一. Cortex-A9 MPCore双核通信方案总体流程

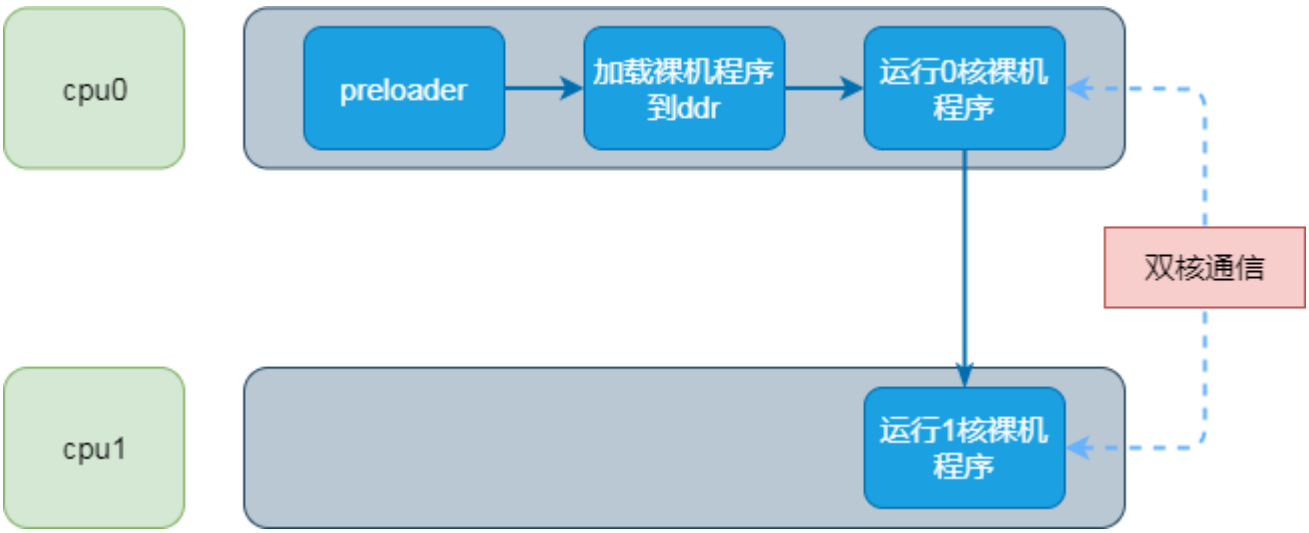


图1 双核通信方案流程

Cortex-A9 MPCore双核通信方案如上图所示。上电时cpu0运行，而cpu1处于休眠状态。cpu0从复位异常向量地址处0x0开始运行，即厂家的固件，固件代码将preloader加载到on-chip ram上运行。preloader负责初始化ddr，然后将0核裸机程序cpu0.bin和1核裸机程序cpu1.bin分别从qspi加载到对应的链接地址，然后运行0核裸机程序，0核裸机程序负责唤醒1核然后进行双核通信。各模块主要工作：

1. preloader

- 初始化ddr和qspi
- 加载0核裸机程序和1核裸机程序到ddr对应的连接地址中
- 启动0核裸机程序

2. 0核裸机程序

- 设置异常向量表和异常处理函数
- 初始化L1cache
- 设置栈
- 设置页表
- 设置和使能中断控制器的分发器和CPU接口，注册中断处理函数
- 唤醒1核
- 使能SCU，SMP，MMU和L1 cache
- 等待1核的交互通信

3. 1核裸机程序

- 设置异常向量表和异常处理函数
- 初始化L1cache
- 设置栈
- 设置页表
- 设置和使能中断控制器的CPU接口，注册中断处理函数
- 使能SMP，MMU和L1 cache
- 设置和使能L2 cache
- 发起和0核的交互通信

二. 内存分配

1. 物理内存划分

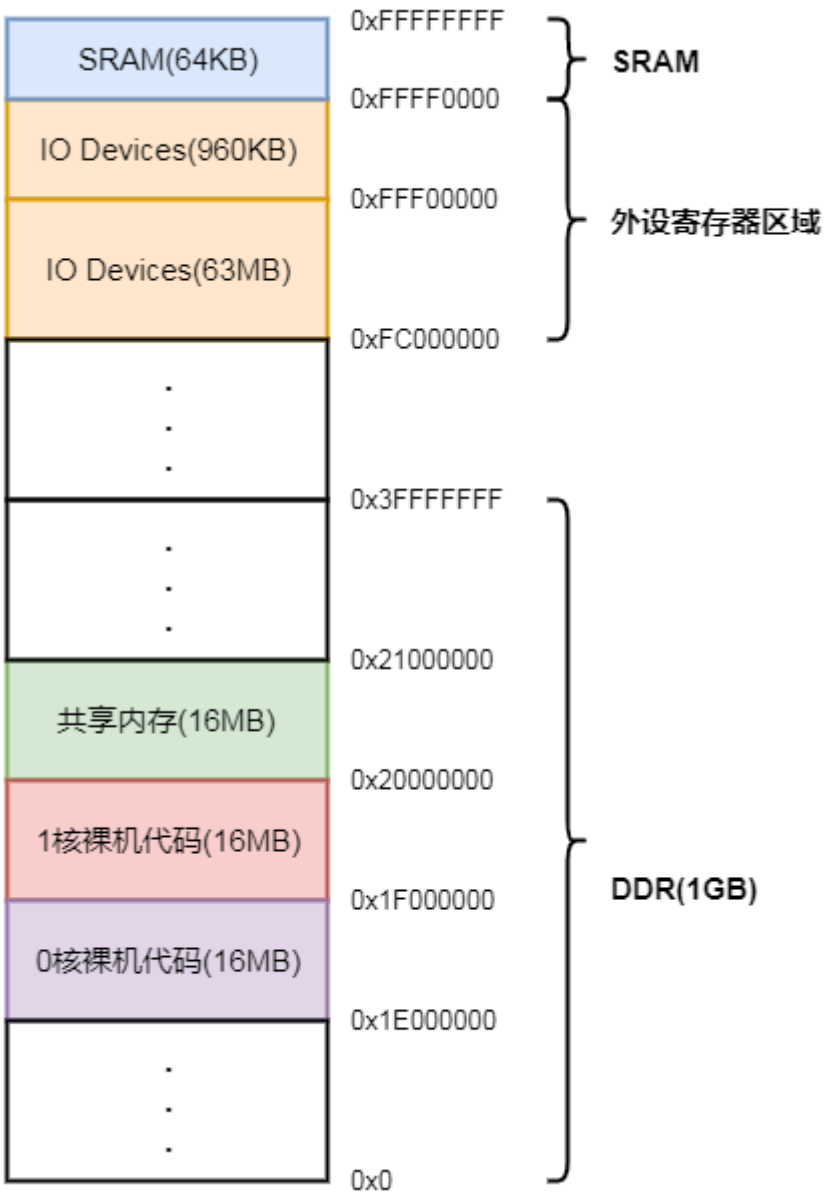


图2 物理内存分布

上图是双裸机通信方案的物理内存分布。具体分配如下：

- 片上静态内存SRAM：0xFFFF0000~0xFFFFFFFF，大小64KB
- SOC外设寄存器空间：0xFC000000~0xFFFEFFFF，大小63MB+960KB
- 共享内存区域：0x20000000~0x20FFFFFF，大小16MB
- 1核裸机代码区域：0x1F000000~0x1FFFFFFF，大小16MB
- 0核裸机代码区域：0x1E000000~0x1EFFFFFF，大小16MB

2. 0核裸机虚拟地址映射

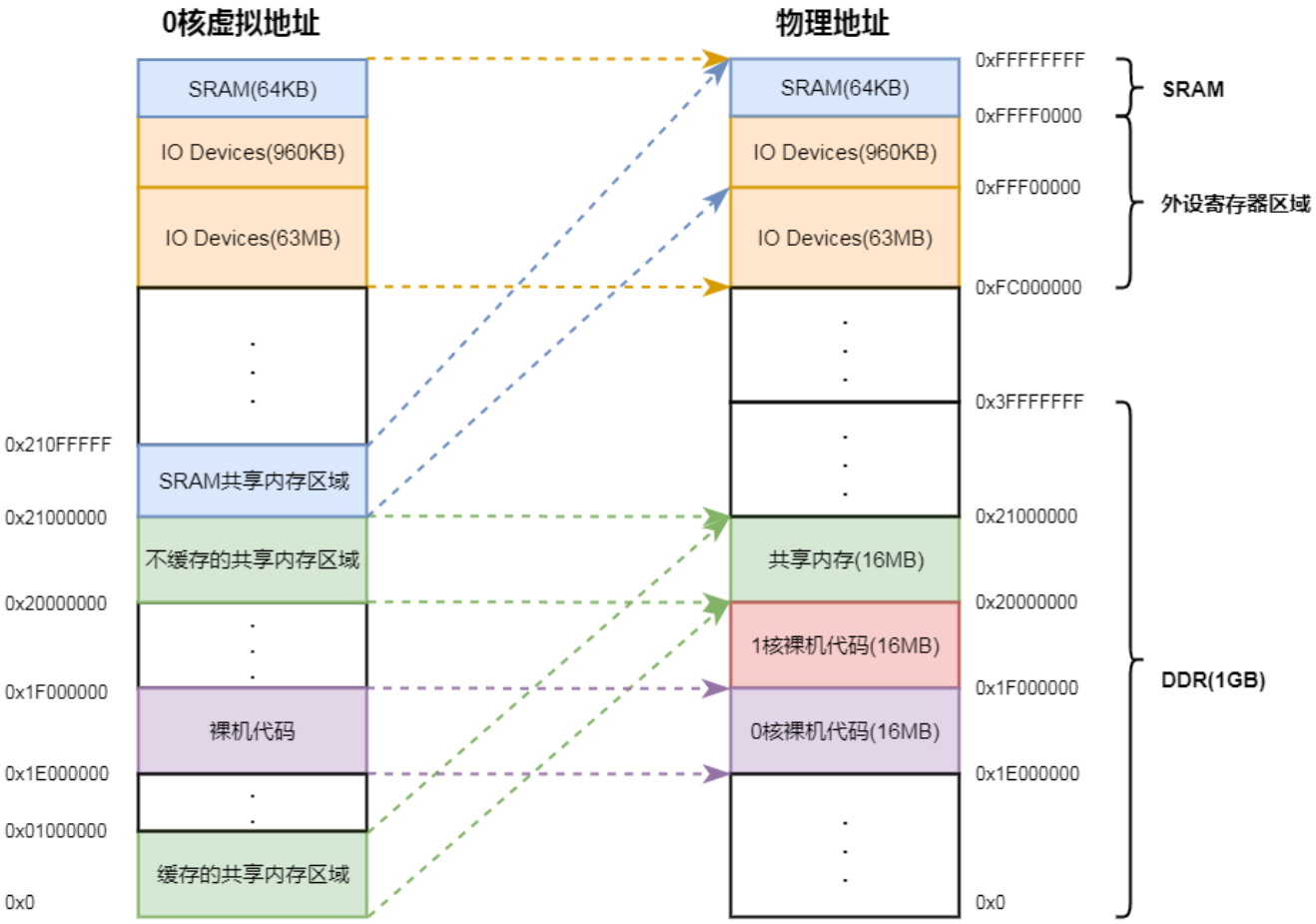


图3 0核裸机虚拟地址映射

3. 1核裸机虚拟地址映射

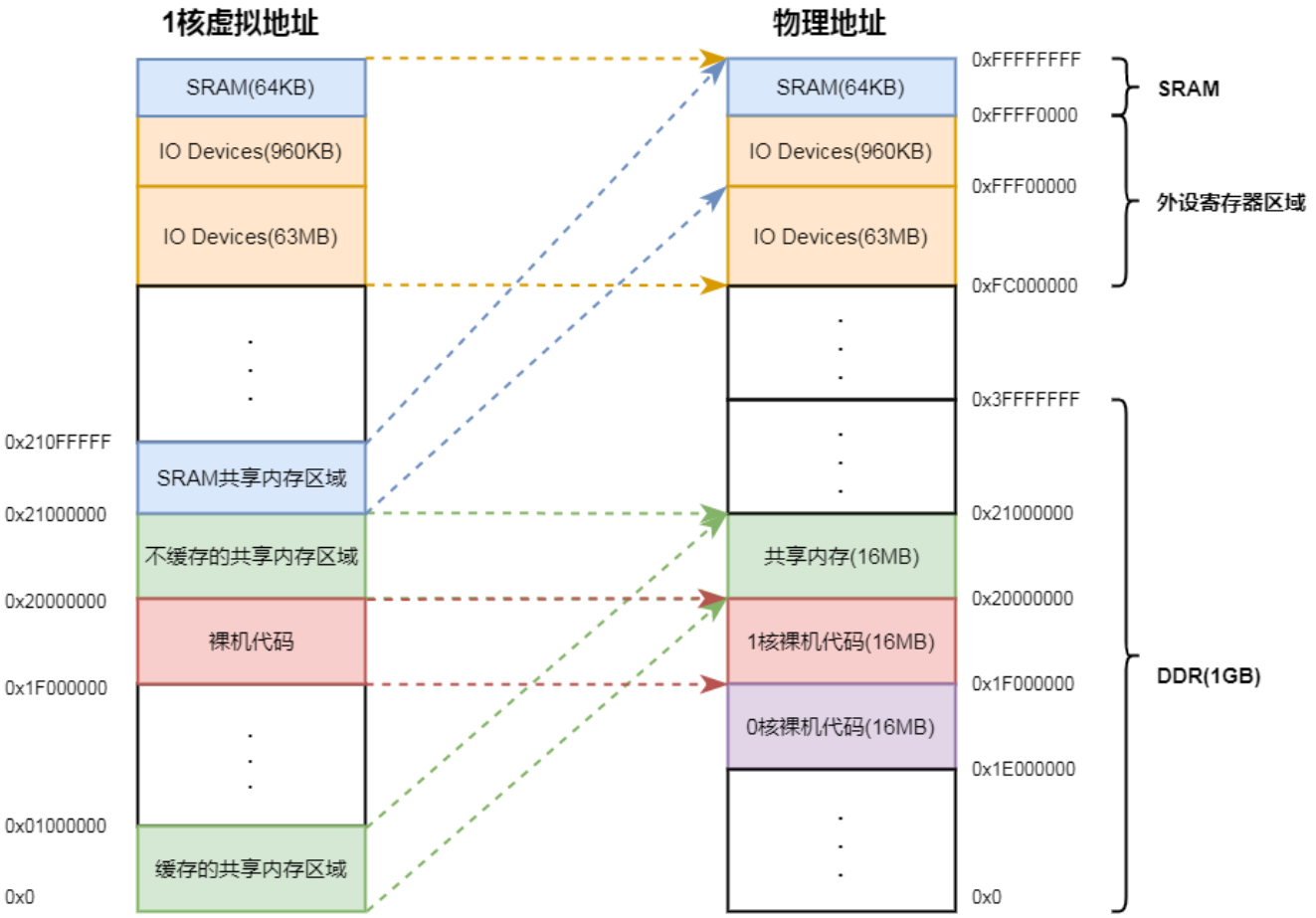


图4 1核裸机虚拟地址映射

三. 测试方案设计

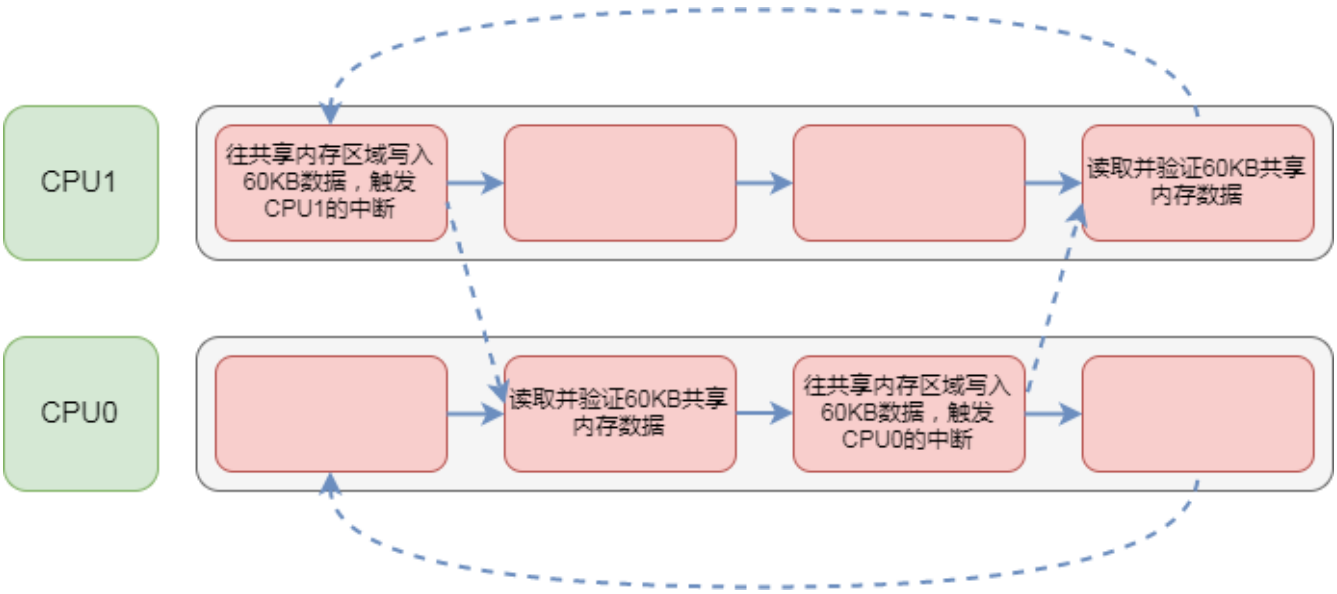


图5 双核通信测试流程

上图为CPU0和CPU1之间的测试共享内存读写速率的一个循环周期。当0核裸机程序做完0核相关初始化后唤醒1核然后进入一个循环。1核启动后也需要相关的初始化，然后往共享内存写入60KB约定好的特定的数据，发送软中断通知0核，接着也和0核一样进入一个循环，在这个循环里两个核之间就按照上图一直进行交互通信。以下是较为详细的步骤：

首先在双核通信之前cpu0和cpu1都注册一个软中断处理函数，在这个中断处理函数中点亮led。

1. cpu1往共享内存中写入60KB的0xAA，然后触发相应的软中断给cpu0。

2. cpu0收到中断后会触发irq异常，最后会进入之前注册的中断处理函数，然后将cpu0控制的led点亮，接着从共享内存区域读取并验证这60KB数据是不是0xAA，只要有一个数据不是0xAA就会在串口输出错误信息。

3. cpu0读取验证完数据后将60KB的0x55写入共享内存，然后关闭cpu0控制的led，触发相应的软中断给cpu1。

4. cpu1收到中断后在中断处理函数中将cpu1控制的led点亮，接着从共享内存区域读取并验证这60KB数据是不是0x55，只要有一个数据不是0x55就会在串口输出错误信息。

然后回到第1步开始下一个循环。根据以上的双核通信规律，利用示波器测出led电平的频率即可计算出双核通信的读写速率。

四. 测试、分析与验证

以下速率的测试方法是利用示波器测量每个循环切换GPIO管脚电平的频率来计算通信速率，计算公式：频率 * 2 *（每半个周期的读写次数和）* (读写数据大小)

1. DDR读写速率测试

1.1 无缓存使能

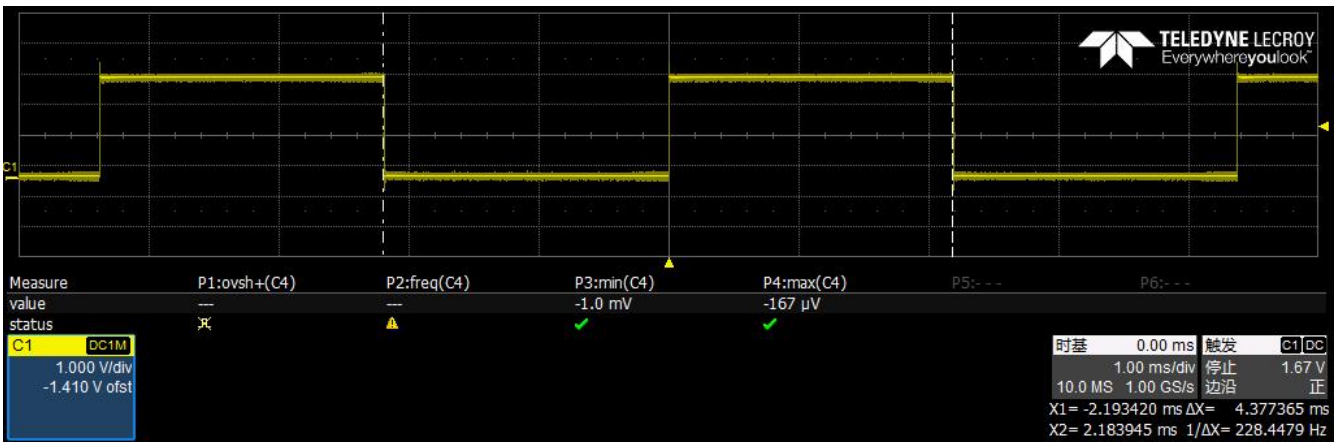


图6 无缓存使能

读写数据大小：60K 缓存使能：无 频率：228.4479Hz 周期：4.377365ms 每半个周期的读写次数和：2次（一次读操作、一次写操作） 速率：438.92Mbps (228.4479 * 2 * 2* 60K = 54827496B/s = 438.92Mbps)

1.2 L1缓存使能

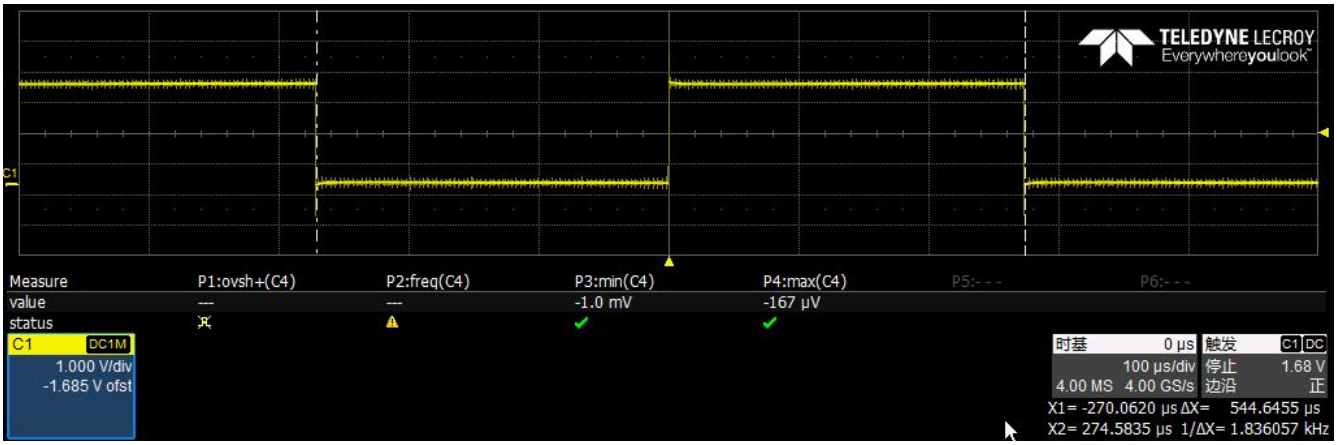


图7 L1缓存使能

读写数据大小：60K 缓存使能：L1缓存使能、L2缓存不使能 频率：1.836057kHz 周期：544.6455us 每半个周期的读写次数和：2次（一次读操作、一次写操作） 速率：3.5Gbps(1.836057k * 2 * 2* 60K = 3.5Gbps)

1.3 L1、L2缓存使能

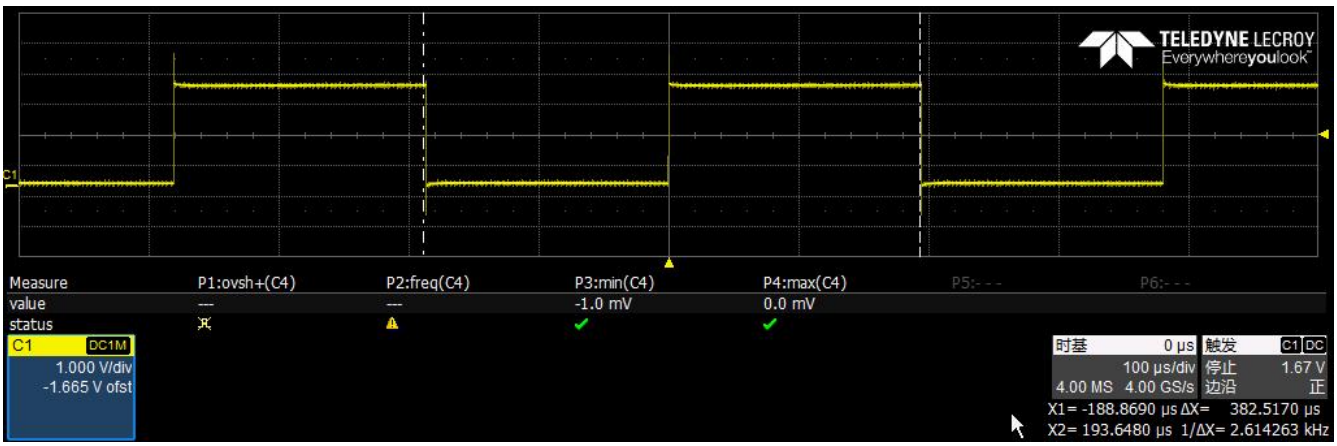


图8 L1、L2缓存使能

读写数据大小：60K 缓存使能：L1缓存使能、L2缓存使能 频率：2.614263kHz 周期：382.5170us 每半个周期的读写次数和：2次（一次读操作、一次写操作） 速率：5.0Gbps(2.614263k * 2 * 2* 60K = 5.0Gbps)

2. SRAM读写速率测试

2.1 无缓存使能

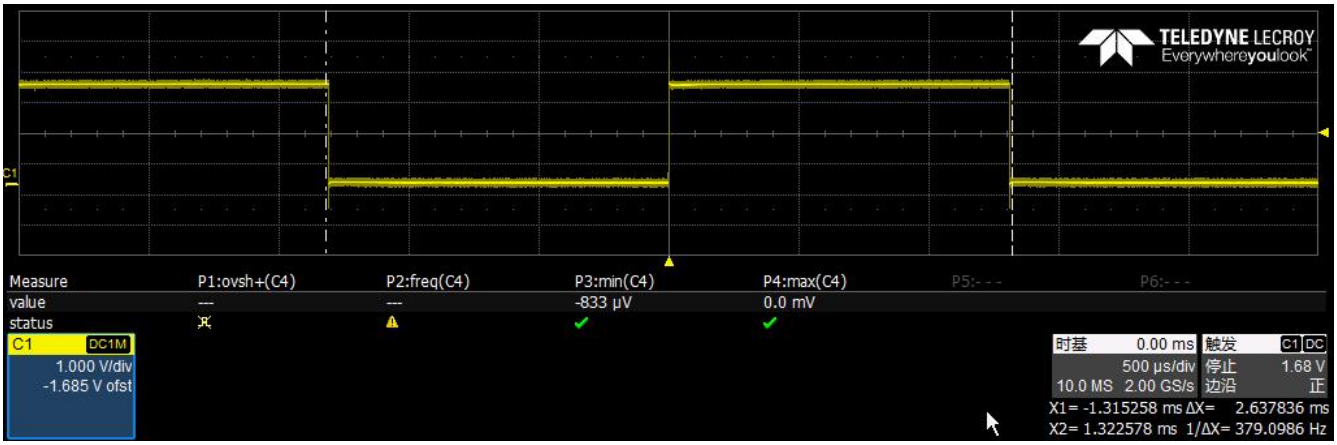


图9 无缓存使能

读写数据大小：60K 缓存使能：无 频率：379.0986Hz 周期：2.637836ms 每半个周期的读写次数和：2次（一次读操作、一次写操作）速率：727.87Mbps (379.0986 * 2 * 2 * 60K = 727.87Mbps)

2.2 L1缓存使能

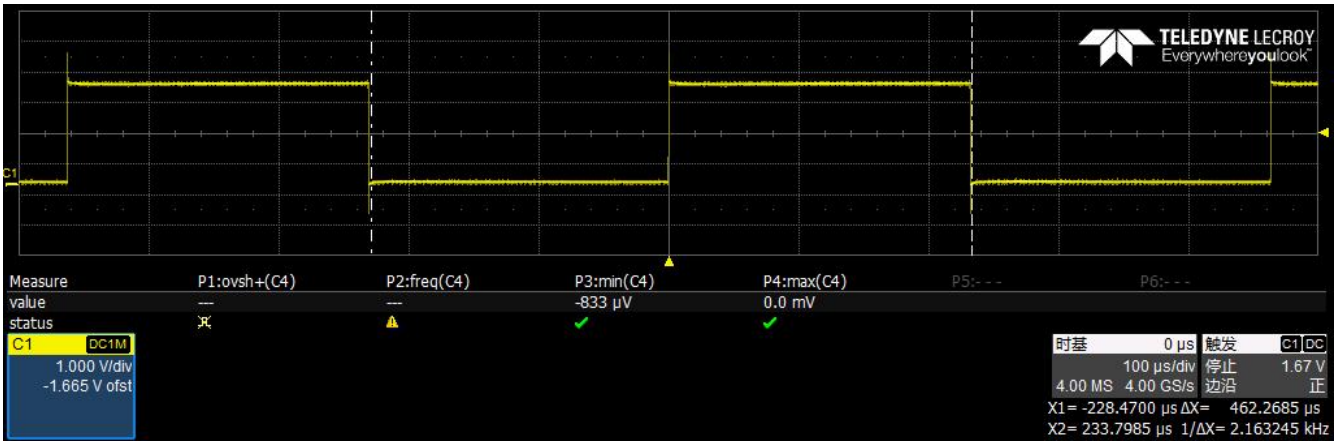


图10 L1缓存使能

读写数据大小：60K 缓存使能：L1缓存使能、L2缓存不使能 频率：2.163245kHz 周期：462.2685us 每半个周期的读写次数和：2次（一次读操作、一次写操作）速率：4.15Gbps(2.163245k * 2 * 2 * 60K = 4.15Gbps)

2.3 L1、L2缓存使能

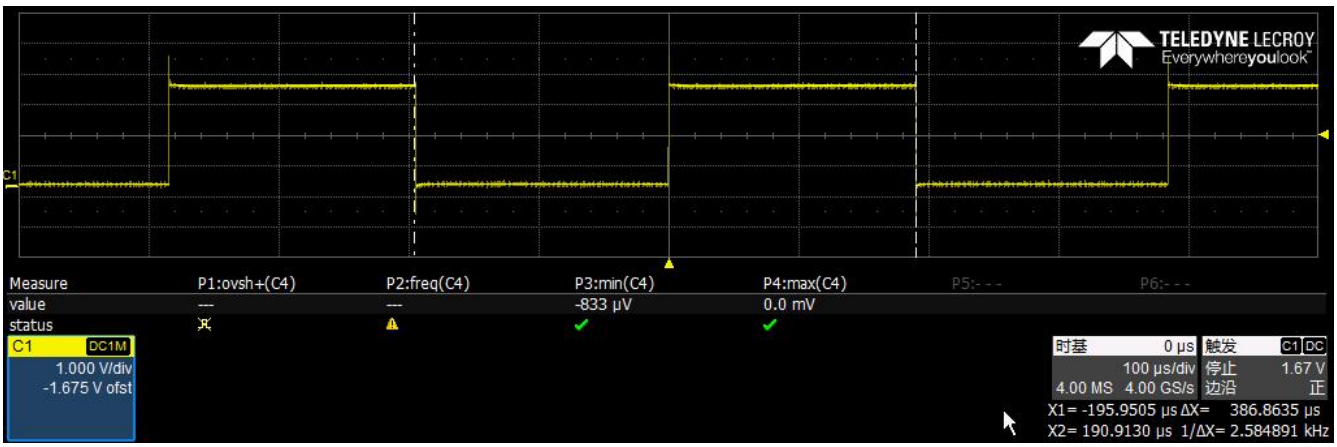


图11 L1、L2缓存使能

读写数据大小：60K 缓存使能：L1缓存使能、L2缓存使能 频率：2.584891kHz 周期：386.8636us 每半个周期的读写次数和：2次（一次读操作、一次写操作） 速率：4.96Gbps(2.584891k * 2 * 2 * 60K = 4.96Gbps)

3. 速率对比分析

存储设备	单次读写buff大小	缓存使用情况	频率	速度
DDR	60K	无缓存使能	228.4479Hz	438.92Mbps
DDR	60K	L1缓存使能、L2缓存不使能	1.836057kHz	3.5Gbps
DDR	60K	L1缓存使能、L2缓存使能	2.614263kHz	5.0Gbps
SRAM	60K	无缓存使能	379.0986Hz	727.87Mbps
SRAM	60K	L1缓存使能、L2缓存不使能	2.163245kHz	4.15Gbps
SRAM	60K	L1缓存使能、L2缓存使能	2.584891kHz	4.96Gbps

图12 速度对比

根据前文计算的结果绘制出速度对比表格。上表的结果和理论保持一致，对于同样一种存储设备，开了缓存后的读写速度比没有开缓存的情况下大大提高，开了L1和L2比只开L1要快，sram的读写速度比ddr的读写速度快。唯一看起来有点不太正常的现象是同样都使能了L1和L2的DDR和SRAM的读写速度看起来是一样的。其实这是因为读写buff的大小限制所导致的，因为sram只有64KB，这里使用了不超过该容量的60KB，而L2 cache的大小有512KB，因此读写过程中在L1中没有缓存到的数据会全部缓存到L2中，因为这60KB地址是连续的，每一次对内存的写操作不会发生驱逐，能全部写到缓存当中。因此在buff大小不超过512KB且连续的情况下存储设备的差异对L1和L2缓存都使能的读写速率没有影响。

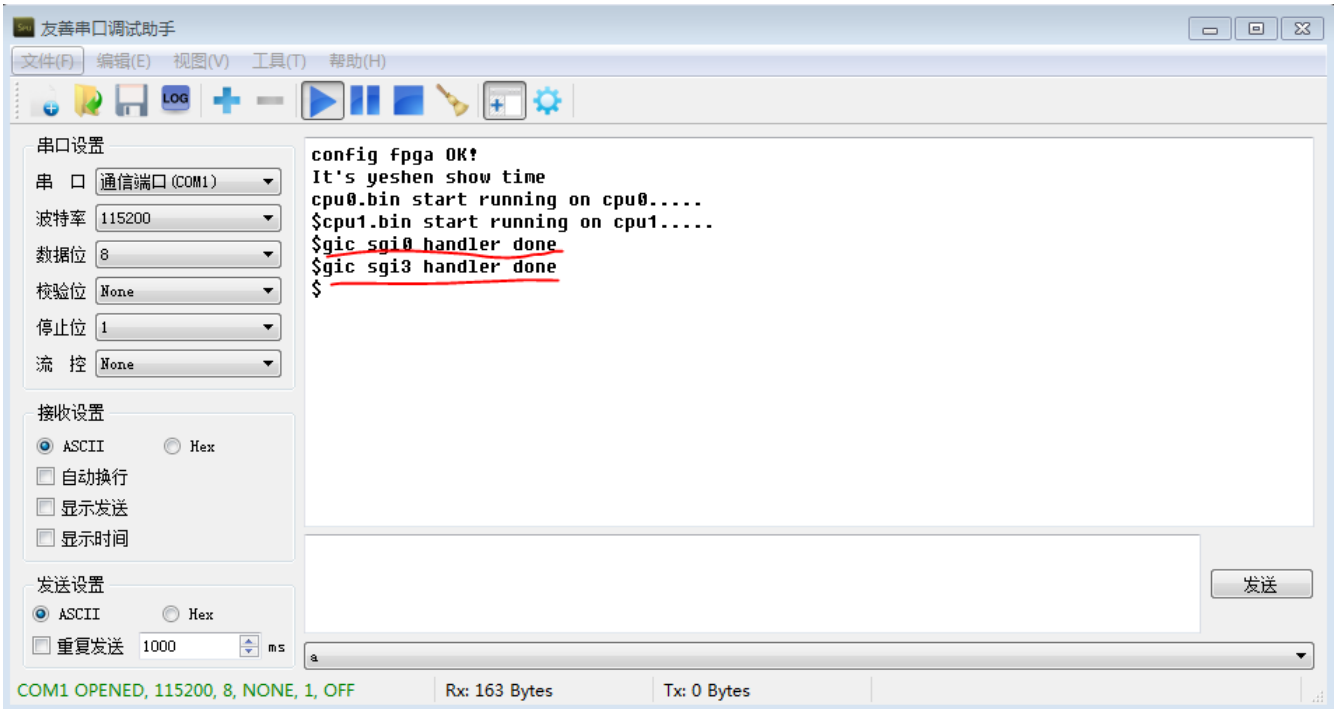
4. 嵌套中断验证

以上的测试只是在每个核中只使用了一个中断，并且在中断处理的过程不会有第二个中断到达，即没有嵌套中断，这不是一个复杂真实的系统。本节验证各种嵌套中断的现象是否与理论分析一致。

以下是模拟过程：在cpu0中设置软中断SGI0、SGI1、SGI2、SGI3的权限值分别为0xa0、0x80、0xa0、0xa8，权限值越大权限越低，因此权限从低到高为SGI3 < SGI0 = SGI2 < SGI1。注册这4个软中断的处理函数，其中SGI0的中断处理函数较为特别，在延时2秒后打印输出信息然后返回，而其他三个软中断的处理函数只打印一句输出信息就返回。以下就用SGI0表示正在处理的中断，在处理SGI0的过程分别发出SGI3 、SGI2 、SGI1、SGI0来分别代表权限比它低、相等、比它高和相同中断的四种嵌套抢占情况。

4.1 中断处理中来了一个更低权限的中断

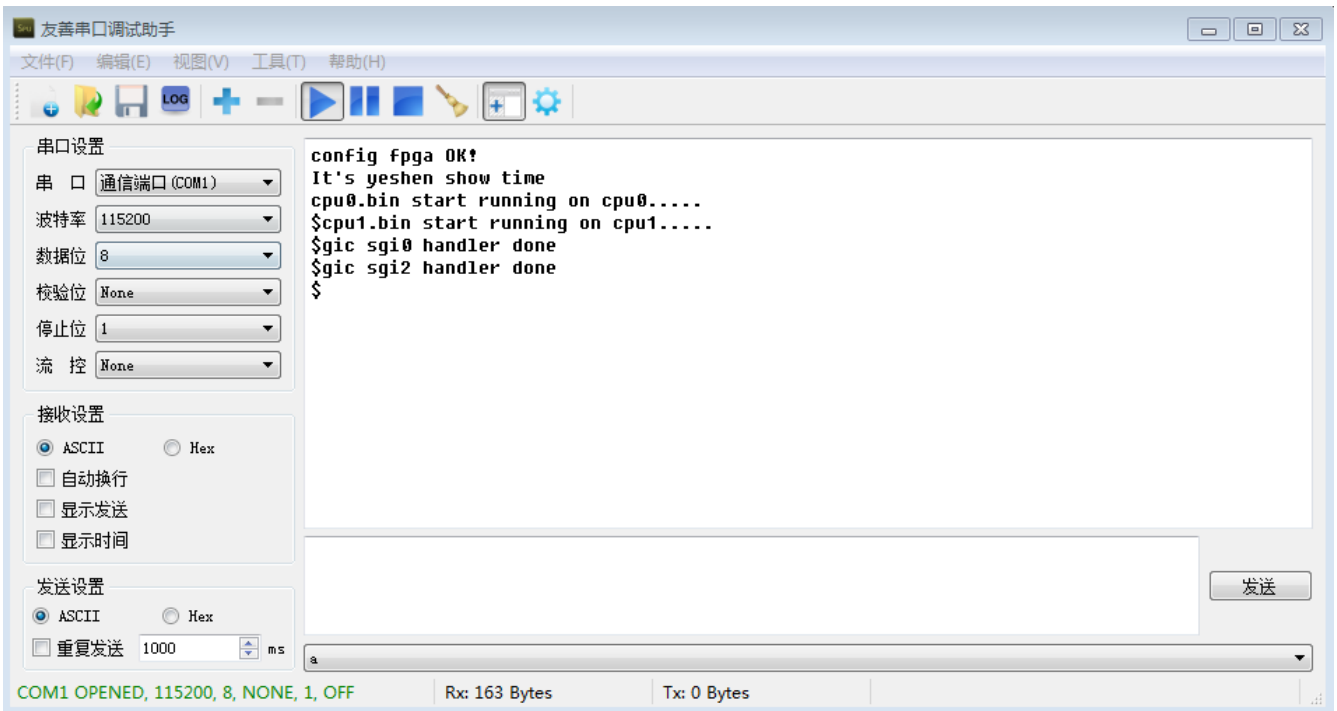
首先在cpu1中触发SGI0给cpu0，然后延时500毫秒再次触发SGI3给cpu0。以下是串口的输出打印信息：



对以上串口打印信息的解释：在cpu0接收到SGI0后进入SGI0中断处理函数，即先延时两秒，在延时过程中又接收到了SGI3，但是SGI3权限比SGI0低，因此cpu0继续在SGI0中断处理函数中处理完剩下的延时然后打印gic sgi0 handler done，之后再处理SGI3打印gic sgi3 handler done。

4.2 中断处理中来了另一个相同权限的中断

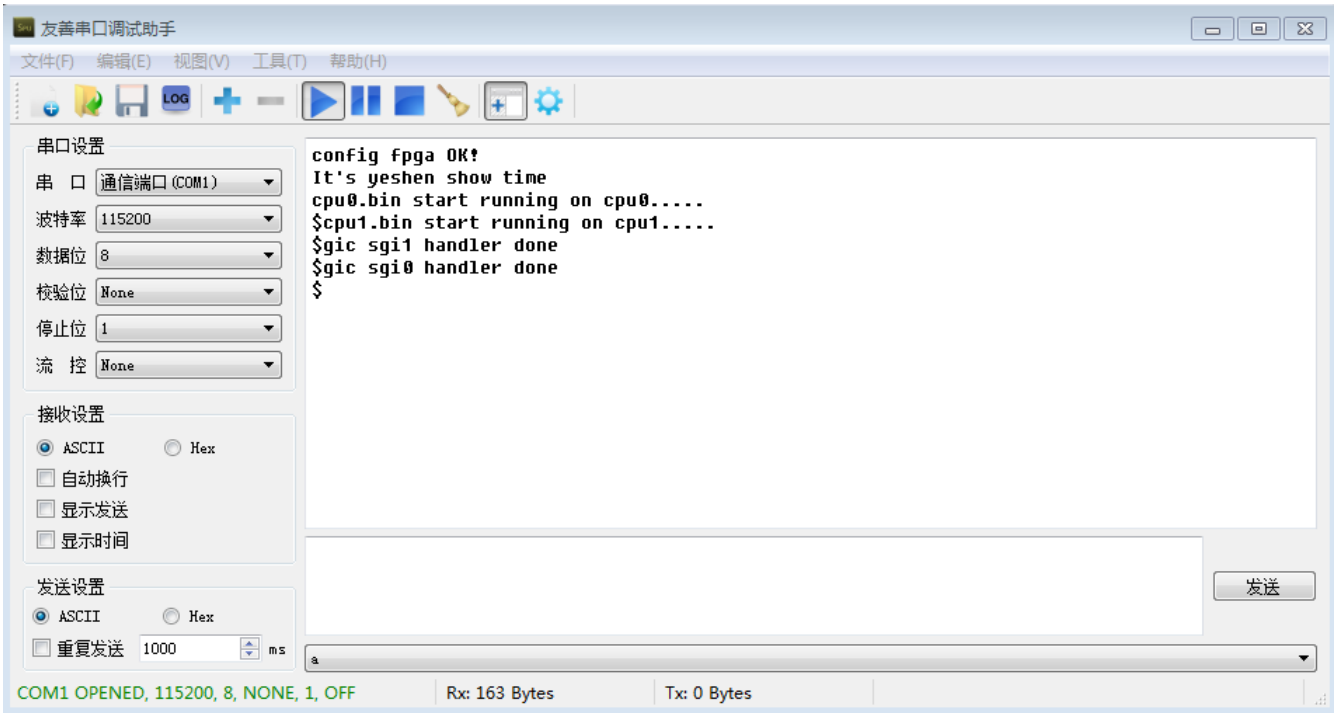
首先在cpu1中触发SGI0给cpu0，然后延时500毫秒再次触发SGI2给cpu0。以下是串口的输出打印信息：



对以上串口打印信息的解释：在cpu0接收到SGI0后进入SGI0中断处理函数，即先延时两秒，在延时过程中又接收到了SGI2，但是SGI2和SGI0相同，cpu0继续在SGI0中断处理函数中处理完剩下的延时然后打印gic sgi0 handler done，之后再处理SGI2打印gic sgi2 handler done，可见相同的权限也无法抢占。

4.3 中断处理中来了一个更高权限的中断

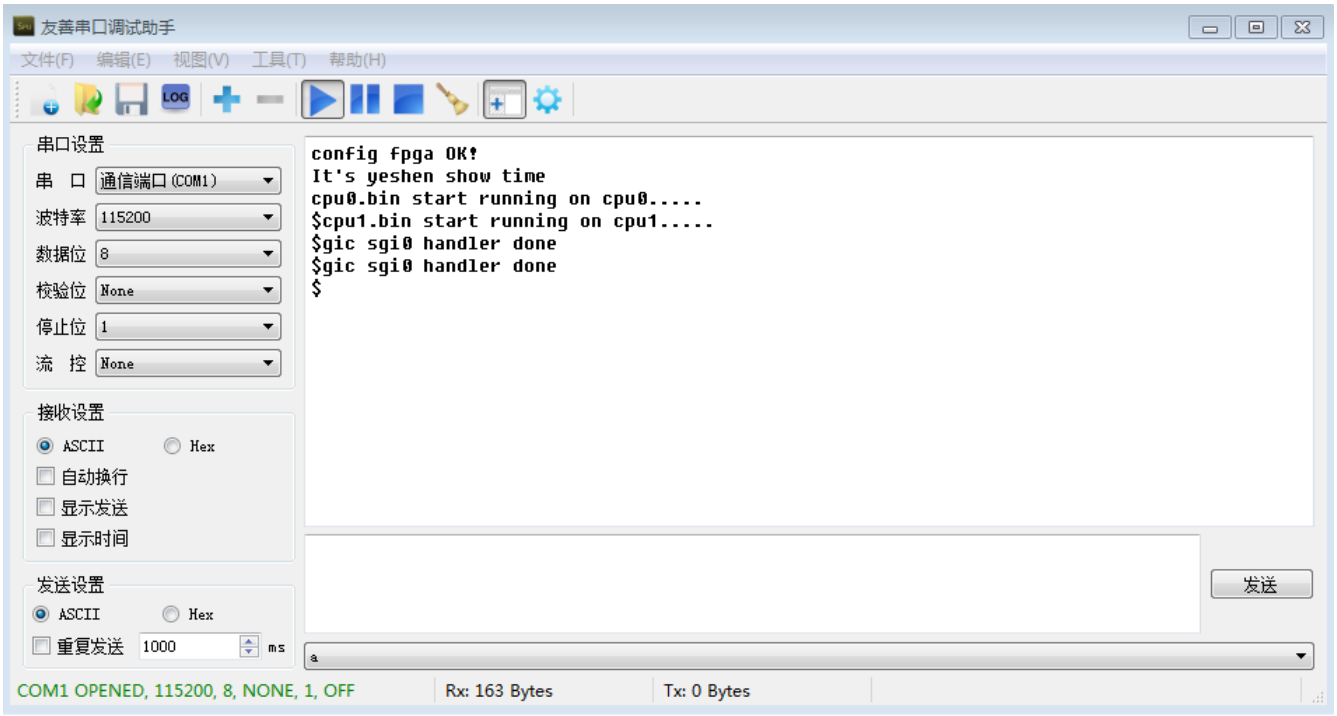
首先在cpu1中触发SGI0给cpu0，然后延时500毫秒再次触发SGI1给cpu0。以下是串口的输出打印信息：



对以上串口打印信息的解释：在cpu0接收到SGI0后进入SGI0中断处理函数，即先延时两秒，在延时过程中又接收到了SGI1，但是SGI1权限比SGI0高，因此cpu0在SGI0中断处理函数处理延时没有结束的情况下被SGI1抢占，进入SGI1中断处理函数打印gic sgi1 handler done返回继续处理SGI0中断，处理完后打印gic sgi0 handler done。

4.4 中断处理中来了多个相同中断

首先在cpu1中触发SGI0给cpu0，然后每延时200毫秒触发一次SGI0给cpu0，触发5次，即一秒钟内每隔200毫秒发送一次。以下是串口的输出打印信息：



对以上串口打印信息的解释：在cpu0接收到SGI0后进入SGI0中断处理函数，即先延时两秒，在延时过程中又接收到了多个SGI0，但是从输出信息中只看到了两次中断处理，这是因为同种类型中断在任意时刻最多只能有一个处于活动(active)状态，即正在处理，另外最多只能有一个处于悬挂(pending)状态，即在排队等待处理，之后到达的都会被丢弃，这正符合图中的输出信息。

4.5 小结

在开启嵌套中断后，中断只能被比自己权限高的中断抢断，在任意时刻最多只有一个同类型的中断在处理，另外最多只有一个同类型的中断在排队，之后到达的会丢弃。