采样了

    //接收数据状态指示寄存器，是否开始接收数据
    logic data_vld;
    always_ff @ (posedge clk)
        if (rst) data_vld <= 1'b0;
        else if (rx_en && ~rxx && ~data_vld) data_vld <= 1'b1; //rxx 接收到一个低电平信
号，说明接收将在一周期后开始
        else if (data_vld && (data_cnt == 4'd9) && rx_en) data_vld <= 1'b0;
        else;

    //专用于接收过程的计数器，含义是当前接收过程已接收了多少位（不含起始位），每接收 1 位则值加 1
    logic[3:0] data_cnt;
    always_ff @ (posedge clk)
        if (rst) data_cnt <= 4'b0;
        else if (data_vld) //在接收已经开始后，每周期计数
            if (rx_en) data_cnt <= data_cnt + 1'b1;
            else; //latch
        else data_cnt <= 4'b0;

    //接收到的前 8 位（0-7）：数据位
    //logic[7:0] rx_data;
    always_ff @ (posedge clk)
```

```
        if (rst) rx_data <= 8'b0;
        else if (data_vld & rx_en & ~data_cnt[3]) rx_data[data_cnt] <= rxx; //rx_data
<= {rxx, rx_data[7:1]};
        else;

    //接收完毕，发送接收有效信号
    //logic rx_vld;
    always_ff @ (posedge clk)
        if (rst) rx_vld <= 1'b0;
        else rx_vld <= data_vld && rx_en && (data_cnt == 4'd9);

    //发送模块，接收到请求发送信号时暂存发送数据
    logic[7:0] tx_rdy_data;
    always_ff @ (posedge clk)
        if (rst) tx_rdy_data <= 8'b0;
        else if (tx_vld && tx_rdy) tx_rdy_data <= tx_data;
        else;

    //发送数据状态指示寄存器
    logic trans_vld;
    always_ff @ (posedge clk)
        if (rst) trans_vld <= 1'b0;
        else if (tx_vld) trans_vld <= 1'b1;
        else if (rx_en && trans_vld && (trans_cnt == 4'd10)) trans_vld <= 1'b0;
        else;

    //发送过程的计数器
    logic[3:0] trans_cnt;
    always_ff @ (posedge clk)
        if (rst) trans_cnt <= 4'b0;
        else if (trans_vld)
            if (rx_en) trans_cnt <= trans_cnt + 1'b1; //采样信号只需要在计数器中判断，其它
过程中不必
            else;
        else if (!trans_vld) trans_cnt <= 4'b0;
```

```
    //发送过程
    //logic tx;
    always_ff @ (posedge clk)
        if (rst) tx <= 1'b1;
        else if (trans_vld)
            if (rx_en)
                case (trans_cnt)
                4'd0: tx <= 1'b0;
                4'd1: tx <= tx_rdy_data[0];
                4'd2: tx <= tx_rdy_data[1];
                4'd3: tx <= tx_rdy_data[2];
                4'd4: tx <= tx_rdy_data[3];
                4'd5: tx <= tx_rdy_data[4];
                4'd6: tx <= tx_rdy_data[5];
                4'd7: tx <= tx_rdy_data[6];
                4'd8: tx <= tx_rdy_data[7];
                4'd9: tx <= ^tx_rdy_data;
                4'd10: tx <= 1'b1;
                default: tx <= 1'b1;
                endcase
            else; //latch
        else tx <= 1'b1;

    //发送就绪
    //logic tx_rdy;
    assign tx_rdy = ~trans_vld;

endmodule
```

3.4 仿真实例：UART 串口仿真

```
`timescale 1ns / 1ps

module rxtx_tb();
    reg clk = 1'b0; //初值
```

```
always clk = #20 ~clk;


reg rst = 1'b1;
initial #40 rst = 1'b0;


reg RxD = 1'b1, tx_vld = 1'b0;
reg[7:0] tx_data = 8'h0;
logic rx_vld, TxD = 1'b0, tx_rdy; //从 X 变成 0 也会算作一次 negedge，然后触发一次意料之外的 Tx 的检测过程。所以需要赋初值为 0.
logic[7:0] rx_data;


rxtx u_rxtx(
       .clk    (clk),
       .rst    (rst),
       .rx     (RxD),
       .tx_vld (tx_vld),
       .tx_data(tx_data),


       .rx_vld (rx_vld),
       .rx_data(rx_data),
       .tx     (TxD),
       .tx_rdy (tx_rdy)
);


task rx_send;
input[7:0] b;
integer i;
begin
    RxD = 1'b0; //#100
    for (i = 0; i < 8; i = i+1) #104167 RxD = b[i]; //#104267
    #104167 RxD = ^b; //#
    #104167 RxD = 1'b1;
    #104167 RxD = 1'b1;
end
endtask
```

```
task tx_byte; //等到 tx_rdy 为有效的时候，将输入信号送 tx_data
input[7:0] b;
begin
    while (~tx_rdy)
        @ (posedge clk) ; //等待一个时钟周期再检测 tx_rdy 信号
    @ (posedge clk) ; //等待一个时钟周期
    #3 tx_vld = 1'b1; tx_data = b;
    @ (posedge clk) ;
    #3 tx_vld = 1'b0; tx_data = 8'b0;
end
endtask

//检测 RxD 端口
always @ (posedge clk)
    if (rx_vld)
        $display("--Byte %2h received @ %0d.", rx_data, $time); //显示当前时间
    else;

//检测 TxD 端口
integer i;
reg[7:0] rec_byte;
reg checkbit;
always @ (negedge TxD) //在 TxD 下降沿（起始的 0bit 出现时），开始一次检测过程
begin
    #52080 if (TxD != 1'b0) $display("--Start bit error @ %0d.", $time);
    for (i = 0; i < 8; i = i+1) #104167 rec_byte[i] = TxD;
    #104167 checkbit = TxD;
    #104167 if (TxD != 1'b1) $display("--End bit error @ %0d.", $time);
    #52080 $display("--Byte %2h transmitted @ %0d.", rec_byte, $time);
           if (checkbit != ^rec_byte) $display("--Check bit error @ %0d.", $time);
end

//检测过程
initial begin
```

```
    #100 rxtx_tb.rx_send(8'b01011010);
        rxtx_tb.tx_byte(8'b10100101);
    #2000000 $stop;
    end
endmodule
```

7.3 仿真输出 Hello World

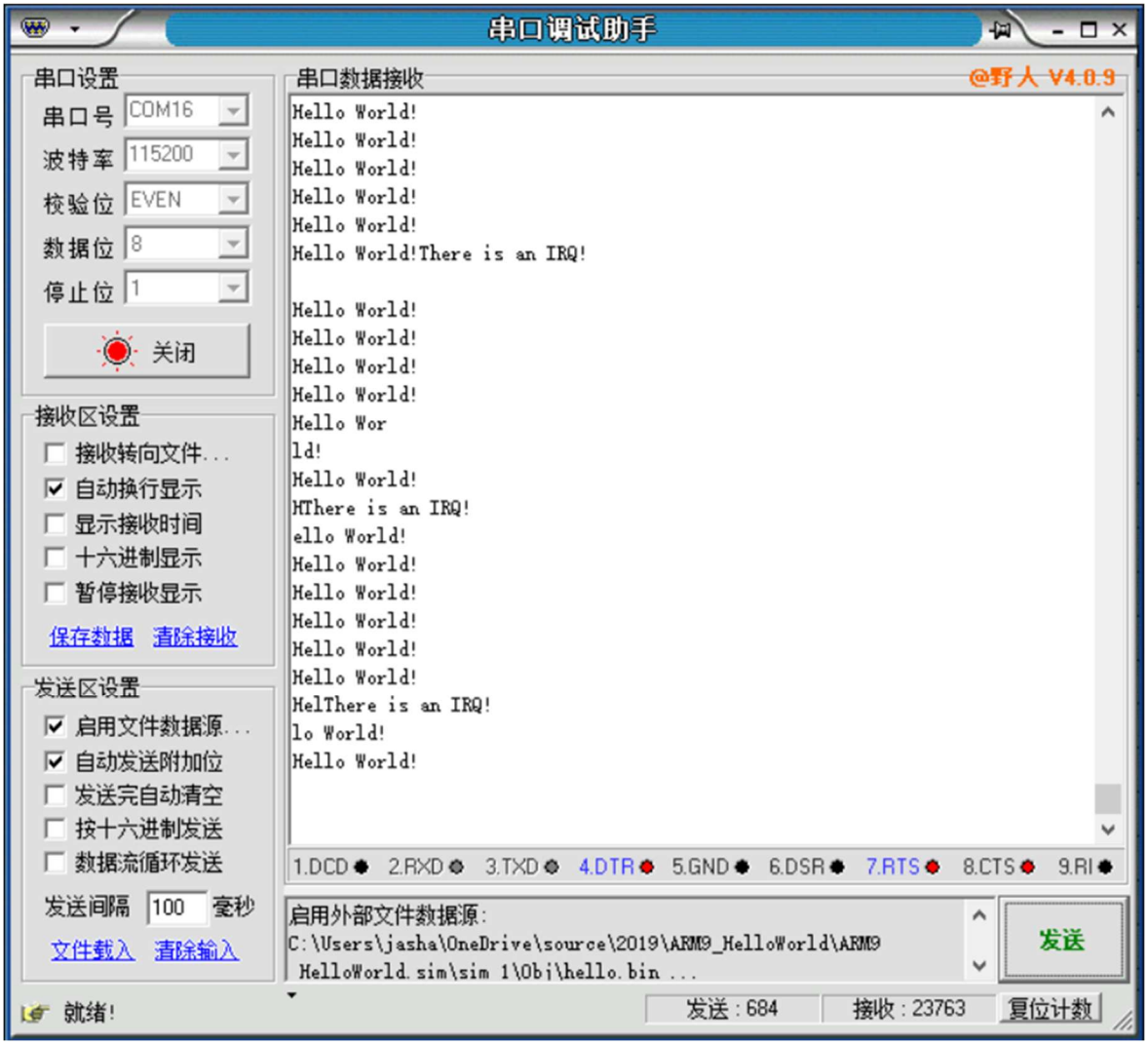
```
`timescale 1ns / 1ns
`define DEL 2

module tb();
    logic clk = 1'b0;
    always clk = #5 ~clk;

    logic rst = 1'b1;
    initial #10 rst = 1'b0;

    logic[7:0] rom[8191:0]; //大小为 8KB（按照 Keil 中的设定）的 ROM 块
    logic rom_en;
    logic[31:0] rom_addr;
    logic[31:0] rom_data;
    always_ff @ (posedge clk)
        if (rom_en) rom_data <= #`DEL {rom[rom_addr+3], rom[rom_addr+2], rom[rom_addr+1], rom[rom_addr]};
        else;

    integer fd, fx, i;
    initial begin
        for (i = 0; i < 8192; i = i+1) rom[i] = 0; //避免 ROM 中没有被 hello.bin 覆盖的区域为 X，供.coe 文件生成用
        fd = $fopen("../Obj/hello.bin", "rb");
        fx = $fread(rom, fd); //把.bin 文件的内容送入 ROM，作为指令池
        $fclose(fd);
        fd = $fopen("hello.coe", "w"); //把.bin 文件的内容按要求的格式写入另一个.coe 文件，供 Design Sources 中初始化 ROM IP 用
        $fdisplay(fd, "memory_initialization_radix = 16;");
```

```
        $fdisplay(fd, "memory_initialization_vector = ");
        for (i = 0; i < 8192; i = i+4)
            $fdisplay(fd, "%2h%2h%2h%2h%1s", rom[i+3], rom[i+2], rom[i+1], rom[i], i
== 8188 ? ";" : ",");
        $fclose(fd);
    end


    logic[31:0] ram[511:0];
    logic ram_cen, ram_wen;
    logic[3:0] ram_flag;
    logic[31:0] ram_addr;
    logic[31:0] ram_wdata;
    logic[31:0] ram_rdata;
    always_ff @ (posedge clk)
        if (ram_cen && ~ram_wen) //读数据池
            if (ram_addr == 32'he000_0000) ram_rdata <= #`DEL 32'h0; //如果是读寄存器，
则给出 SERIAL_FLAG 的值 0x0（因为现在还没有连接 UART 模块，而是通过仿真器输出数据，所以 SERIAL_FLAG
保持为 0，表示随时可以输出数据）
            else if (ram_addr[31:28] == 4'h0) ram_rdata <= #`DEL {rom[ram_addr+3],
rom[ram_addr+2], rom[ram_addr+1], rom[ram_addr]}; //根据地址段，判别是读 ROM 区域还是读 RAM
区域
            else if (ram_addr[31:28] == 4'h4) ram_rdata <= #`DEL ram[ram_addr[27:2]];
            else;
        else;
    always_ff @ (posedge clk)
        if (ram_cen && ram_wen && ram_addr[31:28] == 4'h4) //写 RAM
            ram[ram_addr[27:2]] <= #`DEL {
                (ram_flag[3] ? ram_wdata[31:24] : ram[ram_addr[27:2]][31:24]),
                (ram_flag[2] ? ram_wdata[23:16] : ram[ram_addr[27:2]][23:16]),
                (ram_flag[1] ? ram_wdata[15: 8] : ram[ram_addr[27:2]][15: 8]),
                (ram_flag[0] ? ram_wdata[ 7: 0] : ram[ram_addr[27:2]][ 7: 0])
            };
        else;
    always_ff @ (posedge clk)
        if (ram_cen && ram_wen && ram_addr == 32'he000_0004) $write("%s",
ram_wdata[7:0]); //写 SERIAL_OUT，也就是输出字符串
```

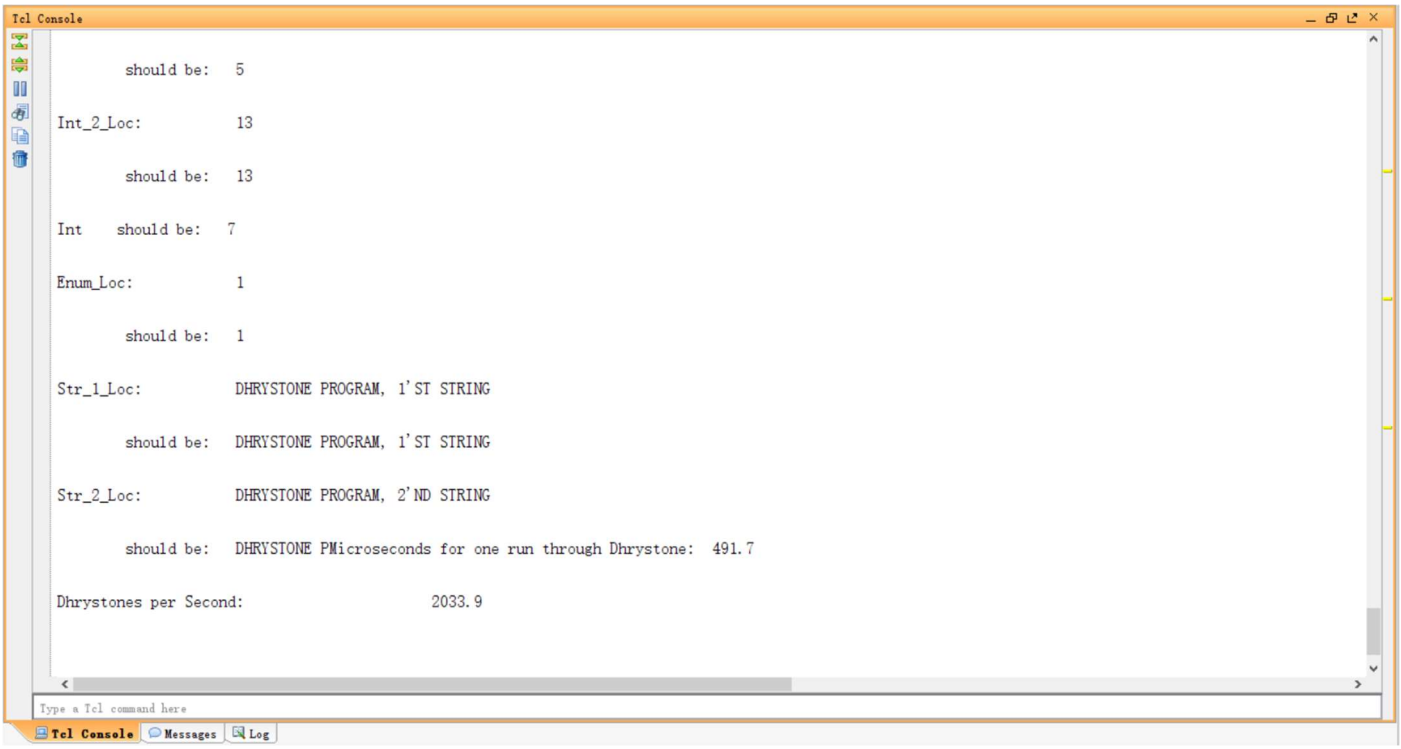
```
        else;

    logic irq = 1'b0;
    initial begin
        #100000 irq = 1'b1; //运行到100000ns时给出一周期irq脉冲
        #10 irq = 1'b0;
    end

    arm9_compatiable_code u_arm9(
        .clk        (clk),
        .cpu_en     (1'b1),
        .cpu_restart(1'b0),
        .fiq        (1'b0),
        .irq        (irq),
        .ram_abort  (1'b0),
        .ram_rdata  (ram_rdata),
        .rom_abort  (1'b0),
        .rom_data   (rom_data),
        .rst        (rst),

        .ram_addr   (ram_addr),
        .ram_cen    (ram_cen),
        .ram_flag   (ram_flag),
        .ram_wdata  (ram_wdata),
        .ram_wen    (ram_wen),
        .rom_addr   (rom_addr),
        .rom_en     (rom_en)
    );

endmodule
```

运行结果

7.4 建立 Hello World 的 FPGA 设计工程



8.3 仿真运行 Dhrystone Benchmark



8.5 Dhrystone Benchmark 在开发板中运行

9.4 SkyEye 硬件模拟平台

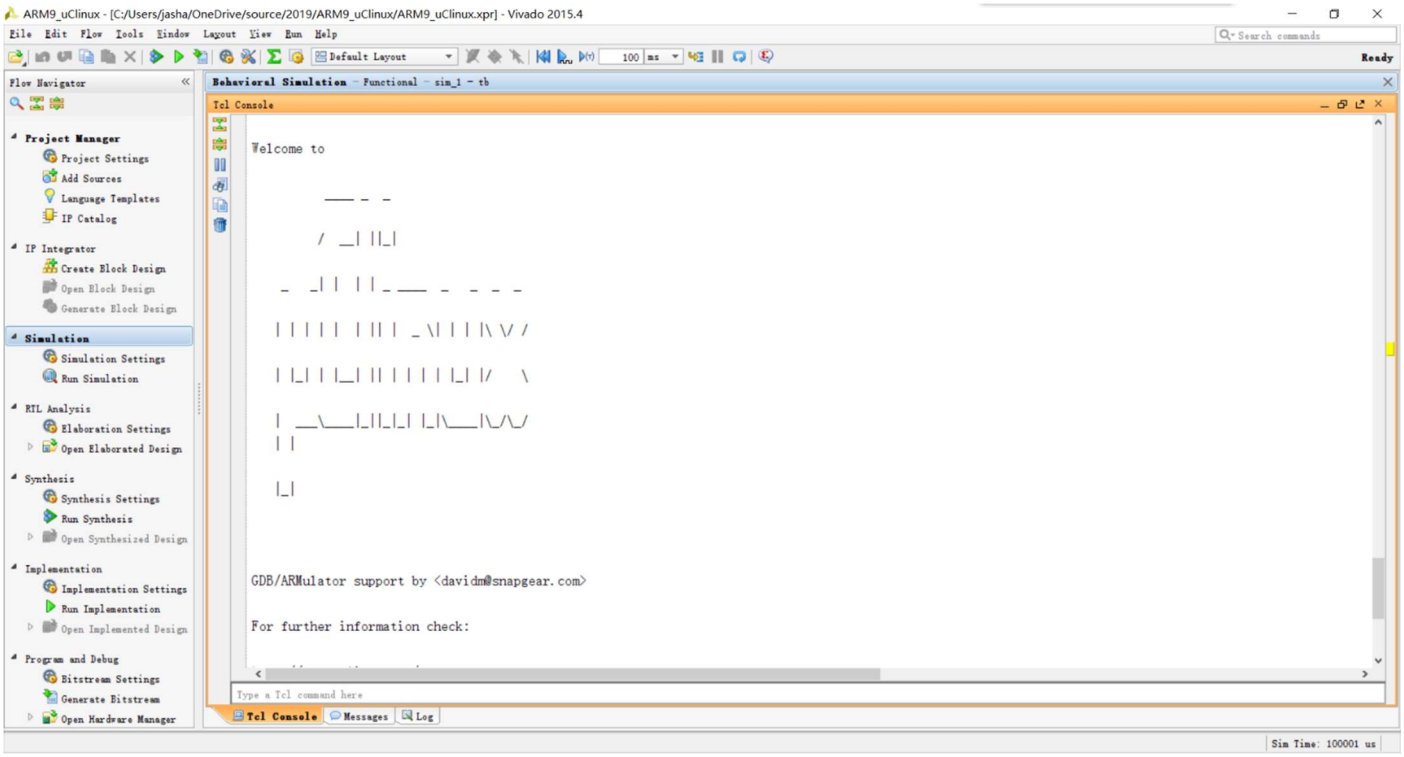

```
Your elf file is little endian.

arch: arm

cpu info: armv3, arm7tdmi, 41007700, fff8ff00, 0

mach info: name at91, mach_init addr 0x55bd387a3f70

uart_mod:0, desc_in:, desc_out:, converter:

SKYEYE: use arm7100 mmu ops

Loaded ROM   ./romfs.img

start addr is set to 0x01000000 by exec file.

Linux version 2.4.27-uc1 (skyeyeuser@debian) (gcc version 2.95.3 20010315 (release)(ColdFire patches - 20010318 from http://fiddes.net/coldfire/)(uClinux XIP and shared lib patches from http://www.snapgear.com/)) #3 Tue Aug 9 18:57:29 CST 2005

Processor: Atmel AT91M40xxx revision 0

Architecture: EB01

On node 0 totalpages: 1024

zone(0): 0 pages.

zone(1): 1024 pages.
```

```
zone(2): 0 pages.
Kernel command line: root=/dev/rom0
Calibrating delay loop... 15.82 BogoMIPS
Memory: 4MB = 4MB total
Memory: 2916KB available (903K code, 178K data, 40K init)
Dentry cache hash table entries: 512 (order: 0, 4096 bytes)
Inode cache hash table entries: 512 (order: 0, 4096 bytes)
Mount cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 1024 (order: 0, 4096 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Starting kswapd
Atmel USART driver version 0.99
ttyS0 at 0xfffd0000 (irq = 2) is a builtin Atmel APB USART
ttyS1 at 0xfffcc000 (irq = 3) is a builtin Atmel APB USART
Blkmem copyright 1998,1999 D. Jeff Dionne
Blkmem copyright 1998 Kenneth Albanowski
Blkmem 1 disk images:
0: 1400000-1512BFF [VIRTUAL 1400000-1512BFF] (RO)
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
Cirrus Logic CS8900A driver for Linux (V0.02)
 IO error in at91_io_write_halfword
IO erro in at91_io_read_halfword
 IO error in at91_io_write_halfword
 IO error in at91_io_write_halfword
 IO error in at91_io_write_halfword
IO erro in at91_io_read_halfword
 IO error in at91_io_write_halfword
IO erro in at91_io_read_halfword
 IO error in at91_io_write_halfword
IO erro in at91_io_read_halfword
eth0: incorrect signature 0x0000
```

```
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 512 bind 512)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
VFS: Mounted root (romfs filesystem) readonly.
Shell invoked to run file: /etc/rc
Command: hostname GDB-ARMulator
Command: /bin/expand /etc/ramfs.img /dev/ram0
Command: mount -t proc proc /proc
mount: /etc/mtab: Read-only file system
Command: mount -t ext2 /dev/ram0 /var
mount: /etc/mtab: Read-only file system
Command: mkdir /var/tmp
Command: mkdir /var/log
Command: mkdir /var/run
Command: mkdir /var/lock
Command: mkdir /var/empty
Command: cat /etc/motd
Welcome to

        ____ _  _
       /  __| ||_|
   _   _| |  | | _ ____  _   _  _  _
  | | | | |  | || |  _ \| | | |\ \/ /
  | |_| | |__| || | | | | |_| |/   \
  |  ___\____|_||_|_| |_|\____|\_/\_/
  | |
  |_|


GDB/ARMulator support by <davidm@snapgear.com>
For further information check:
http://www.uclinux.org/

Command: /bin/ifconfig eth0 up 10.0.0.2
SIOCGIFFLAGS: No such device
```

```
SIOCSIFADDR: No such device
pid 11: failed 512
Execution Finished, Exiting
init: Booting to single user mode


Sash command shell (version 1.1.1)
/>
```

9.5 Modelsim 下仿真 uClinux 启动过程



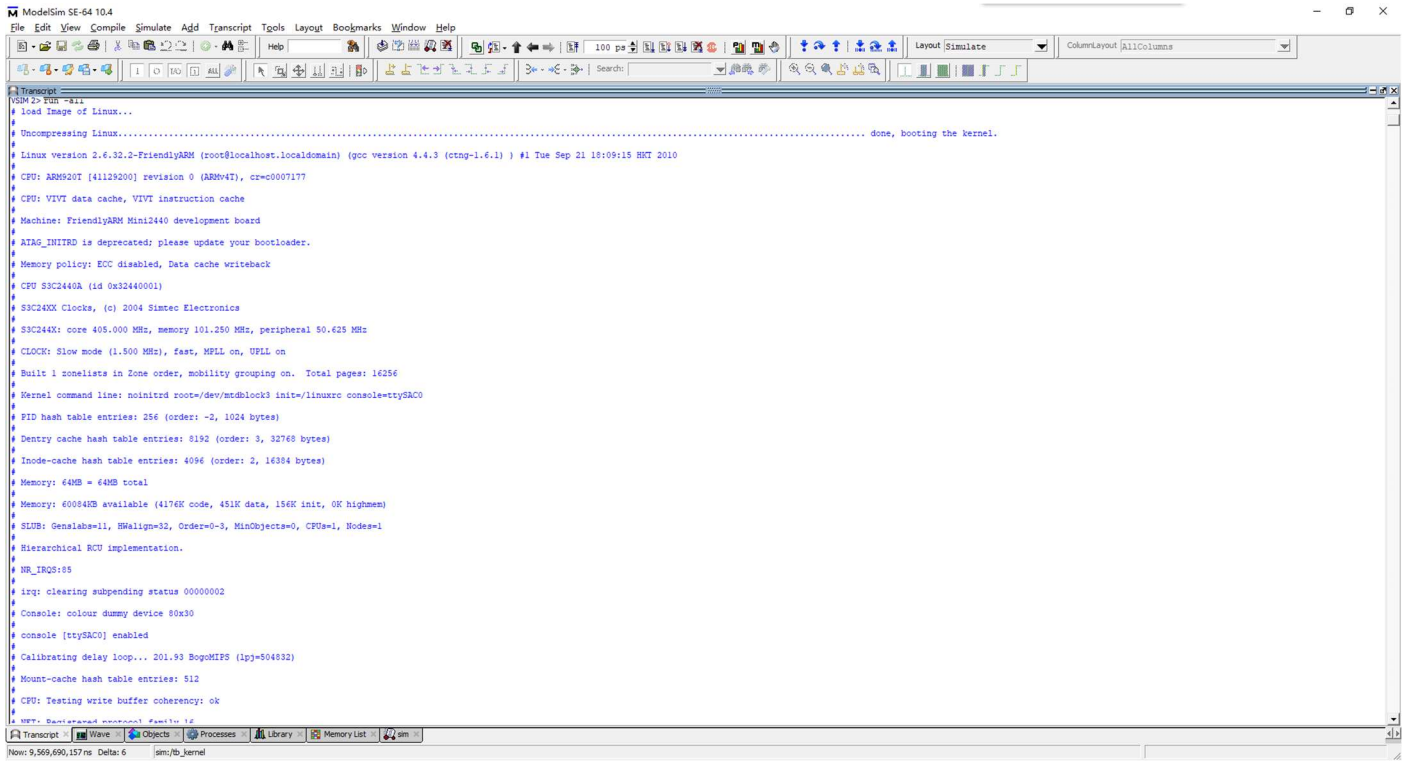

```
ux version 2.4.17-uc0 (root@hpclab.cs.tsinghua.edu.cn) (gcc version 2.95.3 20010315
(release)(ColdFi0318 from http://fiddes.net/coldfire/)(-msep-data patches)) #1 Sun
Feb 16 21:36:15 HKT 2003


T91M40xxx revision 0


Architecture: EB01


On node 0 totalpages: 1024


zone(0): 0 pages.


zone.
```

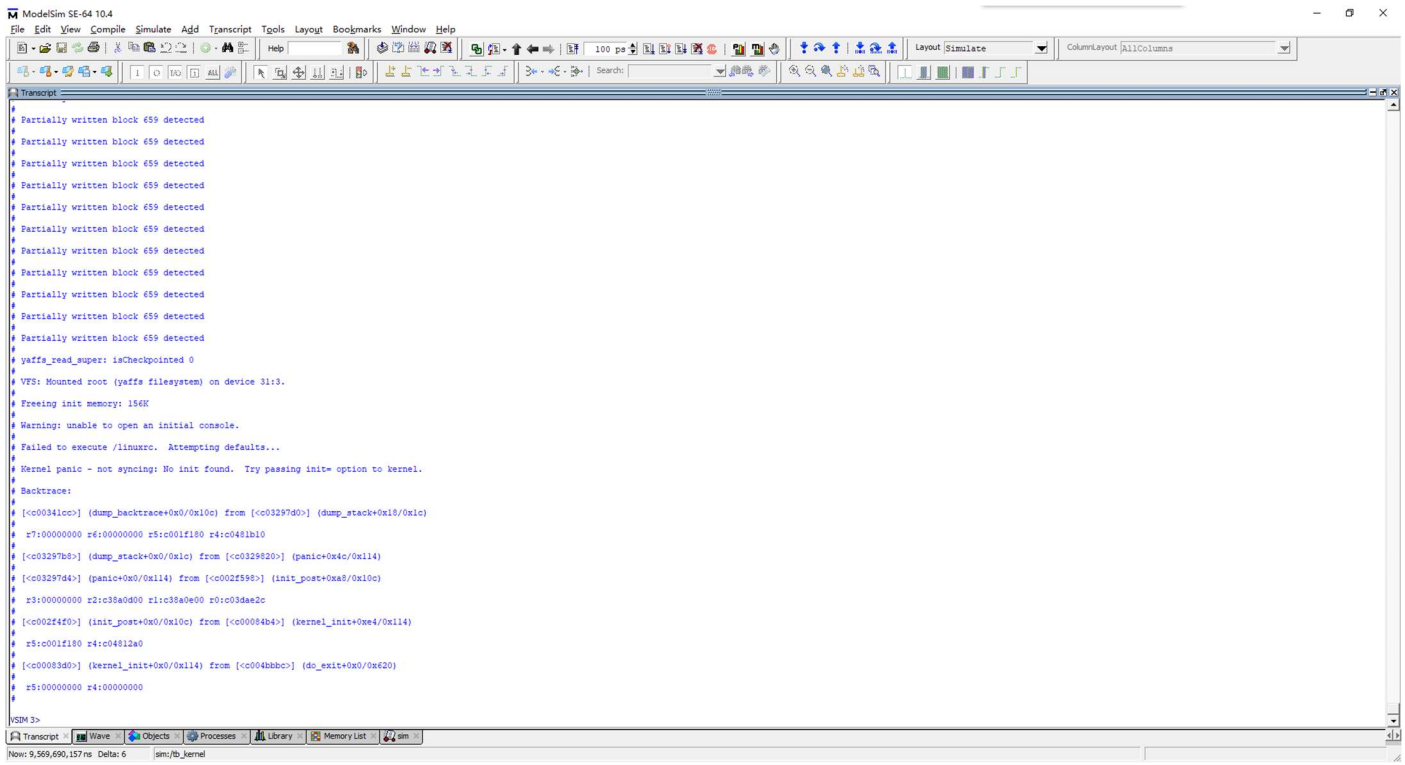
```
zone(2): 0 pages.

Kernel command line: root=/dev/rom0

Calibrating  13.00 BogoMIPS

Memory: 4MB = 4MB totMemory: 2992KB available (827K code, 163K data, 36K initDentry-
cache hash table entries: 512 (order: 0,)

Inode-cache hash table entries: 512 (or: 0, 4096 bytes)

Mount-cache hash table entries: 512 (order: 0, 6 bytes)

Be hash table entries: 1024 (order: 0, 4096 bytes)

s: 1024 (order: 0, 4096 bytes)

POSIX conformance testing by UNIFIX

Linux NET4.0 for Linux 2.4

Based upon Swansea Univeuter Society NET3.039

ink socket

ing kswapd

Atmel USART der version 0.99

ttyS0 at 0xfffd0000 (irq = 2)iltin Atmel APB USART

ttyS1 at 0xfffcc000 (irq = 3)a builtin Atmel APB USART

 slots per queue, batch=16
```

```
RAMDISK driver initialize 16 RAM disks of 4096K size 1024 blocksize

Bem copyright 1998,1999 D. Jeff Dionne

Blkmem copyright 1998 Kenneth Albanowski

Blkmem 1 disk iges:

00-145DBFF [VIRTUAL 1400000-145DBFF] (RO)

NET4: Linux TCP/IP 1.0 for NET4.0

P Protocols: ICMP, UDP, TCP

IP: routing cache hash table of 512 buckets, 4Kbyte
TCP: Hash tables configured (established 512 bind 512)

NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
 (romfs filesystem) readonly.


Command: hostname GDB-ARMulator

Command: /bin/expand /etc/ramfs.img /dev/ram0

Command: mount -t proc proc /proc

Command: mkdir /var/log

Command: mkdir /var/run


Welcome to
```

```
       ____ _  _

      /  __| ||_|

   _  _| |  | | _ ____  _  _  _  _

  | | | | |  | || |  _ \| | | |\ \/ /

  | |_| | |__| || | | | | |_| |/   \

  |  ___\____|_||_|_| |_|\____|\_/\_/
  | |

  |_|



GDB/ARMulator support by <davidm@snapgear.com>

For further information check:

http://www.uclinux.org/



Execution Finished, Exiting



Sash command shell (version 1.1.1)
```

10.6 仿真启动 Linux 操作系统






```
# load Image of Linux...
#
#                                                    Uncompressing
Linux.....................................................................
..................................................... done, booting
the kernel.
#
```

```
# Linux version 2.6.32.2-FriendlyARM (root@localhost.localdomain) (gcc version 4.4.3
(ctng-1.6.1) ) #1 Tue Sep 21 18:09:15 HKT 2010
#
# CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177
#
# CPU: VIVT data cache, VIVT instruction cache
#
# Machine: FriendlyARM Mini2440 development board
#
# ATAG_INITRD is deprecated; please update your bootloader.
#
# Memory policy: ECC disabled, Data cache writeback
#
# CPU S3C2440A (id 0x32440001)
#
# S3C24XX Clocks, (c) 2004 Simtec Electronics
#
# S3C244X: core 405.000 MHz, memory 101.250 MHz, peripheral 50.625 MHz
#
# CLOCK: Slow mode (1.500 MHz), fast, MPLL on, UPLL on
#
# Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 16256
#
# Kernel command line: noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0
#
# PID hash table entries: 256 (order: -2, 1024 bytes)
#
# Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
#
# Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
#
# Memory: 64MB = 64MB total
#
# Memory: 60084KB available (4176K code, 451K data, 156K init, 0K highmem)
#
```

```
# SLUB: Genslabs=11, HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
#
# Hierarchical RCU implementation.
#
# NR_IRQS:85
#
# irq: clearing subpending status 00000002
#
# Console: colour dummy device 80x30
#
# console [ttySAC0] enabled
#
# Calibrating delay loop... 201.93 BogoMIPS (lpj=504832)
#
# Mount-cache hash table entries: 512
#
# CPU: Testing write buffer coherency: ok
#
# NET: Registered protocol family 16
#
# S3C2440: Initialising architecture
#
# S3C2440: IRQ Support
#
# S3C24XX DMA Driver, (c) 2003-2004,2006 Simtec Electronics
#
# DMA channel 0 at c4808000, irq 33
#
# DMA channel 1 at c4808040, irq 34
#
# DMA channel 2 at c4808080, irq 35
#
# DMA channel 3 at c48080c0, irq 36
#
# S3C244X: Clock Support, DVS off
```

```
#
# bio: create slab <bio-0> at 0
#
# SCSI subsystem initialized
#
# usbcore: registered new interface driver usbfs
#
# usbcore: registered new interface driver hub
#
# usbcore: registered new device driver usb
#
# s3c-i2c s3c2440-i2c: slave address 0x10
#
# s3c-i2c s3c2440-i2c: bus frequency set to 98 KHz
#
# s3c-i2c s3c2440-i2c: i2c-0: S3C I2C adapter
#
# NET: Registered protocol family 2
#
# IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
#
# TCP established hash table entries: 2048 (order: 2, 16384 bytes)
#
# TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
#
# TCP: Hash tables configured (established 2048 bind 2048)
#
# TCP reno registered
#
# NET: Registered protocol family 1
#
# RPC: Registered udp transport module.
#
# RPC: Registered tcp transport module.
#
```

```
# RPC: Registered tcp NFSv4.1 backchannel transport module.
#
# NetWinder Floating Point Emulator V0.97 (double precision)
#
# 3267461685: ram mmu l1 fault at 00000000--rom_addr=c0066d18
# yaffs Sep 21 2010 18:08:36 Installing.
#
# msgmni has been set to 117
#
# alg: No test for stdrng (krng)
#
# io scheduler noop registered (default)
#
# Console: switching to colour frame buffer device 30x20
#
# fb0: s3c2410fb frame buffer device
#
# backlight         initialized
#
# leds              initialized
#
# buttons           initialized
#
# pwm               initialized
#
# adc               initialized
#
# s3c2440-uart.0: s3c2410_serial0 at MMIO 0x50000000 (irq = 70) is a S3C2440
#
# s3c2440-uart.1: s3c2410_serial1 at MMIO 0x50004000 (irq = 73) is a S3C2440
#
# s3c2440-uart.2: s3c2410_serial2 at MMIO 0x50008000 (irq = 76) is a S3C2440
#
# loop: module loaded
#
```

```
# S3C24XX NAND Driver, (c) 2004 Simtec Electronics
#
# s3c24xx-nand s3c2440-nand: Tacls=3, 29ns Twrph0=7 69ns, Twrph1=3 29ns
#
# s3c24xx-nand s3c2440-nand: NAND soft ECC
#
# NAND device: Manufacturer ID: 0xec, Chip ID: 0xda (Samsung NAND 256MiB 3,3V 8-bit)
#
# Scanning device for bad blocks
#
# Creating 5 MTD partitions on "NAND 256MiB 3,3V 8-bit":
#
# 0x000000000000-0x000000040000 : "supervivi"
#
# 0x000000040000-0x000000060000 : "param"
#
# 0x000000060000-0x000000560000 : "Kernel"
#
# 0x000000560000-0x000040560000 : "root"
#
# mtd: partition "root" extends beyond the end of device "NAND 256MiB 3,3V 8-bit" -
- size truncated to 0xfaa0000
#
# 0x000000000000-0x000040000000 : "nand"
#
# mtd: partition "nand" extends beyond the end of device "NAND 256MiB 3,3V 8-bit" -
- size truncated to 0x10000000
#
# dm9000 Ethernet Driver, V1.31
#
# eth0: dm9000e at c4814300,c4818304 IRQ 51 MAC: 08:90:90:90:90:90 (chip)
#
# ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
#
# s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
```

```
#
# s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number 1
#
# s3c2410-ohci s3c2410-ohci: irq 42, io mem 0x49000000
#
# s3c2410-ohci s3c2410-ohci: init err (00000000 0000)
#
# ohci_hcd: can't start s3c24xx
#
# s3c2410-ohci s3c2410-ohci: startup error -75
#
# s3c2410-ohci s3c2410-ohci: USB bus 1 deregistered
#
# s3c2410-ohci: probe of s3c2410-ohci failed with error -75
#
# Initializing USB Mass Storage driver...
#
# usbcore: registered new interface driver usb-storage
#
# USB Mass Storage support registered.
#
# usbcore: registered new interface driver ums-alauda
#
# usbcore: registered new interface driver ums-cypress
#
# usbcore: registered new interface driver ums-datafab
#
# usbcore: registered new interface driver ums-freecom
#
# usbcore: registered new interface driver ums-isd200
#
# usbcore: registered new interface driver ums-jumpshot
#
# usbcore: registered new interface driver ums-karma
#
```

```
# usbcore: registered new interface driver ums-onetouch
#
# usbcore: registered new interface driver ums-sddr09
#
# usbcore: registered new interface driver ums-sddr55
#
# usbcore: registered new interface driver ums-usbat
#
# usbcore: registered new interface driver usbserial
#
# USB Serial support registered for generic
#
# usbcore: registered new interface driver usbserial_generic
#
# usbserial: USB Serial Driver core
#
# USB Serial support registered for aircable
#
# usbcore: registered new interface driver aircable
#
# USB Serial support registered for ark3116
#
# usbcore: registered new interface driver ark3116
#
# USB Serial support registered for Belkin / Peracom / GoHubs USB Serial Adapter
#
# usbcore: registered new interface driver belkin
#
# belkin_sa: v1.2:USB Belkin Serial converter driver
#
# USB Serial support registered for ch341-uart
#
# usbcore: registered new interface driver ch341
#
# USB Serial support registered for cp210x
```

```
#
# usbcore: registered new interface driver cp210x
#
# cp210x: v0.09:Silicon Labs CP210x RS232 serial adaptor driver
#
# USB Serial support registered for Reiner SCT Cyberjack USB card reader
#
# usbcore: registered new interface driver cyberjack
#
# cyberjack: v1.01 Matthias Bruestle
#
# cyberjack: REINER SCT cyberJack pinpad/e-com USB Chipcard Reader Driver
#
# USB Serial support registered for DeLorme Earthmate USB
#
# USB Serial support registered for HID->COM RS232 Adapter
#
# USB Serial support registered for Nokia CA-42 V2 Adapter
#
# usbcore: registered new interface driver cypress
#
# cypress_m8: v1.09:Cypress USB to Serial Driver
#
# USB Serial support registered for Digi 2 port USB adapter
#
# USB Serial support registered for Digi 4 port USB adapter
#
# usbcore: registered new interface driver digi_acceleport
#
# digi_acceleport: v1.80.1.2:Digi AccelePort USB-2/USB-4 Serial Converter driver
#
# USB Serial support registered for Edgeport 2 port adapter
#
# USB Serial support registered for Edgeport 4 port adapter
#
```

```
# USB Serial support registered for Edgeport 8 port adapter
#
# USB Serial support registered for EPiC device
#
# usbcore: registered new interface driver io_edgeport
#
# io_edgeport: v2.7:Edgeport USB Serial Driver
#
# USB Serial support registered for Edgeport TI 1 port adapter
#
# USB Serial support registered for Edgeport TI 2 port adapter
#
# usbcore: registered new interface driver io_ti
#
# io_ti: v0.7mode043006:Edgeport USB Serial Driver
#
# USB Serial support registered for empeg
#
# usbcore: registered new interface driver empeg
#
# empeg: v1.2:USB Empeg Mark I/II Driver
#
# USB Serial support registered for FTDI USB Serial Device
#
# usbcore: registered new interface driver ftdi_sio
#
# ftdi_sio: v1.5.0:USB FTDI Serial Converters Driver
#
# USB Serial support registered for funsoft
#
# usbcore: registered new interface driver funsoft
#
# USB Serial support registered for Garmin GPS usb/tty
#
# usbcore: registered new interface driver garmin_gps
```

```
#
# garmin_gps: v0.33:garmin gps driver
#
# USB Serial support registered for hp4X
#
# usbcore: registered new interface driver hp4X
#
# hp4x: v1.00:HP4x (48/49) Generic Serial driver
#
# USB Serial support registered for PocketPC PDA
#
# usbcore: registered new interface driver ipaq
#
# ipaq: v0.5:USB PocketPC PDA driver
#
# USB Serial support registered for IPWireless converter
#
# usbcore: registered new interface driver ipwtty
#
# ipw: v0.3:IPWireless tty driver
#
# USB Serial support registered for IR Dongle
#
# usbcore: registered new interface driver ir-usb
#
# ir_usb: v0.4:USB IR Dongle driver
#
# USB Serial support registered for iuu_phoenix
#
# usbcore: registered new interface driver iuu_phoenix
#
# iuu_phoenix: v0.11:Infinity USB Unlimited Phoenix driver
#
# USB Serial support registered for Keyspan - (without firmware)
#
```

```
# USB Serial support registered for Keyspan 1 port adapter
#
# USB Serial support registered for Keyspan 2 port adapter
#
# USB Serial support registered for Keyspan 4 port adapter
#
# usbcore: registered new interface driver keyspan
#
# keyspan: v1.1.5:Keyspan USB to Serial Converter Driver
#
# USB Serial support registered for Keyspan PDA
#
# USB Serial support registered for Keyspan PDA - (prerenumeration)
#
# USB Serial support registered for Xircom / Entregra PGS - (prerenumeration)
#
# usbcore: registered new interface driver keyspan_pda
#
# keyspan_pda: v1.1:USB Keyspan PDA Converter driver
#
# USB Serial support registered for KL5KUSB105D / PalmConnect
#
# usbcore: registered new interface driver kl5kusb105d
#
# kl5kusb105: v0.3a:KLSI KL5KUSB105 chipset USB->Serial Converter driver
#
# USB Serial support registered for KOBIL USB smart card terminal
#
# usbcore: registered new interface driver kobil
#
# kobil_sct: 21/05/2004:KOBIL USB Smart Card Terminal Driver (experimental)
#
# USB Serial support registered for MCT U232
#
# usbcore: registered new interface driver mct_u232
```

```
#
# mct_u232: z2.1:Magic Control Technology USB-RS232 converter driver
#
# USB Serial support registered for Moschip 2 port adapter
#
# mos7720: 1.0.0.4F:Moschip USB Serial Driver
#
# usbcore: registered new interface driver moschip7720
#
# USB Serial support registered for Moschip 7840/7820 USB Serial Driver
#
# mos7840: 1.3.2:Moschip 7840/7820 USB Serial Driver
#
# usbcore: registered new interface driver mos7840
#
# USB Serial support registered for moto-modem
#
# usbcore: registered new interface driver moto-modem
#
# USB Serial support registered for navman
#
# usbcore: registered new interface driver navman
#
# USB Serial support registered for ZyXEL - omni.net lcd plus usb
#
# usbcore: registered new interface driver omninet
#
# omninet: v1.1:USB ZyXEL omni.net LCD PLUS Driver
#
# USB Serial support registered for opticon
#
# usbcore: registered new interface driver opticon
#
# USB Serial support registered for GSM modem (1-port)
#
```

```
# usbcore: registered new interface driver option
#
# option: v0.7.2:USB Driver for GSM modems
#
# USB Serial support registered for oti6858
#
# usbcore: registered new interface driver oti6858
#
# USB Serial support registered for pl2303
#
# usbcore: registered new interface driver pl2303
#
# pl2303: Prolific PL2303 USB to serial adaptor driver
#
# USB Serial support registered for Qualcomm USB modem
#
# usbcore: registered new interface driver qcserial
#
# safe_serial: v0.0b:USB Safe Encapsulated Serial
#
# USB Serial support registered for safe_serial
#
# usbcore: registered new interface driver safe_serial
#
# USB Serial support registered for siemens_mpi
#
# usbcore: registered new interface driver siemens_mpi
#
# Driver for Siemens USB/MPI adapter
#
# Version 0.1 09/26/2005 Thomas Hergenhahn@web.de http://libnodave.sf.net
#
# USB Serial support registered for Sierra USB modem
#
# usbcore: registered new interface driver sierra
```

```
#
# sierra: v.1.3.8:USB Driver for Sierra Wireless USB modems
#
# USB Serial support registered for SPCP8x5
#
# usbcore: registered new interface driver spcp8x5
#
# spcp8x5: v0.04:SPCP8x5 USB to serial adaptor driver
#
# USB Serial support registered for symbol
#
# usbcore: registered new interface driver symbol
#
# USB Serial support registered for TI USB 3410 1 port adapter
#
# USB Serial support registered for TI USB 5052 2 port adapter
#
# usbcore: registered new interface driver ti_usb_3410_5052
#
# ti_usb_3410_5052: v0.9:TI USB 3410/5052 Serial Driver
#
# USB Serial support registered for Handspring Visor / Palm OS
#
# USB Serial support registered for Sony Clie 3.5
#
# USB Serial support registered for Sony Clie 5.0
#
# usbcore: registered new interface driver visor
#
# visor: USB HandSpring Visor / Palm OS driver
#
# USB Serial support registered for Connect Tech - WhiteHEAT - (prerenumeration)
#
# USB Serial support registered for Connect Tech - WhiteHEAT
#
```

```
# usbcore: registered new interface driver whiteheat
#
# whiteheat: v2.0:USB ConnectTech WhiteHEAT driver
#
# mice: PS/2 mouse device common for all mice
#
# s3c2410 TouchScreen successfully loaded
#
# input: s3c2410 TouchScreen as /devices/virtual/input/input0
#
# S3C24XX RTC, (c) 2004,2006 Simtec Electronics
#
# s3c2410-rtc s3c2410-rtc: rtc disabled, re-enabling
#
# s3c2410-rtc s3c2410-rtc: rtc core: registered s3c as rtc0
#
# i2c /dev entries driver
#
# Linux video capture interface: v2.00
#
# gspca: main v2.7.0 registered
#
# usbcore: registered new interface driver conex
#
# conex: registered
#
# usbcore: registered new interface driver etoms
#
# etoms: registered
#
# usbcore: registered new interface driver finepix
#
# finepix: registered
#
# usbcore: registered new interface driver jeilinj
```

```
#
# jeilinj: registered
#
# usbcore: registered new interface driver mars
#
# mars: registered
#
# usbcore: registered new interface driver mr97310a
#
# mr97310a: registered
#
# usbcore: registered new interface driver ov519
#
# ov519: registered
#
# usbcore: registered new interface driver ov534
#
# ov534: registered
#
# usbcore: registered new interface driver pac207
#
# pac207: registered
#
# usbcore: registered new interface driver pac7311
#
# pac7311: registered
#
# usbcore: registered new interface driver sn9c20x
#
# sn9c20x: registered
#
# usbcore: registered new interface driver sonixb
#
# sonixb: registered
#
```

```
# usbcore: registered new interface driver sonixj
#
# sonixj: registered
#
# usbcore: registered new interface driver spca500
#
# spca500: registered
#
# usbcore: registered new interface driver spca501
#
# spca501: registered
#
# usbcore: registered new interface driver spca505
#
# spca505: registered
#
# usbcore: registered new interface driver spca506
#
# spca506: registered
#
# usbcore: registered new interface driver spca508
#
# spca508: registered
#
# usbcore: registered new interface driver spca561
#
# spca561: registered
#
# usbcore: registered new interface driver sq905
#
# sq905: registered
#
# usbcore: registered new interface driver sq905c
#
# sq905c: registered
```

```
#
# usbcore: registered new interface driver sunplus
#
# sunplus: registered
#
# usbcore: registered new interface driver stk014
#
# stk014: registered
#
# usbcore: registered new interface driver t613
#
# t613: registered
#
# usbcore: registered new interface driver tv8532
#
# tv8532: registered
#
# usbcore: registered new interface driver vc032x
#
# vc032x: registered
#
# usbcore: registered new interface driver zc3xx
#
# zc3xx: registered
#
# usbcore: registered new interface driver ALi m5602
#
# ALi m5602: registered
#
# usbcore: registered new interface driver STV06xx
#
# STV06xx: registered
#
# gspca_gl860: driver startup - version 0.9d10
#
```

```
# usbcore: registered new interface driver gspca_gl860
#
# gspca_gl860: driver registered
#
# usbcore: registered new interface driver uvcvideo
#
# USB Video Class driver (v0.1.0)
#
# initializing s3c2440 camera interface......
#
# s3c2440 camif init done
#
# Loading OV9650 driver.........
#
# SCCB address 0x60, manufacture ID 0x0000, expect 0x7FA2
#
# SCCB address 0x60, manufacture ID 0x0000, expect 0x7FA2
#
# No OV9650 found!!!
#
# S3C2410 Watchdog Timer, (c) 2004 Simtec Electronics
#
# s3c2410-wdt s3c2410-wdt: watchdog inactive, reset disabled, irq enabled
#
# s3c-sdi s3c2440-sdi: powered down.
#
# s3c-sdi s3c2440-sdi: mmc0 - using pio, sw SDIO IRQ
#
# usbcore: registered new interface driver usbhid
#
# usbhid: v2.6:USB HID core driver
#
# Advanced Linux Sound Architecture Driver Version 1.0.21.
#
# No device for DAI UDA134X
```

```
#
# No device for DAI s3c24xx-i2s
#
# S3C24XX_UDA134X SoC Audio driver
#
# UDA134X SoC Audio Codec
#
# asoc: UDA134X <-> s3c24xx-i2s mapping ok
#
# ALSA device list:
#
#   #0: S3C24XX_UDA134X (UDA134X)
#
# TCP cubic registered
#
# NET: Registered protocol family 17
#
# s3c2410-rtc s3c2410-rtc: hctosys: invalid date/time
#
# yaffs: dev is 32505859 name is "mtdblock3"
#
# yaffs: passed flags ""
#
# yaffs: Attempting MTD mount on 31.3, "mtdblock3"
#
# yaffs: auto selecting yaffs2
#
# Partially written block 1268 detected
#
```

.....（除去空行，此处共有 1108 行形如 `Partially written block *** detected` 的打印信息，限于篇幅，这里省略列出）

```
# Partially written block 659 detected
#
# yaffs_read_super: isCheckpointed 0
#
```

```
# VFS: Mounted root (yaffs filesystem) on device 31:3.
#
# Freeing init memory: 156K
#
# Warning: unable to open an initial console.
#
# Failed to execute /linuxrc.  Attempting defaults...
#
# Kernel panic - not syncing: No init found.  Try passing init= option to kernel.
#
# Backtrace:
#
# [<c00341cc>] (dump_backtrace+0x0/0x10c) from [<c03297d0>] (dump_stack+0x18/0x1c)
#
#  r7:00000000 r6:00000000 r5:c001f180 r4:c0481b10
#
# [<c03297b8>] (dump_stack+0x0/0x1c) from [<c0329820>] (panic+0x4c/0x114)
#
# [<c03297d4>] (panic+0x0/0x114) from [<c002f598>] (init_post+0xa8/0x10c)
#
#  r3:00000000 r2:c38a0d00 r1:c38a0e00 r0:c03dae2c
#
# [<c002f4f0>] (init_post+0x0/0x10c) from [<c00084b4>] (kernel_init+0xe4/0x114)
#
#  r5:c001f180 r4:c04812a0
#
# [<c00083d0>] (kernel_init+0x0/0x114) from [<c004bbbc>] (do_exit+0x0/0x620)
#
#  r5:00000000 r4:00000000
#
```

值得一提的是，最后出现了 panic。

经查，“Kernel panic - not syncing: No init found. Try passing init= option to kernel."很可能是文件系统导致的错误，而文件系统的映像是原书作者提供的，这超出了本课程的范围。因此不再继续探究。且原书最后还附了一句话：在仿真的最后阶段，可能由于某些寄存器的设定不到位，会有略微不同。此次仿真仅供参考。