ARM微处理器启动和调试浅析

刘　强

中国电子科技集团公司第二十七研究所　郑州　450015

摘　要：简要介绍了ARM7体系结构及其特点，对ARM7微处理器的启动过程以及调试过程进行了分析。同时结合AT91M40800 ARM处理器的结构，叙述了其启动代码的组成和调试过程的内容。

关键词：ARM　重映射　启动代码　中断　JTAG

中图分类号：TP273.5　　**文献标识码**：A

1　前言

ARM微处理器广泛应用于工业控制、仪器仪表、汽车电子、通信、家用消费等嵌入式设备，由于传统的8/16位处理器速度不够快以及内存不够大，在一些工程设计中已经不能满足需求了，而32位的ARM微处理器以其完整的体系结构发展系列，小体积、低功耗、低成本、高性能，及时根据嵌入的对象的不同进行功能上的扩展的优势，在众多种类的嵌入式微处理器中脱颖而出，深受设计者们的欢迎。设计者利用ARM处理器的出色性能可得到完全满足其确切要求的设计方案，而且，借助于第三方开发者广泛的支持，设计者可以使用丰富的标准开发工具即快速又简单的来完成自己的设计开发工作。ARM处理器目前有很多系列，广泛使用的是ARM7和ARM9系列，本文主要对针对ARM7系列的启动与调试过程进行讨论。

2　结构特点

ARM7DTMI为低功耗32位RISC(精简指令计算机)型处理器，具有：

● 嵌入式ICE－RT逻辑：ICE－RT逻辑是一种基于JTAG的ARM的内核调试通道，通过ICE－RT逻辑设计者可以实现在线调试以及编程功能，例如条件断点，单步运行。因为这些设备都在片上，ICE－RT技术避免使用笨重的、不可靠的探针接插设备。嵌入在芯片中的调试模块与外部的系统时序是独立的。

● 非常低的功耗：ARM7DTMI采用ARMV4T结构，分为三级流水，大小统一的指令与数据Cache，平均功耗为0.6mW/MHz，在对功耗比较敏感的手持设备的开发应用中，低的功耗将大大增加电池使用时间。

● 最大4G的寻址能力：在嵌入式应用中，寻址大小的提高意味着系统可以有更大的存储空间和外扩设备能力的提高。

● 0.9MIPS/MHz的处理能力：若系统时钟速度为66MHz，则每条指令平均执行1.9个时钟周期，指令执行速度是非常快的。

● 支持32位ARM指令集和16位Thumb指令集：32位ARM指令集工作在ARM状态，每条指令具有多种操作功能，具有很高的使用效率；Thumb指令集工作在Thumb状态下，是常用的32位ARM指令集的子集，压缩成16位宽的操作码，这意味着程序存储器可以比使用32位指令集占用的空间更小，成本更低。在程序执行时。16位指令解压成全32位的ARM指令集，设计者可根据实际需求，在子程序里使用16位Thumb指令集，减少代码大小。

● 增强型乘法器(multiplier)：与以前处理器相比性能更高，可产生全64位的计算结果。

ARM7处理器支持用户、FIQ、IRQ、管理、终止、系统、未定义等7种处理器模式，除用户模式以外的其它模式都叫做特权模式，除用户和系统以外的其它5种模式叫做异常模式。大部分应用程序都在用户模式下运行，当处理器处于用户模式下时，执行的程序无法访问一些被保护的系统资源，以利于操作系统控制系统资源的使用；也不能改变模式，否则就会导致一次异常。

对于系统模式，任何异常都不会导致进入这一模式，而且它使用的寄存器和用户模式下基本相同，主要是用于有访问系统资源请求而又避免使用额外的寄存器的操作系统任务。在特权模式下，它们可以完全访问系统资源，可以自由地改变模式，在处理特定的异常时，系统进入对应的异常模式下。这 5 种异常模式都有各自额外的寄存器，用于避免在发生异常的时候与用户模式下的程序发生冲突[1][2]。

3 启动代码

由于对程序模块化简单化的要求，大部分的 ARM 程序都是用 C/C++语言编写的。然而，在 ARM 系统上电复位运行后，要设置中断、初始化各个模式的堆栈、设置系统时钟频率，还涉及到存储器的地址重映射以及重映射后中断地址的转移、初始化主程序用到的存储区等过程，而这些过程都是针对 ARM 内部寄存器结构的操作，用 C 语言编程是无法实现的。因此在转到 C/C++程序之前，需要用 ARM 的汇编语言编写启动程序，当完成上述过程后再从启动程序跳转到 C 程序运行。在 ARM 设计开发中，启动代码的编写是一个极重要的过程，由于 ARM 处理器的启动过程相对其它处理器较为复杂，如 51 单片机，导致许多初学者对 ARM 处理器的启动代码的编写感到困惑。下面就结合 ATMEL 公司的 AT91M40800 的启动代码编写过程进行分析比较。

3.1 REMAP(重映射)

在进行具体代码编写前有必要弄清楚 REMAP 的意义与作用，我们都清楚任何一个处理器在复位后都从地址 0 读取第一条指令并执行，也就是说在上电后必须有一个只读存储器位于地址 0 上面，这样才能保证处理器有正确可用的指令。但是与 RAM 相比 ROM 有总线窄(8 或 16 位)、访问时间长、不能随心更改内容的缺点，尤其是在中断处理过程中，这些缺点既增加了中断处理所需时间，同时也不能在运行中动态的改变中断向量表。为了解决这些问题，当 ARM7 初始化时把对实时性要求较高的中断向量程序复制到 RAM 中，当 ARM7 初始化结束后，就把 RAM 和 ROM 的地址进行切换，把 ROM 放到其它自定义的地址上，使 RAM 位于地址 0 上，当中断发生后处理器从 RAM 中读取程序从而在执行时提高处理速度。这些不同类型存储器所处地址的切换就是 REMAP。

不同厂家的 ARM 片子实现 REMAP 的方式虽然不同结果却相同，都是把 RAM 映射到地址 0 上。AT91M40800 是通过向 EBI Remap Control Register(外部总线接口重映射控制寄存器)的第 0 位写入 1 来实现 REMAP 操作的，该寄存器的地址是 0xFFE00020，代码形式如下。AT91M40800 内部有 8K 的 RAM，首地址位于地址 0x00300000 上，当执行完 REMAP 后，其首地址就位于地址 0x0 上了。而 ROM 的地址根据其大小，通过 EBI Chip Select Register(外部总线接口片选寄存器)0 可以在 0x00400000－0xFFBFFFFF 之间设置[3]。

MOV R0，#0x00000001 ；将常数 1 存到 R0 里

MOV R1，#0xFFE00020 ；将重映射控制寄存器的地址存到 R1 中

STR R0，[R1] ；向重映射控制寄存器的 0 位写入 1，实现 REMAP

实际具体应用时还要考虑 REMAP 后 ROM 的地址，一般是在 REMAP 前保存 Remap 以后的跳转地址，当相关寄存器设置好后，在把该地址重新赋予 PC 从而使程序跳到正确的位置上去，因此 AT91M40800 实际应用的 REMAP 代码如下，存储器在 REMAP 前后地址变化情况见图 1。

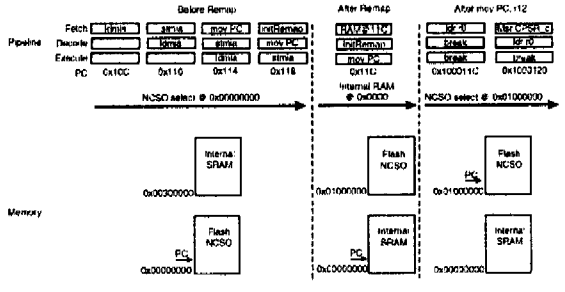

图 1 REMAP 前后存储器地址变化示意图

LDR R12，AfterRemapAdr ；保存实际跳转地址

ADR R11，EBI_Table

LDMIA R11，{R0－R10} ；把 EBI_Table 定义的参数装到 R0－R10

STMIA R10，{R0－R9} ；R0－R9 的参数写入到 从 EBI_BASE 开始的 10 个寄存器里，设

置各片选地址范围、数据宽度，同时执行 REMAP。

```
MOV PC, R12 ;恢复实际跳转地址
EBI_Table DCD EBI_CSR0_Val ;定义外部总线接口配置参数
DCD EBI_CSR1_Val
DCD EBI_CSR2_Val
DCD EBI_CSR3_Val
DCD EBI_CSR4_Val
DCD EBI_CSR5_Val
DCD EBI_CSR6_Val
DCD EBI_CSR7_Val
DCD 0x00000001 ;执行 REMAP 的参数
DCD EBI_MCR_Val
DCD EBI_BASE ;定义外部总线接口寄存器的基地址
AfterRemapAdr DCD AfterRemap ;定义实际跳转地址
AfterRemap
```

如果具体应用中对实时性要求不高或系统应用没有另加外围设备的话，完全可以不使用 REMAP，这样使用起来更加简单。但对于 AT91M40800 如果需要外扩外围设备的话，就得必须执行 REMAP，否则无法对外扩的设备进行选择操作。

表 1 ARM 中断向量表

中断类型	ARM 模式	优先级别	正常地址	高向量地址
复位	管理	1(最高)	0x00000000	0xFFFF0000
未定义指令	未定义	6	0x00000004	0xFFFF0004
软件中断 SWI	管理	6	0x00000008	0xFFFF0008
读取指令终止	终止	5	0x0000000C	0xFFFF000C
访问数据终止	终止	2	0x00000010	0xFFFF0010
IRQ 中断	IRQ	4	0x00000018	0xFFFF0018
FIQ 快速中断	FIQ	3	0x0000001C	0xFFFF001C

3.2 中断向量设置

ARM7 支持 7 种类型的中断，中断向量一般位于 0X0－0X1C 的 8 个字空间中。但 ARM 也支持高地址向量，也就是中断向量可以位于 0XFFFF0000－0XFFFF001C 的 8 个字空间中，具体情况要看生产厂家是否支持这个设定，中断向量如表 1[1] 所示。每当其中的某个异常发生时，PC 值将被强置到相应的中断向量处，同时对当前运行状态进行压栈保存。每个中断向量处放置一个跳转指令到相应的中断服务程序去进行处理。

在执行 REMAP 之前 AT91M40800 的中断代码形式如下：

```
Reset_Vec       B       Hard_Reset
Undef_Vec       B       Undef_Vec
SWI_Vec         B       SWI_Vec
PAbt_Vec        B       PAbt_Vec
DAbt_Vec        B       DAbt_Vec
                NOP
IRQ_Vec         B       IRQ_Vec
FIQ_Vec         B       FIQ_Vec
```

指令 B 是跳转到标号所在地址的程序上去，在标号处(如 Hard_Reset)编写相关处理程序，标号地址在程序编译时编译器会自动计算。上面这些代码位于 ROM 里 0X0－0X1C 上，当执行 REMAP 后就不起作用了。而新的中断处理程序在 REMAP 前从 ROM 里被复制到内部 8K RAM 中，从地址 0 开始向后以 4 字节对准依次排列，代码形式如下：

```
Vectors         LDR     PC, Reset_Addr
                LDR     PC, Undef_Addr
                LDR     PC, SWI_Addr
                LDR     PC, PAbt_Addr
                LDR     PC, DAbt_Addr
                NOP
                LDR     PC, IRQ_Addr
                LDR     PC, FIQ_Addr
Reset_Addr      DCD     Soft_Reset
Undef_Addr      DCD     Undef_Handler
SWI_Addr        DCD     SWI_Handler
PAbt_Addr       DCD     PAbt_Handler
DAbt_Addr       DCD     DAbt_Handler
                DCD     0
IRQ_Addr        DCD     IRQ_Handler
FIQ_Addr        DCD     FIQ_Handler
Soft_Reset      B       Your Soft_Reset Routine
Undef_Handler   B       Your Undef_Handler Routine
SWI_Handler     B       Your SWI_Handler Routine
PAbt_Handler    B       Your PAbt_
```

Handler Routine

DAbt _ Handler B Your DAbt _ Handler Routine

IRQ _ Handler B Your IRQ _ Handler Routine

FIQ _ Handler B Your FIQ _ Handler Routine

3.3 设置系统时钟

ARM7 片子的主时钟可以工作在很宽的频率范围上，为了减少设计难度，有些 ARM 片子内部集成有 PLL 电路，这样用户就可以用低频率的晶振通过 PLL 电路获得一个较高频率的时钟，从而可降低开发成本。AT91M40800 内部没有 PLL 电路，其系统时钟取决于外部的输入时钟，因此该过程不用考虑。

3.4 中断向量复制

中断向量复制的功能是把前面定义的中断向量代码从 ROM 中复制到 RAM 中，这样当 REMAP 后，RAM 首地址切换到地址 0 上，ARM7 就从 RAM 读取中断指令。中断向量复制使用寄存器批传输指令 LDMIA 和 STMIA 实现，下面是 AT91M40800 的中断向量复制代码：

```
ADR R8,Vectors      ;把 ROM 里中断向量的首地址加载到 R8
MOV                 R9, #RAM_Base_Adr ;把 REMPA 前 RAM 的地址加载到 R9
LDMIA               R8!,{R0-R7} ;复制中断向量代码到 RAM
STMIA               R9!,{R0-R7}
LDMIA               R8!,{R0-R7}
STMIA               R9!,{R0-R7}
```

3.5 设置各模式堆栈

启动代码中各模式堆栈空间的设置是为中断服务的，当系统响应中断时要保存当前运行的状态和当前参数就需要一个存储空间，所以对每个模式都要进行堆栈初始化工作，给每个模式的 SP 定义一个堆栈地址和堆栈空间，堆栈大小视需要而定。需要注意的是，对每种模式进行堆栈初始化都要进入相应的模式，才能给各自的 SP 进行定义，在系统初始化完成之前不能对用户模式的堆栈进行初始化，因为用户模式不能对 CPSR 进行操作，处理器也就不能转换到其它的模式中进行相应的操作。要知道，整个初始化工作都是在特权模式下进行的，系统在上电或复位时即自动进入特权模式。模式的切换是通过寄存器 CPSR(当前程序状态寄存器)的 0－4 位写入相应数值进行设置的。AT91M40800 各模式堆栈空间的设置的汇编代码如下：

```
LDR     R0, =Top_Stack ;加载栈顶地址
MSR     CPSR, #Mode_UND:OR:I_Bit:OR:F_Bit ;进入未定义模式
MOV     SP, R0 ;设置未定义堆栈空间指针
SUB     R0, R0, #UND_Stack_Size ;计算下个堆栈的地址
MSR     CPSR, #Mode_ABT:OR:I_Bit:OR:F_Bit
MOV     SP, R0
SUB     R0, R0, #ABT_Stack_Size
MSR     CPSR, #Mode_FIQ:OR:I_Bit:OR:F_Bit
MOV     SP, R0
SUB     R0, R0, #FIQ_Stack_Size
MSR     CPSR, #Mode_IRQ:OR:I_Bit:OR:F_Bit
MOV     SP, R0
SUB     R0, R0, #IRQ_Stack_Size
MSR     CPSR, #Mode_SVC:OR:I_Bit:OR:F_Bit
MOV     SP, R0
SUB     R0, R0, #SVC_Stack_Size ;计算下个用户堆栈的地址
MSR     CPSR, #Mode_USR ;进入用户模式
MOV     SP, R0 ;设置用户堆栈空间指针
```

3.6 C 程序入口

当以上过程完成后，系统就可以跳到 C 程序处运行主程序了，在 ARM 的汇编语言里调用外部 C 语言程序很简单，首先声明一个外部 C 程序的名称，然后把外部 C 程序的地址加载，通过 BX 指令就可以跳到 C 程序处运行。进入 C 程序的汇编代码如下：

```
IMPORT  __main ;用关键字'IMPORT'声名 main 为外部程序
LDR     R0, =__main ;把 main 程序的入口地址赋予 R0
BX      R0 ;利用 BX 指令跳转到 main 处执行 C 程序
```

下面是C程序main：

int main (void){ 用户自己的主程序 }

3.7 小结

回顾上面的过程，可以看出ARM7的启动代码的编写过程其实并不复杂，只要对照所用芯片的资料，通过其内部寄存器设置好它运行所必须的条件，就可以很容易的正常运行起来。图2是ARM启动代码的流程图[4]。

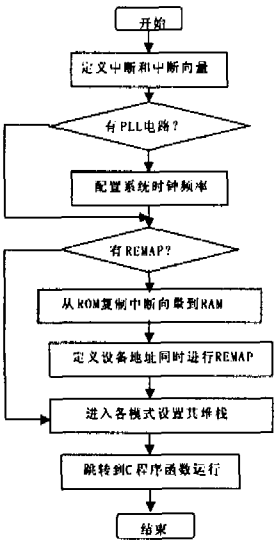

图2 ARM启动代码流程图

4 编译设置和调试

4.1 编译设置

为了让ARM程序的编译器在程序编译链接时清楚存储区都在那些地址范围上以及不同类型的用户数据存放在RAM中的什么位置，就需要通过SCATTER(分散)文件来设置，其格式为文本格式，当用户程序进行编译时，编译器按其能识别的格式读取该文件的内容，根据SCATTER文件里用户的设定对所有数据的存放位置自动进行安排。要注意的是分散描述文件中的定义要按照系统重映射后的存储器分布情况进行[5]。

下面是AT91M40800的分散文件，这里AT91M40800除了8K的片上RAM外，还外扩了16K字节的RAM以及16位宽256K的FLASH ROM，起始地址分别设定在0x02000000和0x01000000。

```
FLASH 0x01000000 0x00040000
{
FLASH 0x01000000 0x00040000 ;ROM起始地址位于0x01000000,大小256K
{
Startup.o(Reset,+FIRST) ;启动代码放到ROM开始位置
*(+RO) ;其它代码和只读数据放在启动代码后面
}
ON_CHIP_RAM 0x00000000 0x00002000;片上RAM起始地址位于0x00,大小8K
{
*.o (+RW) ;一部分程序读写数据放到片上RAM里
}
OFF_CHIP_RAM 0x02000000 0x00004000 ;片外RAM起始地址位于0x02000000,大小16K
{
*.o (+RW,+ZI) ;其它读写数据以及清零区域放在片外RAM里
}
}
```

4.2 调试

当所有程序编写完成并通过编译器的编译链接后，就可以进行调试工作了，调试有模拟调试和板上调试两种，这里主要讨论板上调试工作。为了降低开发成本，ARM支持廉价的JTAG调试。JTAG调试器通过TCK、TMS、TDI、TDO与ARM内核中的ICE模块进行信息交流，使用户可以获得每条指令运行后ARM的状态信息，有DSP开发经验的人对此是很熟悉的。

调试的过程很简单，就是在ARM集成开发环境下通过单步运行来验证程序功能是否与设想一样，笔者使用的是KEIL 7.0的开发环境，ADS 1.2的编译器。首先把编译链接生成的AXF文件通过JTAG调试器下载到板上的FLASH中，完成后ARM从FLASH里读取指令运行，当遇到用户设定的断点后，ARM核各寄存器的信息以及存储器的信息通过JTAG调试器就传到开发环境中，用户根据这些信息解决程序错误，直至程序满足设计要求，最后完成FLASH的烧写。

由于板上调试过程是基于整个ARM硬件

电路板的，所以示波器、万用表这些工具也是必需的，当程序没有错误，但功能仍不满足目的时，就要考虑是否硬件线路的问题了，此时就需要用示波器、万用表参考 PCB 原理图进行观察排故，具体过程不再详述。调试时建议系统时钟不要设置的太高，如果 PCB 布线不合理的话，高时钟会带来一些莫名其妙的问题。另外 AT91M40800 内核和 I/O 工作电压是 3.3V，而外围设备的工作电压一般都是 5V，这些在做原理设计时就应多注意，一旦接错很容易烧片子。

5 结束语

随着工程开发的需求，越来越多的工程技术人员开始使用 ARM 进行开发，本文有助于初学者理解 ARM 的启动过程，结合自己的开发板硬件设置写出相应的启动代码。笔者根据自己的经验，建议初学者开始不要贪多，先抛弃所有可以不需要的设置，只运行几条指令就行，然后慢慢增加代码量，加 C 程序、加中断、加外围设备，等最终这些都实现了，就会发现已经对 ARM 的硬件结构有了较清晰的认识了，实际应用到工程开发中就很容易了。

参考文献

[1] [英]Steve Furber 著，田泽等译. ARM SOC 体系结构[M]. 北京：北京航空航天大学出版社，2002

[2] 马广云等. ARM 嵌入式处理器结构与应用基础. 北京航空航天大学出版社，2002

[3] Atmel Corporation . AT91M40800 DATA SHEET . http ://www. atmel. com ,2002

[4] Atmel Corporation . AT91 Assembler Code StartupSequence for C Code Application Software . http ://www. atmel. com , 2002

[5] ADS Version 1. 2 Linker and Utilities Guide . ARM Limited ,2001

(上接第 59 页)

```
    }
    handleConnection ();  //读服务器文件内容，写入客户机，方法内容省略
                }
        }
    }
```

4 结束语

Java 对网络通信方面的支持是相当强大的，而网络编程又少不了多线程的支持，本文以网络编程对 Java 多线程同步机制进行了探讨，多线程的优点是可以合理协调多个互相独立的任务，但在处理共享数据或共享存储结构时可能会产生数据不一致的错误，导致不可预料的运行结果。所以，线程间的同步，线程间的通信调度是非常重要。因此，我们要深入了解 Java 多线程同步机制，才能更好的发挥多线程应用的优势。

参考文献

[1] 东方人华. Java 2 范例入门与提高. 清华大学出版社，2003

[2] Bruce Eckel . Thinking in Java (2nd Edition). Prentice Hall, 2000

[3] Richard C. Lee . UML 与 Java 面向对象开发实践. 王晨微译. 清华大学出版社，2003

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现

Windows CE:

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究

4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计

Hardware:

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究

16. 基于 UEFI 的 Application 和 Driver 的分析与开发

Programming:

1. 计算机软件基础数据结构 - 算法