|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 00:00:00,262 --> 00:00:01,040 | 各位领导 |  |
| 2 | 00:00:01,040 --> 00:00:02,180 | 和业界同仁下午好 |  |
| 3 | 00:00:02,684 --> 00:00:05,600 | 我是南京沁恒微电子股份有限公司的杨勇 |  |
| 4 | 00:00:06,140 --> 00:00:08,775 | 在这里向各位同仁 |  |
| 5 | 00:00:08,775 --> 00:00:09,962 | 汇报一下我们在 |  |
| 6 | 00:00:10,218 --> 00:00:12,228 | 嵌入式MCU领域的 |  |
| 7 | 00:00:12,525 --> 00:00:14,328 | RISC-V研究和成果 |  |
| 8 | 00:00:14,609 --> 00:00:18,643 | 我的汇报主要分三大部分 |  |
| 9 | 00:00:19,200 --> 00:00:20,003 | 第一部分 |  |
| 10 | 00:00:20,003 --> 00:00:21,275 | 简要介绍一下我们 |  |
| 11 | 00:00:21,606 --> 00:00:24,978 | 沁恒的技术的发展路线 |  |
| 12 | 00:00:24,978 --> 00:00:26,675 | 第二部分是阐述我们在 |  |
| 13 | 00:00:26,943 --> 00:00:30,215 | 研究RISC-V内核的时候做了哪些 |  |
| 14 | 00:00:30,487 --> 00:00:31,768 | 扩展和优化 |  |
| 15 | 00:00:32,000 --> 00:00:34,178 | 第三部分是介绍我们在 |  |
| 16 | 00:00:34,418 --> 00:00:37,618 | RISC-V研究的基础上所形成的 |  |
| 17 | 00:00:37,618 --> 00:00:39,512 | MCU+系列的产品 |  |
| 18 | 00:00:44,284 --> 00:00:45,550 | 沁恒微电子 |  |
| 19 | 00:00:45,550 --> 00:00:47,790 | 主要是成立于2004年 |  |
| 20 | 00:00:47,790 --> 00:00:48,771 | 是位于江苏南京 |  |
| 21 | 00:00:48,771 --> 00:00:51,918 | 是一家专注于接口连接技术的 |  |
| 22 | 00:00:52,143 --> 00:00:53,709 | 集成电路设计公司 |  |
| 23 | 00:00:54,109 --> 00:00:56,437 | 主要产品有以太网 |  |
| 24 | 00:00:56,437 --> 00:00:57,470 | 低功耗蓝牙 |  |
| 25 | 00:00:57,690 --> 00:00:58,596 | USB等 |  |
| 26 | 00:00:58,934 --> 00:01:01,790 | 以及集成上述接口的MCU |  |
| 27 | 00:01:04,056 --> 00:01:06,000 | 这张图是我们公司的 |  |
| 28 | 00:01:06,000 --> 00:01:08,080 | 技术发展历程 |  |
| 29 | 00:01:08,480 --> 00:01:10,260 | 在2004年左右 |  |
| 30 | 00:01:10,261 --> 00:01:12,960 | 我们是自研了一个早期的 |  |
| 31 | 00:01:12,961 --> 00:01:14,280 | 8位的RISC内核 |  |
| 32 | 00:01:14,975 --> 00:01:17,762 | 但周期内核基于它 |  |
| 33 | 00:01:17,762 --> 00:01:19,468 | 研发了系列的一些 |  |
| 34 | 00:01:19,468 --> 00:01:20,650 | 接口转换芯片 |  |
| 35 | 00:01:20,650 --> 00:01:22,884 | 包括大家熟知的CH340系列 |  |
| 36 | 00:01:23,570 --> 00:01:25,310 | 另外在2010年左右 |  |
| 37 | 00:01:25,311 --> 00:01:29,368 | 我们是基于传统的E8051内核 |  |
| 38 | 00:01:29,696 --> 00:01:31,309 | 将间接寻址指令 |  |
| 39 | 00:01:31,309 --> 00:01:32,825 | 进行了单周性化设计 |  |
| 40 | 00:01:33,459 --> 00:01:34,950 | 另外创新的将 |  |
| 41 | 00:01:34,950 --> 00:01:36,890 | DMA引入了我们的51架构 |  |
| 42 | 00:01:37,237 --> 00:01:39,162 | 设计出了多款的以 |  |
| 43 | 00:01:39,162 --> 00:01:40,585 | USB接口为特色的 |  |
| 44 | 00:01:40,586 --> 00:01:42,140 | 高性价比的MCU |  |
| 45 | 00:01:42,778 --> 00:01:46,815 | 随着计算机科学技术的发展 |  |
| 46 | 00:01:46,996 --> 00:01:48,081 | 嵌入式领域 |  |
| 47 | 00:01:48,081 --> 00:01:50,537 | 对于陈述速率和 |  |
| 48 | 00:01:50,537 --> 00:01:52,475 | 低功耗提出了更高的要求 |  |
| 49 | 00:01:52,843 --> 00:01:55,165 | 在2012年左右 |  |
| 50 | 00:01:55,165 --> 00:01:56,503 | 我们引入了ARM的 |  |
| 51 | 00:01:56,503 --> 00:01:58,093 | 多款的32位内核 |  |
| 52 | 00:01:58,425 --> 00:02:01,635 | 同时再结合我们的接口优势 |  |
| 53 | 00:02:01,636 --> 00:02:03,859 | 我们研制了多款的以 |  |
| 54 | 00:02:04,199 --> 00:02:06,498 | 32位内核为基础的一个 |  |
| 55 | 00:02:06,700 --> 00:02:08,575 | MCU芯片 |  |
| 56 | 00:02:09,390 --> 00:02:11,862 | 随着RISC-V指令集 |  |
| 57 | 00:02:11,862 --> 00:02:13,775 | 在2014年正式发布后 |  |
| 58 | 00:02:13,776 --> 00:02:15,568 | 我们公司就一直在关注着 |  |
| 59 | 00:02:16,035 --> 00:02:17,315 | 到2017年左右 |  |
| 60 | 00:02:17,316 --> 00:02:20,906 | 因为公司有安全类产品芯片的需要 |  |
| 61 | 00:02:21,321 --> 00:02:24,735 | 我们就决定切换为RISC-V内核 |  |
| 62 | 00:02:24,736 --> 00:02:27,865 | 但当时第三方的RISC-V内核 |  |
| 63 | 00:02:27,865 --> 00:02:29,135 | 还不够成熟 |  |
| 64 | 00:02:29,136 --> 00:02:30,234 | 内核价格庞大 |  |
| 65 | 00:02:30,735 --> 00:02:32,687 | 没有针对于嵌入式领域 |  |
| 66 | 00:02:32,687 --> 00:02:34,709 | 做一些优化 |  |
| 67 | 00:02:35,568 --> 00:02:38,312 | 结合我们公司在多年的 |  |
| 68 | 00:02:38,312 --> 00:02:40,534 | 内核的设计基础 |  |
| 69 | 00:02:40,755 --> 00:02:43,675 | 以及在接口方面的设计经验 |  |
| 70 | 00:02:43,676 --> 00:02:47,155 | 我们最终决定自研RISC-V内核 |  |
| 71 | 00:02:52,493 --> 00:02:55,368 | 我们在研究RISC-V内核的时候 |  |
| 72 | 00:02:55,368 --> 00:02:57,920 | 有遇到有一些 |  |
| 73 | 00:02:57,920 --> 00:02:59,063 | 针对我们产品 |  |
| 74 | 00:02:59,063 --> 00:03:00,475 | 不是很匹配的地方 |  |
| 75 | 00:03:00,800 --> 00:03:03,568 | 首先是我们在做 |  |
| 76 | 00:03:03,574 --> 00:03:06,753 | 开发低功耗蓝牙的【】内核的时候 |  |
| 77 | 00:03:07,034 --> 00:03:09,903 | 发现是我们原来在 |  |
| 78 | 00:03:10,084 --> 00:03:12,437 | ARM内核已经有量产的 |  |
| 79 | 00:03:12,437 --> 00:03:14,662 | M0内核的MCU |  |
| 80 | 00:03:14,770 --> 00:03:17,103 | 当我们把我们的蓝牙协议栈 |  |
| 81 | 00:03:17,493 --> 00:03:21,028 | 移植到RISC-V内核MCU上时发现 |  |
| 82 | 00:03:21,559 --> 00:03:24,018 | 整个代码空间增大了很多 |  |
| 83 | 00:03:24,562 --> 00:03:27,606 | 后来我们研究指令发现是因为 |  |
| 84 | 00:03:27,728 --> 00:03:30,771 | 现在标准的RISC-V指令集只支持 |  |
| 85 | 00:03:31,115 --> 00:03:33,275 | 压缩指令的话只支持16位 |  |
| 86 | 00:03:33,337 --> 00:03:34,490 | 只是自操作 |  |
| 87 | 00:03:35,028 --> 00:03:37,259 | 但嵌入式里面 |  |
| 88 | 00:03:37,259 --> 00:03:39,612 | 对于半字和字节操作的 |  |
| 89 | 00:03:39,931 --> 00:03:41,171 | 还是非常多的 |  |
| 90 | 00:03:42,659 --> 00:03:44,996 | 后面我们就针对的 |  |
| 91 | 00:03:45,390 --> 00:03:47,650 | 因为我们产品需要 |  |
| 92 | 00:03:47,651 --> 00:03:49,409 | 我们就增加了 |  |
| 93 | 00:03:49,830 --> 00:03:53,640 | 支持半字和字操作的压缩指令 |  |
| 94 | 00:03:54,065 --> 00:03:55,546 | 这是我们做了一个对比实验 |  |
| 95 | 00:03:56,853 --> 00:04:02,612 | 加入了字节和半字操作压缩指令后 |  |
| 96 | 00:04:03,003 --> 00:04:04,712 | 同样的协议 |  |
| 97 | 00:04:04,885 --> 00:04:07,253 | 这是以以太网协议的代码为例 |  |
| 98 | 00:04:07,737 --> 00:04:10,281 | 可以看出实际的一个比较结果 |  |
| 99 | 00:04:10,759 --> 00:04:15,003 | 加入我们的压缩指令之后 |  |
| 100 | 00:04:15,003 --> 00:04:18,215 | 整个代码空间缩减了6% |  |
| 101 | 00:04:18,845 --> 00:04:20,565 | 不过这6%是基于 |  |
| 102 | 00:04:20,821 --> 00:04:22,245 | 同样代码没有做 |  |
| 103 | 00:04:22,246 --> 00:04:24,285 | 因为压缩的大小跟我们 |  |
| 104 | 00:04:24,285 --> 00:04:26,887 | 压缩指令的使用频率是正相关的 |  |
| 105 | 00:04:28,746 --> 00:04:30,440 | 我们同时也知道现在 |  |
| 106 | 00:04:30,525 --> 00:04:33,640 | code size reduction group也在研究 |  |
| 107 | 00:04:33,640 --> 00:04:35,920 | 嵌入式领域的压缩指令 |  |
| 108 | 00:04:35,921 --> 00:04:37,400 | 其实我们这里也呼吁 |  |
| 109 | 00:04:37,400 --> 00:04:38,920 | RISC-V组织能够将 |  |
| 110 | 00:04:39,080 --> 00:04:40,400 | 嵌入式领域的压缩指令 |  |
| 111 | 00:04:40,401 --> 00:04:41,615 | 进行一个标准化处理 |  |
| 112 | 00:04:44,840 --> 00:04:47,140 | 在研究RISC-V内核的时候 |  |
| 113 | 00:04:47,141 --> 00:04:48,240 | 我们又遇到第二个问题 |  |
| 114 | 00:04:48,241 --> 00:04:50,865 | 就是现在的PLIC的终端控制器 |  |
| 115 | 00:04:51,475 --> 00:04:52,962 | 在嵌入式领域 |  |
| 116 | 00:04:53,112 --> 00:04:54,206 | 也会有一个 |  |
| 117 | 00:04:54,478 --> 00:04:56,403 | 它是一种集中式管理的模式 |  |
| 118 | 00:04:57,270 --> 00:05:00,430 | 每一种特权模式都提供 |  |
| 119 | 00:05:00,430 --> 00:05:03,130 | 单独的终端信号给内核 |  |
| 120 | 00:05:03,550 --> 00:05:06,130 | 但是在MCU领域一般的 |  |
| 121 | 00:05:06,550 --> 00:05:10,430 | 一般有支持两种模式的 |  |
| 122 | 00:05:10,431 --> 00:05:11,718 | 有支持单个模式的 |  |
| 123 | 00:05:12,225 --> 00:05:14,250 | 在单个模式MCU应用中 |  |
| 124 | 00:05:14,251 --> 00:05:17,310 | 如果采用集中式管理的PLIC【】的话 |  |
| 125 | 00:05:17,311 --> 00:05:20,243 | 就没法很好做到中断抢占的功能 |  |
| 126 | 00:05:20,956 --> 00:05:22,777 | 但是对于嵌入式程序来说 |  |
| 127 | 00:05:23,178 --> 00:05:26,260 | 设置某个外设的优先级 |  |
| 128 | 00:05:26,260 --> 00:05:28,160 | 以及要求其中断及时响应 |  |
| 129 | 00:05:28,161 --> 00:05:30,053 | 是比较常见的操作 |  |
| 130 | 00:05:33,250 --> 00:05:34,610 | 这是我们做的一个实验 |  |
| 131 | 00:05:34,610 --> 00:05:39,550 | 在传统的控制器 |  |
| 132 | 00:05:39,550 --> 00:05:42,918 | 加软件压栈的方式 |  |
| 133 | 00:05:43,318 --> 00:05:47,859 | 从出发到进入中断 |  |
| 134 | 00:05:47,859 --> 00:05:51,562 | 它所消耗的指令时间 |  |
| 135 | 00:05:52,021 --> 00:05:57,287 | 软件压栈一共占用41个周期 |  |
| 136 | 00:05:59,330 --> 00:06:03,490 | 理论上在标量处理器上 |  |
| 137 | 00:06:03,491 --> 00:06:05,928 | 软件压栈至少需要17个周期 |  |
| 138 | 00:06:06,370 --> 00:06:07,650 | 有17条指令 |  |
| 139 | 00:06:07,650 --> 00:06:09,209 | 每条指令需要一个周期 |  |
| 140 | 00:06:09,510 --> 00:06:11,030 | 再加上取址的时间 |  |
| 141 | 00:06:11,284 --> 00:06:14,318 | 延迟往往是大于19个周期的 |  |
| 142 | 00:06:15,109 --> 00:06:17,568 | 但是在一些低成本的 |  |
| 143 | 00:06:17,568 --> 00:06:18,568 | 嵌入产品上面 |  |
| 144 | 00:06:18,925 --> 00:06:21,325 | 软件压栈至少需要几十个周期 |  |
| 145 | 00:06:21,325 --> 00:06:22,900 | 就像我刚刚说的实验 |  |
| 146 | 00:06:22,900 --> 00:06:25,890 | 消耗了41个周期 |  |
| 147 | 00:06:26,425 --> 00:06:28,090 | 这对于一些中断实时性 |  |
| 148 | 00:06:28,090 --> 00:06:29,225 | 要求比较高的产品 |  |
| 149 | 00:06:29,226 --> 00:06:30,845 | 尤其是在USB3.0 |  |
| 150 | 00:06:31,065 --> 00:06:33,685 | 千兆以太网上面是无法接受的 |  |
| 151 | 00:06:35,240 --> 00:06:38,068 | 还有一些对做嵌入式开发都知道 |  |
| 152 | 00:06:38,068 --> 00:06:41,021 | 在一些对于代码运行效率 |  |
| 153 | 00:06:41,350 --> 00:06:44,175 | 和功耗要求比较苛刻的场景 |  |
| 154 | 00:06:44,175 --> 00:06:46,256 | 我们一般都会把 |  |
| 155 | 00:06:46,256 --> 00:06:47,825 | 代码放在RAM里面运行 |  |
| 156 | 00:06:48,437 --> 00:06:50,059 | 但是在软件压栈去 |  |
| 157 | 00:06:50,059 --> 00:06:51,470 | 中断向量表去寻址的时候 |  |
| 158 | 00:06:51,471 --> 00:06:54,406 | 会遇到跳转指令寻址范围的限制 |  |
| 159 | 00:06:55,378 --> 00:06:59,470 | 如果把代码在RAM运行的话 |  |
| 160 | 00:06:59,470 --> 00:07:02,003 | 就需要把中断向量表 |  |
| 161 | 00:07:02,003 --> 00:07:03,675 | 和中断处理函数 |  |
| 162 | 00:07:03,675 --> 00:07:05,518 | 都放在同一块存储区域 |  |
| 163 | 00:07:05,820 --> 00:07:07,060 | 如果都放到RAM里面 |  |
| 164 | 00:07:07,066 --> 00:07:09,831 | 那所有的中断处理函数 |  |
| 165 | 00:07:09,831 --> 00:07:10,986 | 都要一起放RAM |  |
| 166 | 00:07:11,037 --> 00:07:13,160 | 这会导致一个问题 |  |
| 167 | 00:07:13,161 --> 00:07:15,443 | 就是经常会遇到一些 |  |
| 168 | 00:07:15,443 --> 00:07:17,335 | 并文件拼接的问题 |  |
| 169 | 00:07:17,335 --> 00:07:19,500 | 这种情况下是无法做到的 |  |
| 170 | 00:07:20,700 --> 00:07:22,875 | 从这个图上也可以看出 |  |
| 171 | 00:07:22,875 --> 00:07:27,105 | 寻址范围的受限的问题 |  |
| 172 | 00:07:29,275 --> 00:07:31,043 | 针对前面遇到的 |  |
| 173 | 00:07:31,043 --> 00:07:32,150 | 软件压栈问题 |  |
| 174 | 00:07:32,537 --> 00:07:36,320 | 还有中断延迟偏大 |  |
| 175 | 00:07:36,662 --> 00:07:41,037 | 我们针对于设计需要 |  |
| 176 | 00:07:41,040 --> 00:07:43,028 | 我们设计了一个硬件压栈的方式 |  |
| 177 | 00:07:43,437 --> 00:07:48,570 | 同时结合【】编译器的一个优化 |  |
| 178 | 00:07:48,830 --> 00:07:50,806 | 我们设计的如下 |  |
| 179 | 00:07:50,806 --> 00:07:52,521 | 有两种的硬件压栈方式 |  |
| 180 | 00:07:52,903 --> 00:07:56,093 | 一种是上下文保存式 |  |
| 181 | 00:07:56,093 --> 00:07:57,984 | 寄存器或者储存器 |  |
| 182 | 00:07:58,509 --> 00:08:00,303 | 这种方式就会 |  |
| 183 | 00:08:00,303 --> 00:08:01,630 | 增加一定的硬件开销 |  |
| 184 | 00:08:01,631 --> 00:08:03,962 | 但是中断响应延迟 |  |
| 185 | 00:08:03,962 --> 00:08:06,175 | 可以做到两个周期 |  |
| 186 | 00:08:07,230 --> 00:08:08,468 | 另外一种方式是 |  |
| 187 | 00:08:08,737 --> 00:08:11,759 | 硬件保存至堆栈区 |  |
| 188 | 00:08:12,154 --> 00:08:16,878 | 相比较一的话 |  |
| 189 | 00:08:16,878 --> 00:08:19,909 | 硬件消耗大概在 |  |
| 190 | 00:08:19,909 --> 00:08:21,231 | 100到200个门左右 |  |
| 191 | 00:08:21,231 --> 00:08:22,375 | 还是比较低的 |  |
| 192 | 00:08:22,806 --> 00:08:24,975 | 但是延迟可以做到 |  |
| 193 | 00:08:24,975 --> 00:08:26,393 | 大概17个周期左右 |  |
| 194 | 00:08:30,668 --> 00:08:34,075 | 根据我们在产品的需求 |  |
| 195 | 00:08:34,076 --> 00:08:35,062 | 在低功耗蓝牙 |  |
| 196 | 00:08:35,062 --> 00:08:38,109 | 或者是在USB3.0的应用需要 |  |
| 197 | 00:08:38,312 --> 00:08:42,137 | 针对于PLIC终端控制器的诸多不便 |  |
| 198 | 00:08:42,668 --> 00:08:45,610 | 我们就自行设计了一种 |  |
| 199 | 00:08:45,650 --> 00:08:46,956 | 快速中断控制器 |  |
| 200 | 00:08:47,571 --> 00:08:48,355 | FPIC |  |
| 201 | 00:08:49,110 --> 00:08:50,750 | 从示意图上可以看出来 |  |
| 202 | 00:08:51,346 --> 00:08:53,210 | 我们所有的外设中断都是 |  |
| 203 | 00:08:53,210 --> 00:08:56,178 | 分别单独送到内核 |  |
| 204 | 00:08:57,490 --> 00:08:59,306 | 每种跳转方式 |  |
| 205 | 00:08:59,306 --> 00:09:00,140 | 我们还设计了两种 |  |
| 206 | 00:09:00,141 --> 00:09:03,560 | 一种是常规的fast interrupt的方式 |  |
| 207 | 00:09:03,560 --> 00:09:08,043 | 也是通过查询中断向量表 |  |
| 208 | 00:09:08,440 --> 00:09:09,943 | 但是我们是支持 |  |
| 209 | 00:09:09,943 --> 00:09:12,314 | 绝对地址的一个寻址 |  |
| 210 | 00:09:12,500 --> 00:09:14,320 | 这样就可以避免前面说到的 |  |
| 211 | 00:09:14,660 --> 00:09:16,580 | 跳转指令寻址范围的限制 |  |
| 212 | 00:09:17,340 --> 00:09:22,903 | 另外我们在上面又设计了一个 |  |
| 213 | 00:09:22,903 --> 00:09:24,321 | 不用查询就要VTF |  |
| 214 | 00:09:24,321 --> 00:09:27,784 | vector table free interrupt这种方式 |  |
| 215 | 00:09:27,784 --> 00:09:30,668 | 不用查询中断向量表 |  |
| 216 | 00:09:30,668 --> 00:09:31,950 | 直接跳转 |  |
| 217 | 00:09:32,270 --> 00:09:34,070 | 直接中断函数 直接寻址 |  |
| 218 | 00:09:34,550 --> 00:09:36,103 | 这种方式就会 |  |
| 219 | 00:09:36,103 --> 00:09:38,630 | 更加减少中断延时 |  |
| 220 | 00:09:38,950 --> 00:09:41,790 | 在高速接口的MCU应用中 |  |
| 221 | 00:09:41,790 --> 00:09:44,495 | 会增加通信 |  |
| 222 | 00:09:44,775 --> 00:09:46,175 | 提高通信效率 |  |
| 223 | 00:09:47,695 --> 00:09:50,118 | 下面我做了一些对比实验 |  |
| 224 | 00:09:51,518 --> 00:09:53,146 | 这个实验是在我们 |  |
| 225 | 00:09:53,146 --> 00:09:55,631 | 只开启相对于软件压栈 |  |
| 226 | 00:09:55,631 --> 00:09:58,035 | 我们只开启硬件压栈的方式 |  |
| 227 | 00:09:58,035 --> 00:10:01,112 | 从中断触发到 |  |
| 228 | 00:10:01,112 --> 00:10:02,655 | 进入中断处理函数 |  |
| 229 | 00:10:02,980 --> 00:10:05,581 | 我们大概花费了8个周期 |  |
| 230 | 00:10:06,100 --> 00:10:08,221 | 相对于前面41个周期 |  |
| 231 | 00:10:08,340 --> 00:10:11,690 | 我们效率提升了5.1倍 |  |
| 232 | 00:10:17,462 --> 00:10:21,085 | 这张实验图是我们在 |  |
| 233 | 00:10:21,086 --> 00:10:22,545 | 硬件压栈的基础上面 |  |
| 234 | 00:10:22,915 --> 00:10:25,493 | 开启了VTF中断模式 |  |
| 235 | 00:10:26,515 --> 00:10:29,475 | 从左上角的汇编指令 |  |
| 236 | 00:10:29,475 --> 00:10:30,275 | 可以看出来 |  |
| 237 | 00:10:30,584 --> 00:10:34,275 | 汇编加了VTF |  |
| 238 | 00:10:34,275 --> 00:10:36,046 | 和硬件压栈的程序是一样的 |  |
| 239 | 00:10:36,628 --> 00:10:39,203 | 但是它少了一个 |  |
| 240 | 00:10:39,203 --> 00:10:43,137 | 上下文保存的过程 |  |
| 241 | 00:10:43,825 --> 00:10:47,125 | 现在是至少是少了两个周期 |  |
| 242 | 00:10:47,434 --> 00:10:50,945 | 时间是大概是6个周期左右 |  |
| 243 | 00:10:50,946 --> 00:10:53,734 | 所以相对于纯软件压栈 |  |
| 244 | 00:10:53,734 --> 00:10:59,215 | 中断速率提升到了原来的6.8倍 |  |
| 245 | 00:10:59,934 --> 00:11:04,509 | 在我们的USB3.0 3.1 千兆以太网中 |  |
| 246 | 00:11:04,590 --> 00:11:07,343 | 已经完全使用硬件压栈 |  |
| 247 | 00:11:07,343 --> 00:11:08,615 | 和VTF中断模式 |  |
| 248 | 00:11:09,610 --> 00:11:11,870 | 通信传输效率提升显著 |  |
| 249 | 00:11:16,695 --> 00:11:19,706 | 另外我们在研究RISC-V |  |
| 250 | 00:11:20,259 --> 00:11:22,915 | 在设计低功耗蓝牙的RISC-V内核的时候 |  |
| 251 | 00:11:22,916 --> 00:11:23,915 | 又遇到另外一个问题 |  |
| 252 | 00:11:23,916 --> 00:11:27,418 | 标准的RISC-V |  |
| 253 | 00:11:27,418 --> 00:11:29,950 | 只有WFI的睡眠方式 |  |
| 254 | 00:11:30,690 --> 00:11:32,830 | 但是我们有些在应用中 |  |
| 255 | 00:11:33,450 --> 00:11:35,259 | 相对于WFE |  |
| 256 | 00:11:35,943 --> 00:11:38,083 | WFI唤醒源比较有限 |  |
| 257 | 00:11:38,400 --> 00:11:41,806 | 同时对实习要求比较高的场景的话 |  |
| 258 | 00:11:42,178 --> 00:11:46,980 | 我们一般都会使用WFE的方式 |  |
| 259 | 00:11:47,680 --> 00:11:49,178 | 另外我们还设计了一个 |  |
| 260 | 00:11:49,503 --> 00:11:51,334 | 多级睡眠模式 |  |
| 261 | 00:11:53,285 --> 00:11:54,665 | 在debug调试方面 |  |
| 262 | 00:11:54,666 --> 00:11:56,985 | 在嵌入式领域硬件资源比较紧张 |  |
| 263 | 00:11:57,440 --> 00:12:00,706 | 现有标准的RISC-V |  |
| 264 | 00:12:00,706 --> 00:12:02,396 | 只有JTAG调试接口 |  |
| 265 | 00:12:02,700 --> 00:12:04,337 | 所占用的硬件资源较多 |  |
| 266 | 00:12:04,575 --> 00:12:06,225 | 我们为了方便调试 |  |
| 267 | 00:12:06,225 --> 00:12:07,471 | 和资源利用最大化 |  |
| 268 | 00:12:07,775 --> 00:12:09,428 | 我们精简了JTAG调试接口 |  |
| 269 | 00:12:09,428 --> 00:12:11,412 | 改用为自定义的两线调试接口 |  |
| 270 | 00:12:12,043 --> 00:12:13,340 | 以减少I/O资源 |  |
| 271 | 00:12:13,340 --> 00:12:14,725 | 同时也兼顾了速度 |  |
| 272 | 00:12:14,726 --> 00:12:15,765 | 我们还做了一个 |  |
| 273 | 00:12:16,025 --> 00:12:18,021 | 调试速度的实验 |  |
| 274 | 00:12:18,537 --> 00:12:21,385 | 上面是采用JTAG debug做 |  |
| 275 | 00:12:21,691 --> 00:12:26,646 | 都是以RISC-V产品的JTAG debug为对比 |  |
| 276 | 00:12:26,646 --> 00:12:28,365 | 一个是上面是JTAG的模式 |  |
| 277 | 00:12:28,705 --> 00:12:31,693 | 下面是两线调试的模式 |  |
| 278 | 00:12:32,259 --> 00:12:33,918 | 两线的实测的话 |  |
| 279 | 00:12:33,918 --> 00:12:36,990 | 调试速度是JTAG的3.8倍 |  |
| 280 | 00:12:42,971 --> 00:12:44,806 | 最后介绍一下我们 |  |
| 281 | 00:12:44,806 --> 00:12:47,896 | 在研究RISC-V内核的同时 |  |
| 282 | 00:12:48,395 --> 00:12:49,821 | 所带来哪些 |  |
| 283 | 00:12:49,821 --> 00:12:51,459 | 比较有特色的产品 |  |
| 284 | 00:12:51,900 --> 00:12:55,190 | 目前国产MCU芯片的性能都很不错 |  |
| 285 | 00:12:55,535 --> 00:12:57,546 | 有些已经国际领先了 |  |
| 286 | 00:12:57,843 --> 00:12:59,295 | 前面在那也提了不少 |  |
| 287 | 00:12:59,296 --> 00:13:00,155 | 我这边就不展开了 |  |
| 288 | 00:13:00,156 --> 00:13:01,275 | 我主要就介绍一下 |  |
| 289 | 00:13:01,605 --> 00:13:03,731 | 沁恒在MCU+方面的 |  |
| 290 | 00:13:03,731 --> 00:13:04,531 | 专业扩展工作 |  |
| 291 | 00:13:04,531 --> 00:13:05,905 | 做一下简单的汇报 |  |
| 292 | 00:13:06,459 --> 00:13:10,025 | 我们现在主要是以1核3接口 |  |
| 293 | 00:13:10,065 --> 00:13:12,759 | 1核是RISC-V内核 |  |
| 294 | 00:13:12,759 --> 00:13:13,737 | 接口的话3个 |  |
| 295 | 00:13:13,737 --> 00:13:15,465 | 以太网 蓝牙 USB |  |
| 296 | 00:13:15,890 --> 00:13:18,137 | 形成了一个MCU+的一个产品线 |  |
| 297 | 00:13:18,490 --> 00:13:20,753 | MCU+就是集成 |  |
| 298 | 00:13:20,753 --> 00:13:23,640 | 低功耗蓝牙5.1的MCU |  |
| 299 | 00:13:23,853 --> 00:13:25,954 | 集成以太网的MCU |  |
| 300 | 00:13:26,284 --> 00:13:27,957 | 另外集成USB的MCU |  |
| 301 | 00:13:28,360 --> 00:13:29,940 | 其中蓝牙的基带 |  |
| 302 | 00:13:29,940 --> 00:13:32,040 | 射频收发器协议栈 |  |
| 303 | 00:13:32,650 --> 00:13:34,560 | 以太网的MAC PHY |  |
| 304 | 00:13:34,560 --> 00:13:36,606 | USB PHY都是自主研发 |  |
| 305 | 00:13:40,075 --> 00:13:42,225 | 另外针对嵌入式不同的应用领域 |  |
| 306 | 00:13:42,226 --> 00:13:43,550 | 对速度 功耗 |  |
| 307 | 00:13:43,550 --> 00:13:44,965 | 性能的不同的要求 |  |
| 308 | 00:13:44,966 --> 00:13:48,325 | 我们设计了一系列的内核 |  |
| 309 | 00:13:48,326 --> 00:13:51,751 | 有V3A V4B V4C V4F |  |
| 310 | 00:13:51,796 --> 00:13:54,090 | 其他V4F是支持浮点运算 |  |
| 311 | 00:13:54,090 --> 00:13:55,359 | 和内存保护 |  |
| 312 | 00:13:55,834 --> 00:13:58,875 | Dhrystone跑分大概在 |  |
| 313 | 00:13:58,875 --> 00:14:00,990 | 1.89 TIMx MGHz |  |
| 314 | 00:14:02,656 --> 00:14:08,056 | 另外还有一些集成的蓝牙加 |  |
| 315 | 00:14:08,056 --> 00:14:10,525 | 32位通用MCU的一些产品 |  |
| 316 | 00:14:10,825 --> 00:14:13,606 | 目前以RISC-V内核的MCU+产品 |  |
| 317 | 00:14:13,606 --> 00:14:15,440 | 月出货量已经达到了KK级 |  |
| 318 | 00:14:16,925 --> 00:14:21,435 | 我们坚持是研究MCU的 |  |
| 319 | 00:14:21,436 --> 00:14:23,215 | 一直是坚持应用驱动 |  |
| 320 | 00:14:23,853 --> 00:14:25,335 | 深度结合 |  |
| 321 | 00:14:25,336 --> 00:14:28,643 | 我们将针对应用继续优化改进 |  |
| 322 | 00:14:29,087 --> 00:14:32,043 | 助力RISC-V更好 |  |
| 323 | 00:14:32,043 --> 00:14:33,503 | 更快地扎根落地 |  |
| 324 | 00:14:33,870 --> 00:14:35,943 | 并在应用中不断壮大 |  |
| 325 | 00:14:36,575 --> 00:14:37,803 | 我的报告就到这里 |  |
| 326 | 00:14:37,803 --> 00:14:38,293 | 谢谢大家 |  |