# 珊瑚（aCoral）操作系统用户手册

## 第一章 简介

珊瑚（aCoral）是电子科技大学嵌入式实时计算研究团队开发的一款嵌入式实时操作系统，开源，具有高可配，高扩展性。珊瑚（aCoral）1.0版本目前提供对mini2440单核开发板的支持，后续将提供对多核开发板的支持。珊瑚内核从功能结构角度，由中断系统、内存管理系统、线程管理系统、线程交互系统组成。

中断系统对2440中的中断复用进行了区分，总共支持来自50个中断源的请求，包括内部定时器、DMA控制器、串口、RTC时钟、外部按键等，并支持中断嵌套。中断系统针对不同的使用场景，提供了三种不同的中断模式，比如针对对中断响应实时性要求更高的场景，就可以将中断类型设置为实时中断。关于如何管理中断，如何添加自己的中断服务函数，将在第三章进行阐述。

内存管理系统由两级组成，向下由伙伴系统对2440的64MB SDRAM进行管理，向上由资源池系统为其它模块提供内存资源的使用接口，当然，也可以越过资源池系统，直接调用伙伴系统的接口对内存进行使用。

线程管理系统支持对线程的创建、调度、杀死等操作，在目前的2.0版本中只支持固定优先级、可抢占式周期调度策略。珊瑚中线程优先级数越小，优先级越高，线程越优先被调度。最高优先级为0，最小优先级为100（支持POSIX线程下为130）。支持多个线程同时拥有相同的优先级，这些线程在就绪队列中按照先来后到进行调度。关于如何在珊瑚中创建一个周期线程，将在第三章中的3.3进行阐述。

内核之外还包括对一些外设的驱动支持、文件系统、GUI界面等模块。目前珊瑚支持的外设有串口、网口和LCD。对外设的使用以及如何扩展将在第四章中的4.1进行阐述。

珊瑚上电之后执行的第一行代码就位于**hal\s3c2440\src\start.s**中，标号**\_\_ENTRY**处。2440上电之后，CPU将从0地址处开始取指执行指令。而0地址处正是这句跳转指令。这句跳转程序将跳转到**ResetHandler**标号处，执行一些上电之后的硬件初始化工作，包括关闭看门狗、配置时钟、堆栈初始化、复制OS到SDRAM等。

代码最终将寄存器pc设置为**acoral\_start**的值，表示CPU将跳转到**acoral\_start**函数处执行。**acoral\_start**函数位于**kernel\include\core.c**中。

代码1-1 **acoral\_start函数**

|  |
| --- |
| void acoral\_start(){  orig\_thread.console\_id=ACORAL\_DEV\_ERR\_ID;  acoral\_set\_orig\_thread(&orig\_thread);  /\*内核模块初始化\*/  acoral\_module\_init();  /\*串口终端应该初始化好了，将根线程的终端id设置为串口终端\*/  #ifdef CFG\_DRIVER  orig\_thread.console\_id=acoral\_dev\_open("console");;  #endif  /\*主cpu开始函数\*/  acoral\_core\_cpu\_start();  } |

这部分代码中，最重要的就是**acoral\_module\_init()**这个函数。这个函数将初始化珊瑚（aCoral）的各个模块，包括中断系统、内存管理系统、线程管理系统、线程交互系统等嵌入式操作系统必需的模块。初始化完成后，执行**acoral\_core\_cpu\_start()**函数，至此，珊瑚（aCoral）开始运行。

## 第二章 快速上手

这一章将介绍如何快速将珊瑚操作系统烧写到mini2440开发板上运行。在烧写之前，要先对烧写以及后续开发调试的环境进行配置。

### 2.1 环境配置

#### 2.1.1 USB转串口驱动



图2-1 USB转串口安装

#### 2.1.2 虚拟机配置（可以直接安装Ubuntu系统，编译用）

1. 安装vmware workstation pro、安装ubuntu虚拟机、安装vmtool、启用共享文件夹，按下面这个链接做就行了，其中vmware的版本我用的是16，这个不重要，ubuntu版本最好是18.04.5

<https://zhuanlan.zhihu.com/p/38797088>

1. 配置交叉编译工具。可以参考

<https://www.jianshu.com/p/5a10c3237255>

先把arm-2010q1-188-arm-none-eabi-i686-pc-linux-gnu.tar.bz2放到虚拟机里解压了

**tar jxvf arm-2010q1-188-arm-none-eabi-i686-pc-linux-gnu.tar.bz2**

再编辑profile文件添加环境变量

**gedit /etc/profile**

在profile中最后一行添加：

**export PATH=$PATH: 你自己的路径/arm-2010q1/bin**

然后使用命令：

**source /etc/profile**

使环境变量生效（在不同用户下最好都运行一下此语句，例如shgao和root用户下都运行一下）。

检查是否将路径加入到PATH：

**echo $PATH**

显示的内容中有 你自己的路径/arm-2010q1/bin，说明已经将交叉编译器的路径加入PATH。

之后可以使用命令：

**arm-none-eabi-gcc -v**

查看环境是否搭建成功，如果输出以下内容，代表已经搭建成功：

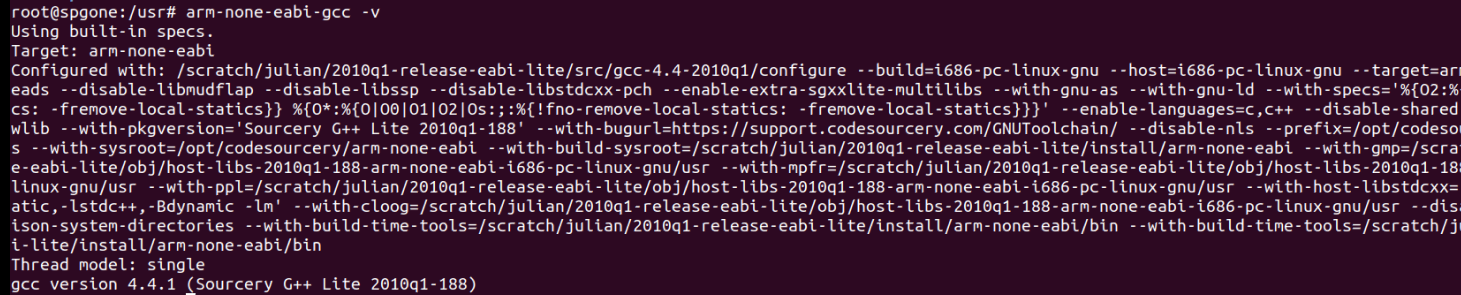


图2-2 检查编译环境

注意需要把顶层MAKEFILE中的CROSS\_COMPILE改为自己的路径

#### j-link套件（调试用）

1. 安装工具segger jlink 6.32。[SEGGER - The Embedded Experts - Downloads - J-Link / J-Trace](https://www.segger.com/downloads/jlink/)

2. 测试是否安装成功。先把jlink连上电脑，如图



图2-3 jlink连接

打开安装的j-Link Commander，下面这样就对了（版本应为6.32）（可能会提示更新固件，不理他先）

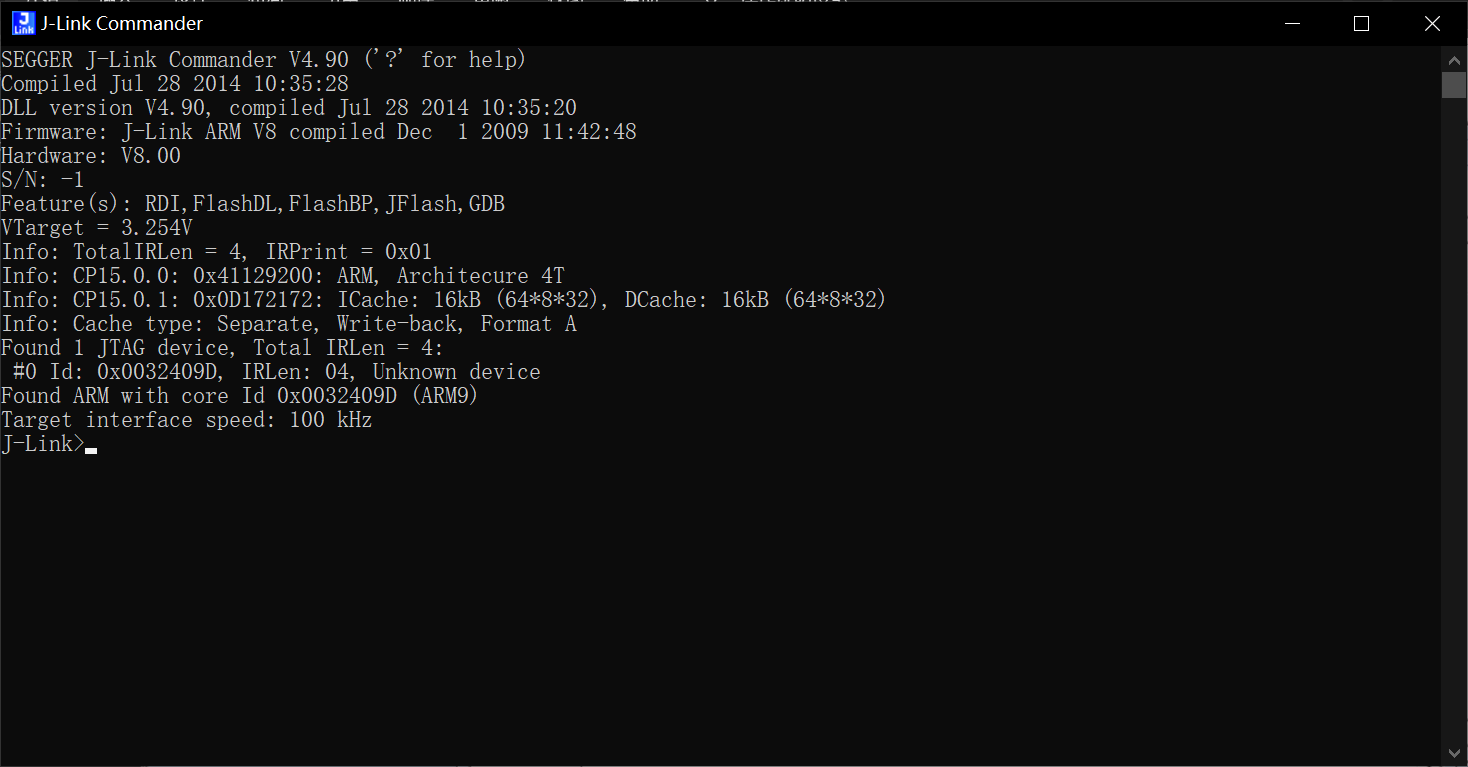


图2-4 检查jlink安装是否成功

#### 2.1.4 VSCode插件安装（可选，配合jlink调试用）

安装下面这个VSCode插件

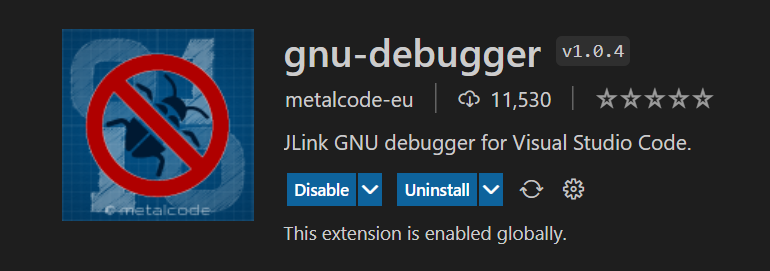


图2-5 VSC插件

按照说明进行配置。

### 2.2 第一次运行

打开安装的j-Flash，选择existing project：工具/mini2440.flash如下图，start j-flash

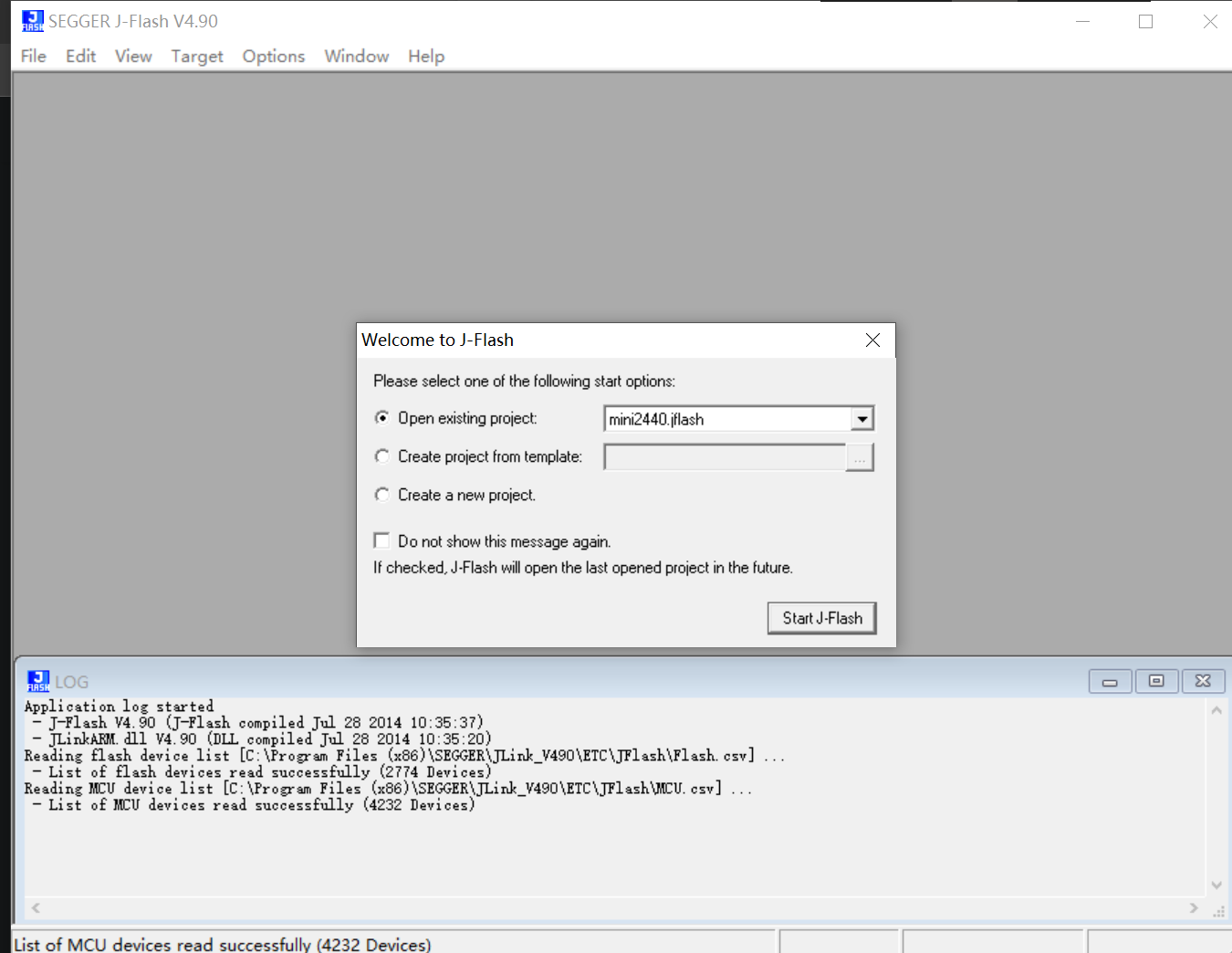


图2-6 jflash界面

file/open data file，选择acoral.bin

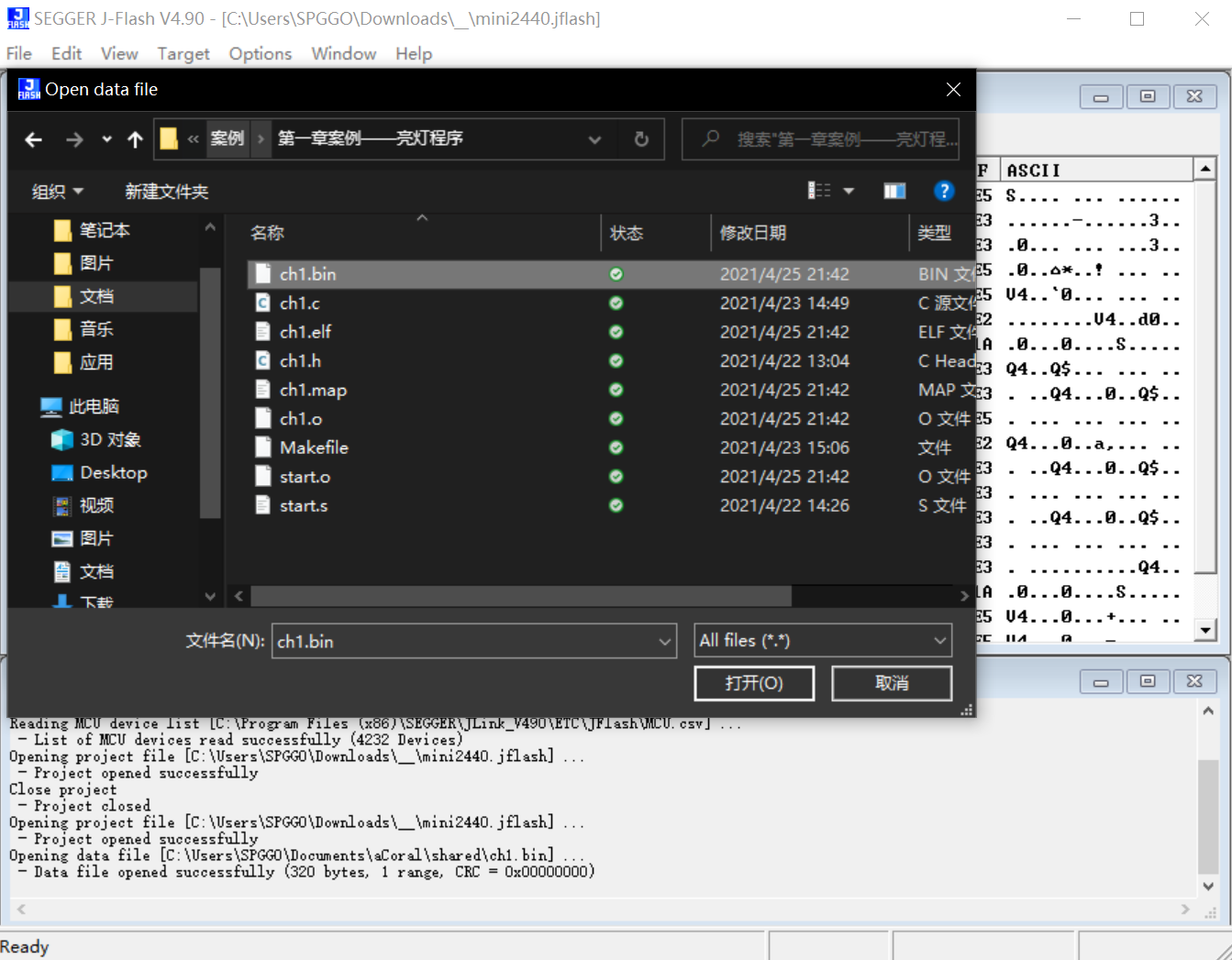


图2-7 选择文件

target/auto，这样就成功了

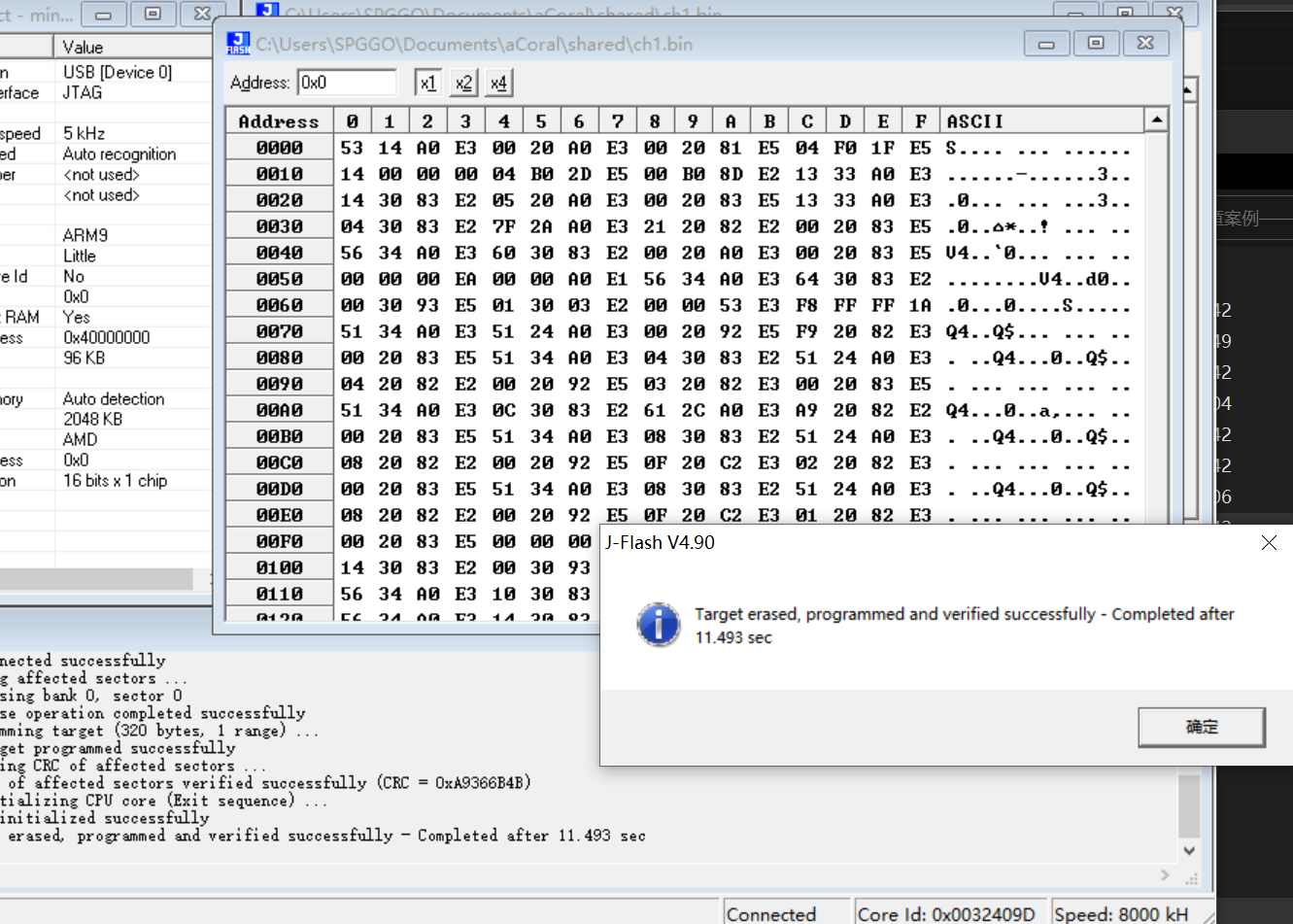


图2-8 烧写成功

烧写完成后，使用串口对操作系统进行观察，连线图如下：

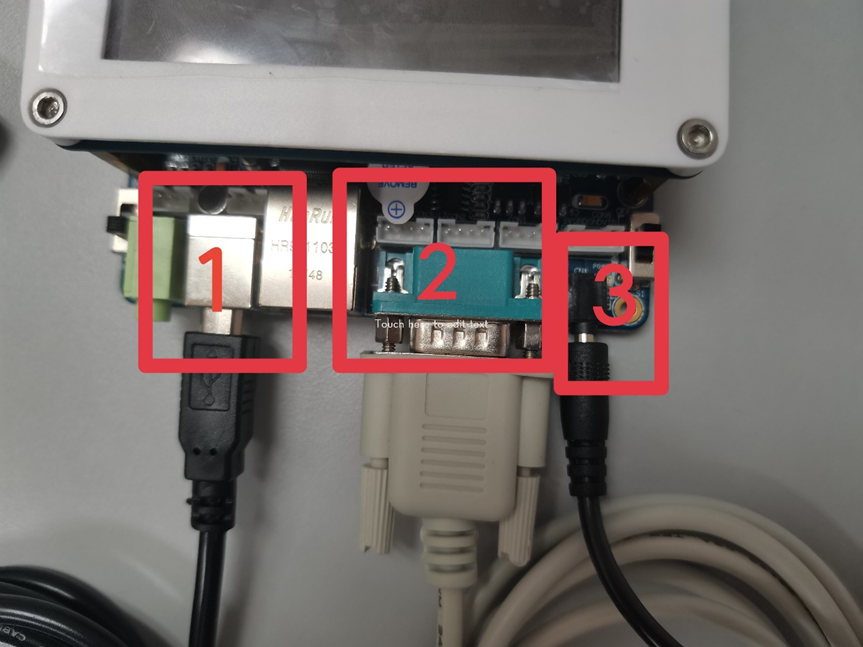


图2-9 串口连线

最后使用串口助手，将会看到”acoral:>”的字样，表示启动成功。

## 第三章 使用内核

### 3.1 目录结构

珊瑚的一级目录结构如图7.1所示

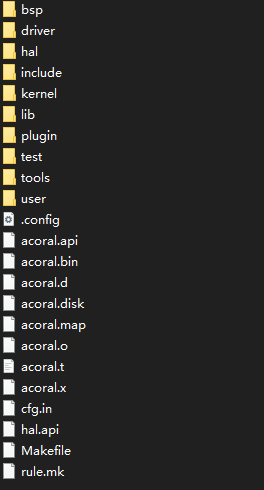


图3-1 目录结构

其中一些重要的文件或者目录的作用如下（重要的已进行标红处理）：

* **driver：**不同开发板的外设驱动程序，例如UART串口、网卡、LCD显示屏等
* **hal：**硬件抽象层（Hardware Abstraction Layer），将不同开发板的硬件资源访问封装成统一的接口，供上层操作系统使用，
* **include：**主要就是几个与系统相关的头文件定义，比如珊瑚（aCoral）中的各种数据类型定义（acoral\_u8等），处理器型号（S3C2440），还有珊瑚（aCoral）某个功能是否启用等。这个文件夹中的autocfg.h头文件是在make menuconfig之后自动生成的，与根目录的.config配置文件中的内容对应。
* **kernel：珊瑚（**aCoral）的内核层源码，与硬件无关
* **lib：**定义了上层操作系统中常用的一些数据结构，如list、queue、bitmap等
* **test：**操作系统启动之后，对系统的一些测试函数。因为是系统启动完成后自动执行的任务，所以所需要执行的任务就可以写在这个文件夹内。
* **.config：**内核编译配置，每次执行make命令，都会读取这个文件中的一些配置，包括使用的处理器型号等。每次执行make menuconfig命令调整系统配置时都会将保存的配置写入这个文件。
* **Makefile：**编译命令集合，需要注意的是，要想让新加入的文件被编译入内核，需要在同级的Makefile中，扩充obj-y变量，例如在driver/s3c2440/src中，要添加串口的驱动，就需要：

代码3-1 修改Makefile，添加文件

|  |
| --- |
| O\_TARGET := src.o  obj-y :=brd\_driver.o  obj-y+=con\_uart.o  include $(TOPDIR)/rule.mk |

### 3.2 API

用户在使用珊瑚进行开发的过程中，可以使用一些现有的API，简化开发流程。下面几张表按照不同的模块（子系统），罗列了珊瑚中的API。至于这些API的参数，用户可以自行在代码中进行检索查看。

#### 3.2.1 线程管理API

珊瑚操作系统中，线程管理API如表3-1：

表3-1 线程管理API

|  |  |  |
| --- | --- | --- |
| index | API | 描述 |
| 1 | acoral\_sched | 线程调度器 |
| 2 | acoral\_create\_thread | 线程创建（指定CPU、优先级、策略） |
| acoral\_create\_thread\_ext |
| 3 | acoral\_release\_thread | 指定线程释放空间、资源 |
| 4 | acoral\_suspend\_thread | 指定线程挂起，前者会启动调度器，后者不会 |
| acoral\_unrdy\_thread |
| 5 | acoral\_resume\_thread | 指定线程就绪，前者会启动调度器，后者不会 |
| acoral\_rdy\_thread |
| 6 | acoral\_delay\_self | 线程自我延时，加入延时等待队列 |
| 7 | acoral\_kill\_thread | 指定线程杀死 |
| 8 | acoral\_thread\_change\_prio | 改变线程优先级 |

#### 3.2.2 队列管理API

珊瑚操作系统中，队列管理API如表3-2：

表3-2 队列管理API

|  |  |  |
| --- | --- | --- |
| index | 队列名 | 描述 |
| 1 | acoral\_ready\_queues[] | 就绪队列（CPU私有） |
| 2 | time\_delay\_queue | 延时等待队列，acoral\_delay\_self加入 |
| 3 | timeout\_queue | 超时等待队列，比如等待共享资源超时后就会加入这个队列 |
|  | API | 描述 |
| 1 | acoral\_rdyqueue\_add | 向就绪队列中加入一个线程 |
| 2 | acoral\_rdyqueue\_del | 从就绪队列中删除指定线程 |
| 3 | timeout\_queue\_add | 将线程挂到超时等待队列上 |
| 4 | timeout\_queue\_del | 将线程从超时等待队列删除 |
| 5 | acoral\_delayqueue\_add | 将线程挂到延时等待队列上 |
| 6 | acoral\_list\_empty | 判断一个队列是否为空 |
| 7 | acoral\_list\_del | 删除指定队列中某个指定的节点 |
| 8 | acoral\_list\_add | 向指定队列加入一个节点（头插法） |
| 9 | acoral\_list\_add2\_tail | 向指定队列加入一个节点（尾插法） |
| 10 | list\_entry | 获得队列上某个节点所处的完整数据结构 |
| 11 | acoral\_list\_init | 队列初始化 |

#### 3.2.3 线程交互API

珊瑚操作系统中，线程交互API如表3-3：

表3-3 线程交互API

|  |  |  |
| --- | --- | --- |
| index | API | 描述 |
| 1 | acoral\_mutex\_create | 创建一个互斥量 |
| 2 | acoral\_mutex\_del | 删除一个互斥量 |
| 3 | acoral\_mutex\_pend | 给互斥量上锁（后者为非阻塞式） |
| acoral\_mutex\_trypend |
| 4 | acoral\_mutex\_post | 给互斥量解锁 |
| 5 | acoral\_sem\_create | 创建一个信号量 |
| 6 | acoral\_sem\_del | 删除一个信号量 |
| 7 | acoral\_sem\_pend | 申请一个信号量 |
| 8 | acoral\_sem\_post | 释放一个信号量 |
| 9 | acoral\_sem\_getnum | 得到当前信号量数目 |
| 10 | acoral\_mbox\_create | 创建一个邮箱 |
| 11 | acoral\_mbox\_del | 删除一个邮箱 |
| 12 | acoral\_mbox\_send | 发送邮件 |
| 13 | acoral\_mbox\_recv | 接收邮件（后者为非阻塞式） |
| acoral\_mbox\_tryrecv |

#### 3.2.4 中断管理API

珊瑚操作系统中，中断管理API如表3-4：

表3-4 中断管理API

|  |  |  |
| --- | --- | --- |
| index | API | 描述 |
| 1 | acoral\_intr\_enable | 使能某一个中断 |
| 2 | acoral\_intr\_disable | 除能某一个中断 |
| 3 | acoral\_intr\_attach | 绑定中断服务程序到内核中断号 |
| 4 | acoral\_intr\_detach | 解绑某个内核中断号的服务程序并改为默认服务程序 |
| 5 | acoral\_intr\_enable | 全局中断使能 |
| 6 | acoral\_intr\_disable | 全局中断除能 |
| 7 | acoral\_intr\_nesting | 获取当前CPU的中断嵌套数 |
| 8 | acoral\_intr\_nesting\_inc | 当前CPU的中断嵌套数+1 |
| 9 | acoral\_intr\_nesting\_dec | 当前CPU的中断嵌套数-1 |

#### 3.2.5 时钟管理API

珊瑚操作系统中，时钟管理API如表3-5：

表3-5 时钟管理API

|  |  |  |
| --- | --- | --- |
| index | API | 描述 |
| 1 | acoral\_get\_ticks | 获取系统tick数 |
| 2 | acoral\_set\_ticks | 设置系统tick数 |

#### 3.2.6 内存管理API

珊瑚操作系统中，内存管理API如表3-6：

表3-6 内存管理API

|  |  |  |
| --- | --- | --- |
| index | API | 描述 |
| 1 | acoral\_malloc | 从伙伴系统获得满足大小的内存空间 |
| 2 | acoral\_free | 归还给伙伴系统 |
| 3 | acoral\_malloc\_size |  |
| 4 | acoral\_mem\_scan |  |
| 5 | acoral\_malloc2 |  |
| 6 | acoral\_free2 |  |
| 7 | acoral\_mem\_scan2 |  |
| 8 | acoral\_get\_res | 根据资源池控制块获取某一种资源 |
| 9 | acoral\_create\_pool | 根据资源池控制块创建新的资源池 |
| 10 | acoral\_get\_free\_pool | 获取一块空闲资源池 |
| 11 | acoral\_get\_pool\_by\_id | 获取指定资源池 |
| 12 | acoral\_collect\_pool | 回收某一资源内存池空闲的内存池 |
| 13 | acoral\_release\_res | 释放某一资源 |
| 14 | acoral\_get\_res\_by\_id | 获取指定资源 |

### 3.3 任务创建

在珊瑚内创建线程有三种方式，第一种是创建test任务，第二种是创建shell任务，第三种是创建中断任务。test任务将在系统启动后自动执行，shell任务可以在系统运行过程中通过shell动态创建，中断任务在中断发生后由中断服务程序创建。下面将分别介绍如何创建每一种线程任务。

#### 3.3.1 test任务

下面以led1为例，我们将在系统初始化阶段创建一个周期为1220的线程，反复改变led1的点亮/熄灭状态。在test/src文件夹内新建test\_period.c，内容如下：

代码3-2创建周期test任务示例

|  |
| --- |
| #include "acoral.h"  #include"../include/test.h"  static void test(int args){  rGPBDAT ^= 0x20;           //取反led1的状态  }  void period\_init()  {  rGPBCON = 0x400;   //led1作为输出io  acoral\_period\_policy\_data\_t data;  data.cpu=0;  data.prio=25;  data.prio\_type=ACORAL\_BASE\_PRIO;  data.time=1220;  acoral\_create\_thread\_ext(test,ACORAL\_PRINT\_STACK\_SIZE,1,spg,NULL,ACORAL\_SCHED\_POLICY\_PERIOD,&data);  }  TEST\_CALL(period\_init) |

调用acoral\_create\_thread\_ext接口和acoral\_period\_policy\_data\_t数据结构，创建了一个名为spg的周期任务，执行函数为test，将以1220毫秒为周期反复点亮mini2440开发板上的led1。

通过宏TEST\_CALL(period\_init)将这一任务放入test段。test段内的任务将在系统初始化阶段被执行。

#### 3.3.2 shell任务

在系统启动完成后，shell线程将通过串口接收来自用户的指令。可以通过创建新的指令来从shell创建任务。在user/src文件夹下的cmd.c文件内添加新的指令示例如下：

代码3-2 创建shell任务示例

|  |
| --- |
| static void test(int args){  rGPBDAT ^= 0x20;           //取反led1的状态  }  void spg ()  {  rGPBCON = 0x400;   //led1作为输出io  acoral\_period\_policy\_data\_t data;  data.cpu=0;  data.prio=25;  data.prio\_type=ACORAL\_BASE\_PRIO;  data.time=1220;  acoral\_create\_thread\_ext(test,ACORAL\_PRINT\_STACK\_SIZE,1,spg,NULL,ACORAL\_SCHED\_POLICY\_PERIOD,&data);  }  acoral\_shell\_cmd\_t spg\_cmd={      "spg",      spg,      "Easter egg",      NULL  }; |

再在cmd\_init函数内加入：

代码3-3 添加shell命令

|  |
| --- |
| add\_command(&spg\_cmd); |

就添加了一条新的指令。在shell中输入”spg”，即可在系统中创建一个名为spg的周期任务。

#### 3.3.3 中断任务

也可以在中断服务函数内调用线程创建接口来创建任务。在driver\s3c2440\include\reg.h中定义了所有2440的中断号，可以在任何阶段调用acoral\_intr\_attach接口来绑定中断处理函数。

## 第四章 设备和驱动

### 4.1 添加设备驱动

设备驱动位于driver/s3c2440/src目录下。通过定义acoral\_dev\_ops\_t结构体，并填充其write、read等指针来完成驱动的编写，以串口驱动为例：

代码4-1 添加串口驱动示例

|  |
| --- |
| #include<driver.h>  #include<console.h>  #include"../include/reg.h"  #define put\_char(ch) uart\_write(ch)  #define get\_char(ch) uart\_read(ch)  acoral\_dev\_ops\_t console\_ops;  void uart\_init();  void uart\_write(acoral\_u8 ch){      while(!(rUTRSTAT0 & 0x4));      WrUTXH0(ch);  }  void uart\_read(acoral\_char \*ch){      while(!(rUTRSTAT0 & 0x1));      \*ch=RdURXH0();  }  acoral\_32 console\_write(void \*data,acoral\_u32 size,acoral\_time tm\_out){      acoral\_u32 i;      char p;      for(i=0;i<size;i++){          p=\*((acoral\_char \*)data+i);          if(p=='\n')              put\_char('\r');          put\_char(p);      }  }  acoral\_32 console\_read(void \*data,acoral\_size size,acoral\_size offset,acoral\_time tm\_out){      acoral\_u32 i;      acoral\_char \*p;      for(i=0;i<size;i++){          p=(acoral\_char \*)data+i;          get\_char(p);      }      return size;  }  acoral\_32 console\_config(acoral\_u32 cmd,void \*data,acoral\_time time\_out){      switch(cmd){          case CONSOLE\_ECHO:              \*(acoral\_u8 \*)data=1;          default:              break;      }  }  acoral\_32 console\_open(){      uart\_init();      return 0;  }  void uart\_init(){      rUCON0=0x245;      rULCON0=0x3;      rUFCON0=0x0;      rUMCON0=0x0;      rUBRDIV0=(PCLK /(CFG\_BAUD\_RATE \* 16))-1;      rGPHCON = 0x0016faaa;      rGPHUP  = 0x7ff;    /\* The pull up function is disabled GPH[10:0]\*/  }  void con\_uart\_init(){      console\_ops.open=console\_open;      console\_ops.write=console\_write;      console\_ops.read=console\_read;      console\_ops.config=console\_config;      acoral\_drv\_register(&console\_ops,"console");  } |

再将串口初始化函数con\_uart\_init()加入brd\_driver.c，即可添加串口驱动到内核中。其他的外设驱动，添加方式也类似。总结下来，就是将acoral\_dev\_ops\_t结构体中的设备读、写、打开指针等和对应函数进行绑定的过程。