1. 搭建CCS工程包含IPC组件后报错问题（ipc.cfg.xs中SR0Mem.base报错未定义）

必须设置platform = ti.platforms.evmDRA7XX:dsp1

1. res\_table映射关系  
   2.1

http://e2e.ti.com/support/processors/f/791/p/834599/3088238?tisearch=e2e-sitesearch&keymatch=TYPE\_CARVEOUT#3088238

2.2   
[http://e2e.ti.com/support/processors/f/791/p/804479/2977095?tisearch=e2e-sitesearch&keymatch=TYPE\_CARVEOUT#2977095](http://e2e.ti.com/support/processors/f/791/p/804479/2977095?tisearch=e2e-sitesearch&keymatch=TYPE_CARVEOUT" \l "2977095)

1. 对于这2个DEVMEM有疑问？

{

TYPE\_DEVMEM,

DSP\_MEM\_IOBUFS, PHYS\_MEM\_IOBUFS,

DSP\_MEM\_IOBUFS\_SIZE, 0, 0, "DSP\_MEM\_IOBUFS",

},

{

TYPE\_DEVMEM,

DSP\_SR0\_VIRT, DSP\_SR0,

DSP\_SR0\_SIZE, 0, 0, "DSP\_SR0",

},

PHYS\_MEM\_IOBUFS为共享内存，DSP\_SR0为BIOS核IPC用

1. SR\_0区域需要在linux设备树文件中体现。SR\_0是不是仅仅在DSP、IPU核通信是才需要？

对

/\* Shared region definition used in app \*/

var evmDRA7XX\_SR\_0 = {

name: "SR\_0", space: "data", access: "RWX",

base: 0xBFC00000, len: 0x100000,

comment: "SR#0 Memory (1 MB)"

};

在linux设备树中定义

/\* Reserved 2M as ShareRegion for Heapbuf communication! \*/

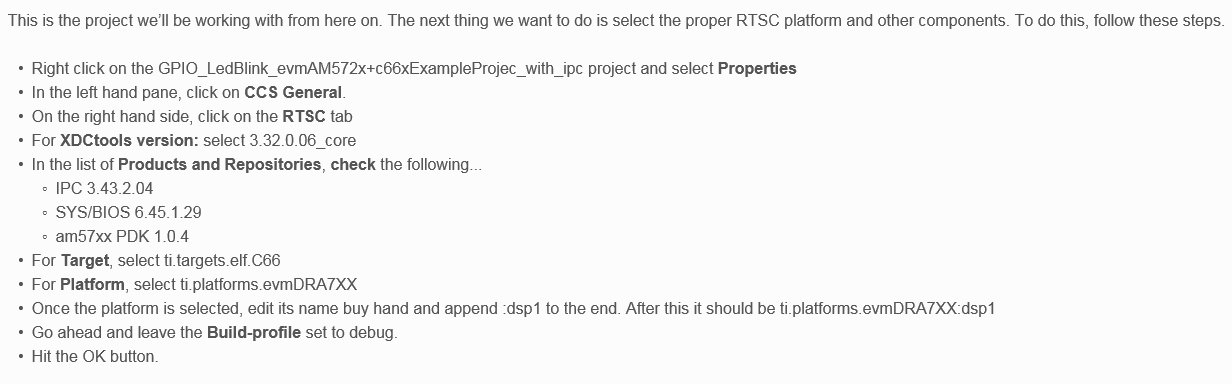
shareRegion\_0: shareRegion@bfc00000 {

reg =<0x0 0xBFC00000 0x0 0x200000>;

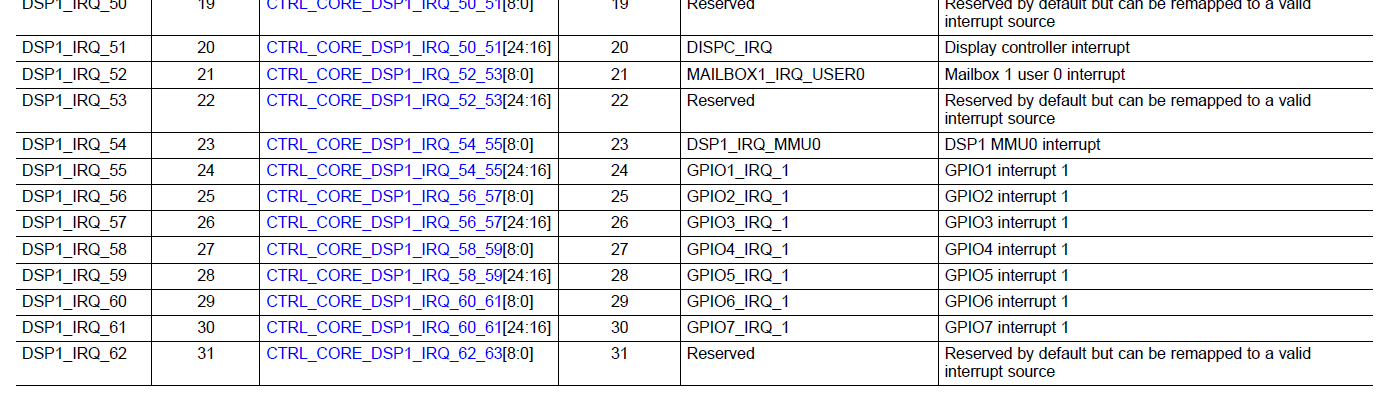
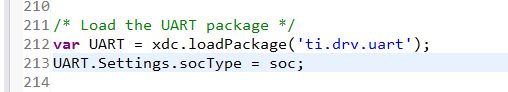
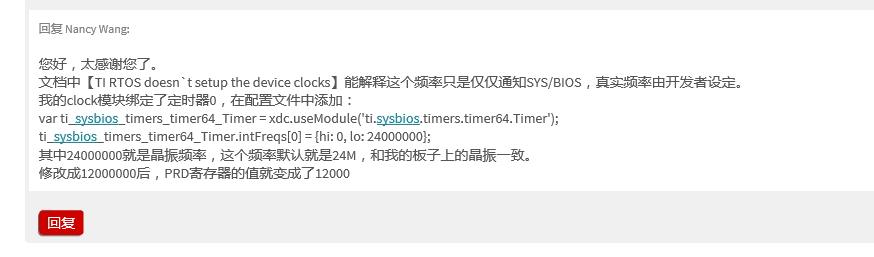
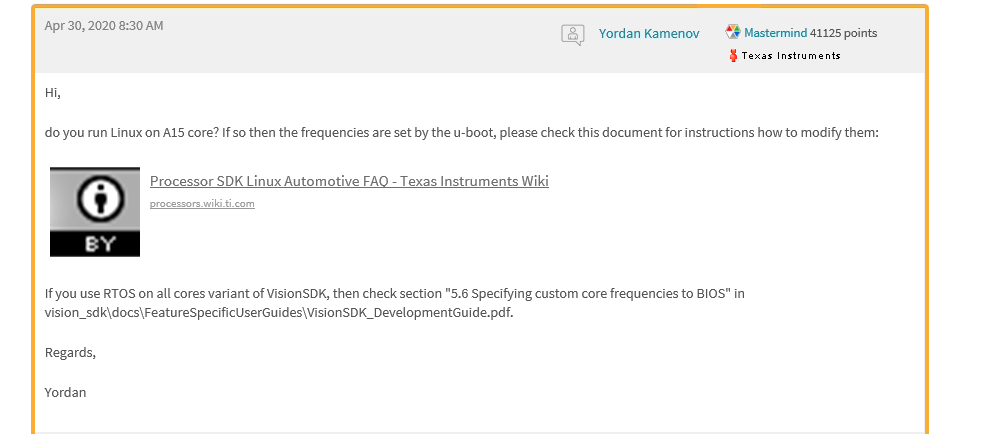
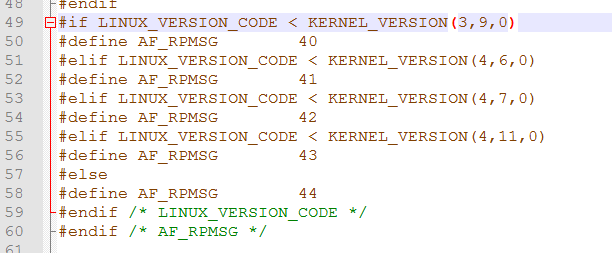
no-map;

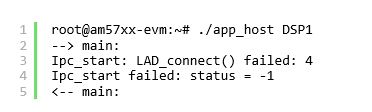
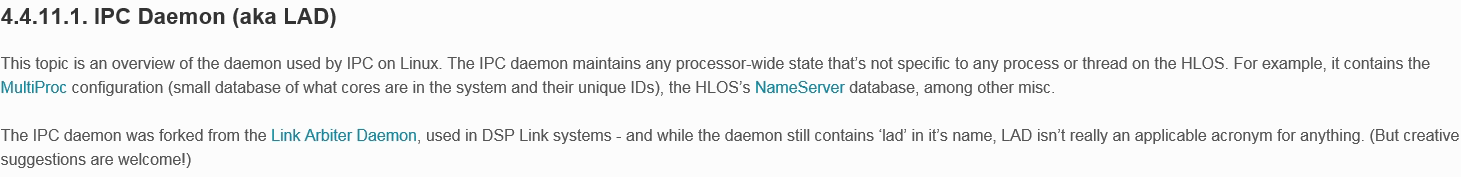
status = "okay";

};

1. All the sections defined in resource table as 'TYPE\_CARVEOUT' will be from CMA region. And from the memory map you can find the actual used memory size, then define the CMA size as needed.
2. DSP\_MEM\_IPC\_VRING and DSP\_MEM\_IPC\_DATA should be kept in rsc\_table.h. The VRING (VirtualIO ring) buffer and shared data section are essential for inter-core communication.
3. cfg后缀名结尾的文件中。在编译程序时，XDCtools解析配置文件生成相应的C源码、C头文件和cmd文件。这些文件和应用程序的其他文件一起编译和链接进最后的可执行文件。
4. Update Linux Kernel device tree to remove UART that will be controlled by M4
5. 750M  
   
6. cat /sys/kernel/debug/remoteproc/remoteproc2/resource\_table  
   root@am57xx-evm:~# cat /sys/class/remoteproc/remoteproc\*/state
7. cat /sys/kernel/debug/omap\_iommu/40d01000.mmu/pagetable (DSP1 MMU0)  
   cat /sys/kernel/debug/omap\_iommu/40d02000.mmu/pagetable (DSP1 MMU1)
8.  Add IPC to a existing example
9. IPC Install Guide Linux

[https://processors.wiki.ti.com/index.php/IPC\_Install\_Guide\_Linux](javascript:;)

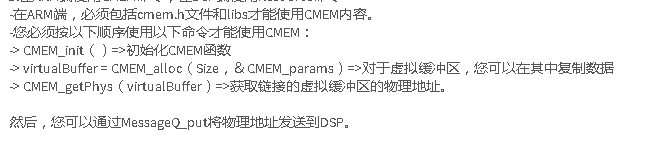
1. cat /proc/cmem 查看cmem
2. system\_image\_burn –h 查询帮助  
   system\_image\_burn -f -t UPDATE\_KERNEL -i /mnt/firmware/
3. USB挂载  
   /mnt/firmware 是Windows上的update  
   /mnt/storage 是static  
   /mnt/firmware\_dsp 是update\_dsp
4. 阻塞在Cache\_wait()问题，Linux恢复默认设备树文件后正常，主要改了DSP的映射地址。后续还要排查。
5. 每个核的硬件中断映射见Technical Reference Manual
6. IPC默认开启了watchdog，要关掉。
7. IPC API 通信顺序  
   A.HOST端发，SLAVE端收(SLAVE端创建slave pipeline)：  
   (HOST)---------->open---->alloc(msg)--->put(msg)---------------->close  
    **| | |**   
   (SLAVE)-->create----------------------------->get(msg)-->free(msg)----->delete  
   B.SLAVE端发，HOST端收（HOST端创建host pipeline）  
   (SLAVE)---------->open---->alloc(msg)--->put(msg)---------------->close  
    **| | |**   
   (HOST)-->create----------------------------->get(msg)-->free(msg)----->delete  
   NOTE:一个pipeline只能单向传输，不能又收又发。
8. Pinmux见创龙文档《AM5728设备树pinmux配置》。  
   查看原理图，跟据引脚第一配置模式名在TRM文档的Control Module中搜索，得到该引脚配置pinmux的寄存器地址。
9. C:\ti\pdk\_dra7xx\_1\_0\_10\packages里面脚本可以直接生成所需要工程。
10. IPC组件 DSP端、Host端都可以直接编译，无SDK的依赖。Host需要按实际情况修改products.mak，然后make就可生成Host端的可执行程序。
11. Linux 搭建交叉编译环境  
    A. /etc/profile文件中增加export PATH=$PATH:/home/tronlong/ti-processor-sdk-linux-rt-am57xx-evm-04.03.00.05/linuxdevkit/sysroots/x86\_64-arago-linux/usr/bin/  
    B. 修改IPC ex02例程下products.mak文件。
12. remoteproc/rpmsg driver from ARM Linux kernel communicates with IPC driver on slave processor (e.g. DSP, IPU, etc) running RTOS
13. 内部集成RTC，见TRM 23.3 RTC Integration。
14. GPIO中断  
    每个GPIO组有两个中断并不意味着只能使用两个GPIO引脚。所有GPIO引脚都可以用作中断输入，但每个GPIO组仅将它们作为两个中断请求映射给core。ISR软件必须检测中断来自哪个GPIO引脚。
15. UART中断模式  
    A. Padconfig，指定引脚工作在UART模式。  
    B. 设置xbar。  
    C. 绑定的中断号，在UART\_Config中体现，该结构体可以用这两个接口获取和设置。  
    UART\_socGetInitCfg、 UART\_socSetInitCfg
16. RTOS SDK 的驱动架构  
    A. .cfg文件中配置相关驱动库的引用。  
    B. 然后编译过程中会链接驱动库已经预编译生成的lib  
    Linking with library ti.drv.uart:./lib/dra72x/c66/release/ti.drv.uart.ae66  
    C. 改动了库代码显然要重新编译生成新的ti.drv.uart.ae66库。
17. 设备树.dtb文件在/boot 目录下。  
    cat /proc/cmem 可以查看共享内存区域情况
18. A15 linuxOS上查看DSP工作频率。  
    /sys/kernel/debug/clk# cat clk\_summary | grep dsp
19. DSP相关时钟  
    BIOS的时钟源是可配置的，C6747默认选择的tick是40号中断，即T64P1\_TINT12。 请参考BIOS默认配置的HWI\_INT15的中断号。  
    timer的时钟源是AUXCLK, 即输入时钟。  
    board clock是芯片输入时钟。一定要对应板子的输入时钟频率，不对的话，后面的tick就不对了。CPU时钟是运行时钟，可配置，需要自己在初始化代码里将芯片配置到这个频率。
20. DSP的工作频率在ARM端uboot里面设置。
21. IPC守护进程  
    linux 内核源码 /include/linux/socket.h中 #define AF\_RPMSG 45  
    IPC 守护进程源码对应定义不一致，ipc\_3\_47\_01\_00\linux\include\net\rpmsg.h中定义的AF\_RPMSG不是45。(该问题应该是RTOS SDK与LINUX SDK版本不一致)
22. IPC守护进程

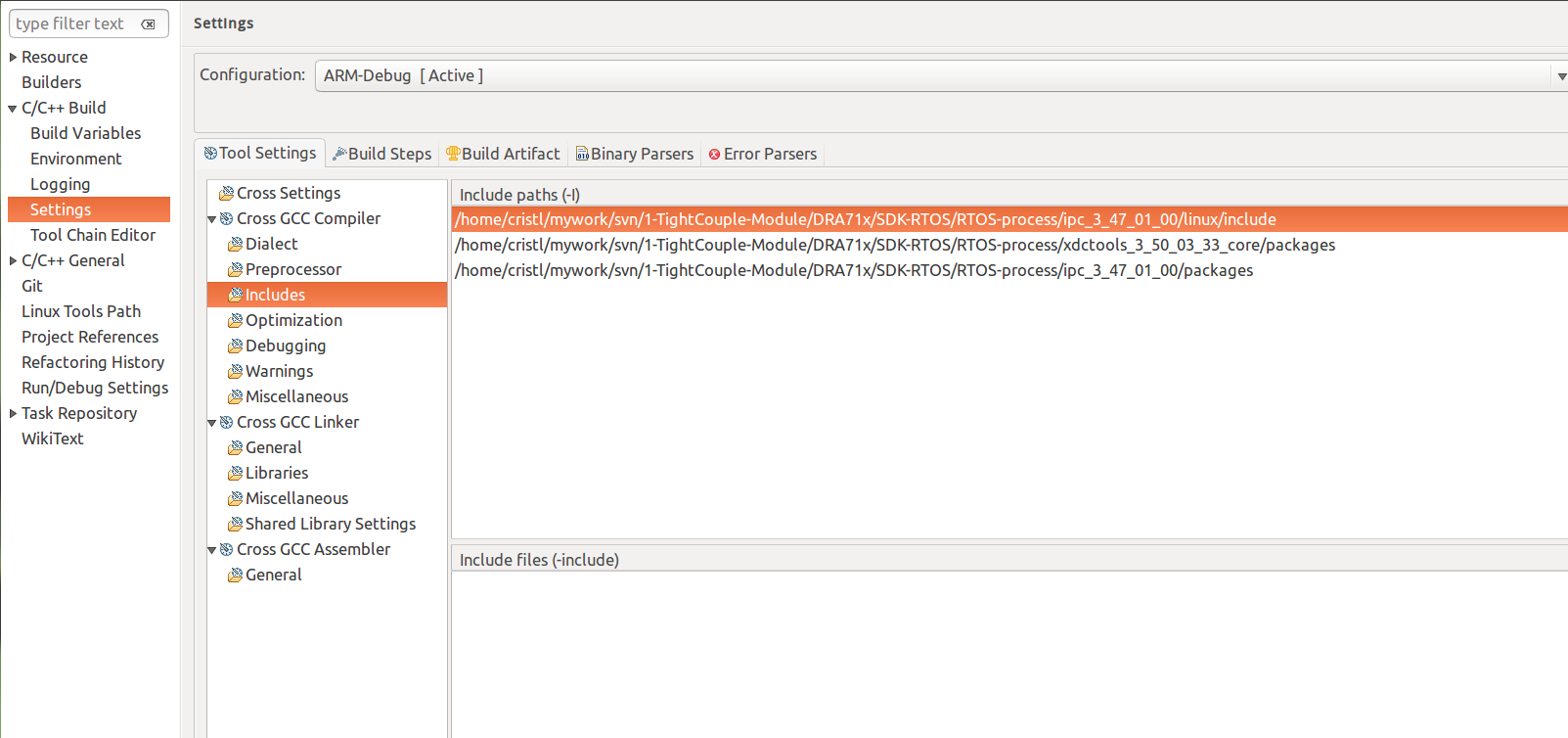
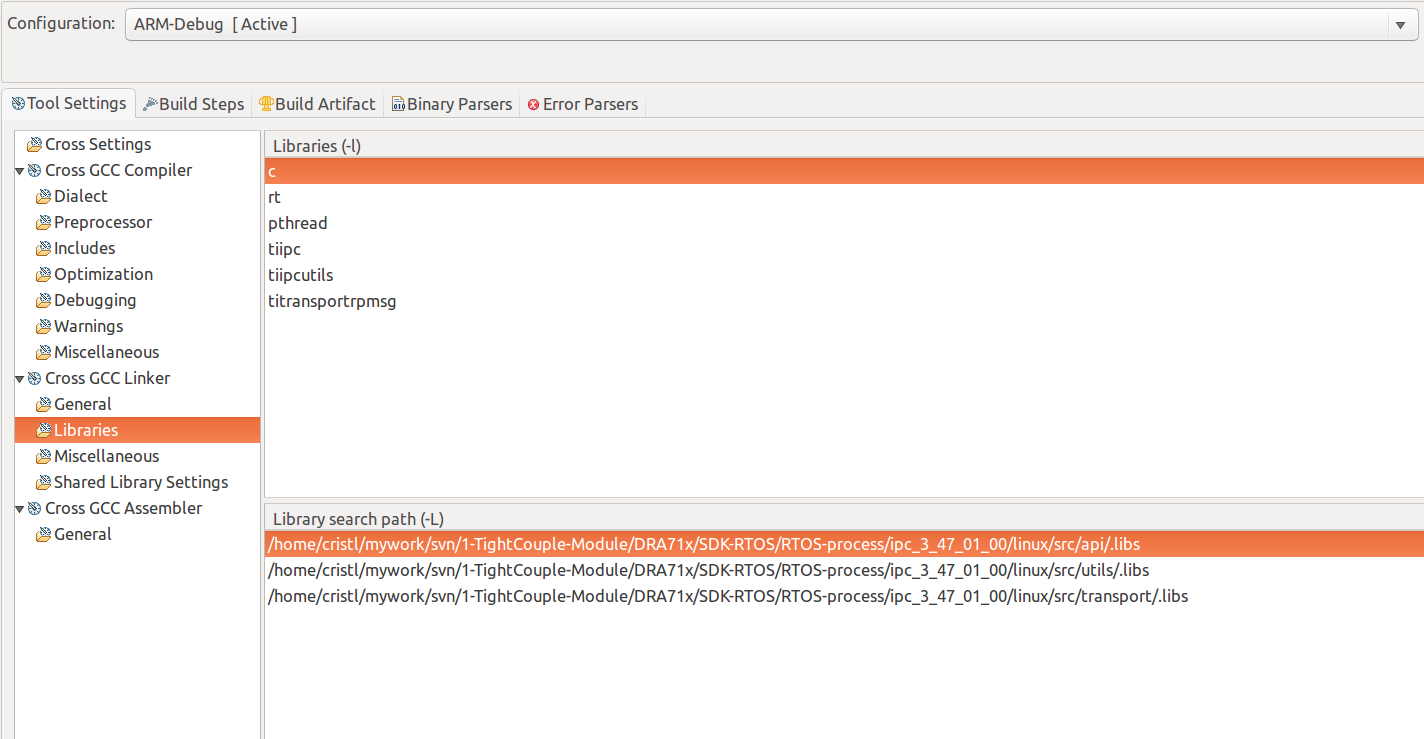
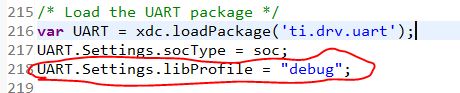
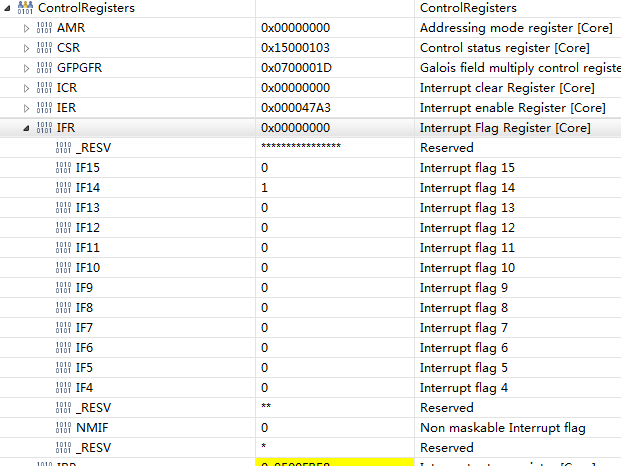
守护进程版本和IPC要一致，Rpmsg驱动必须有。

1. **Remoteproc system**  
   There is a remoteproc subsystem on linux side to control the remote processors(such as IPU1, IPU2 and DSP) to power on, load firmware, or power off and so on.
2. The physical location where the DSP code/data will actually reside is defined by the CMA carveout  
   B. Once you’ve created your custom resource table, you must update the address of PHYS\_MEM\_IPC\_VRING to be the same base address as your corresponding CMA  
   C. DSP\_PERIPHERAL\_L4CFG (0x4A000000) is the virtual address while L4\_PERIPHERAL\_L4CFG (0x4A000000) is the physical address. **This is an identity mapping, meaning that peripherals can be referenced by the DSP using their physical address.**
3. For example, the argument of 55020000.ipu corresponds to IPU2 as can be seen from dra7.dtsi.  
   ipu2: ipu@55020000 {

compatible = "ti,dra7-rproc-ipu";

1. Drive conflict  
   To prevent the conflict, it is recommanded to ensure that there is only one driver to serve this devies.
2. Resource\_table  
   A section named “.resource\_table” should be specified at firmware ELF file. The remoteproc subsystem will parse a resource table out of the firmware ELF file. This resource table tells the kernel about the resources in the remote cores subsystem such as memories, interrupt configuration, and vrings. After that, the remoteproc subsystem will configures all of the resources that are being requested by the firmwares.
3. 安装**Linux**版本**RTOS Processor-SDK-04.03**由于RTOS安装包下的cgt6x\_x86\_installer等组件为32 bit工具，因此需要在64 bit的Ubuntu下安装32 bit依赖库，否则会导致cgt6x\_x86\_installer等组件安装失败，最后无法正常编译RTOS工程。  
   **Host#** sudo apt-get install libc6:i386
4. CMEM  
   由A15发来共享内存的地址（物理地址），需要Resource\_physToVirt转换。

[https://e2e.ti.com/support/processors/f/791/p/503515/1831693?tisearch=e2e-quicksearch&keymatch=cmem%20%20physaddr#1831693](https://e2e.ti.com/support/processors/f/791/p/503515/1831693?tisearch=e2e-quicksearch&keymatch=cmem%20%20physaddr" \l "1831693)

1. 利用.obj文件隐藏代码  
   删除源码；保留.obj；在CCS的linker选项中-l选项增加指定.obj文件。  
   直接把.obj文件粘贴在工程目录下就可以，CCS会自动寻找链接所有.obj文件。
2. 在Linux端搭建IPC例程环境。  
   a.装Eclipse，需要依赖Java虚拟机支持。  
   b.配置Eclipse编译环境。  
     
     
   c.需要配置编译IPC的库，其中依赖linux内核。  
    先在products.mak文件中设置环境；然后调用configure会生成makefile文件。参考doc文件夹里的安装文件。
3. Windows下编译RTOS SDK  
   A.安装make-3.81.exe，window下的make工具。  
   B.进入processor\_sdk\_rtos\_dra7xx\_4\_03\_00\_05目录下，运行setupenv.bat。  
   C.设置make-3.81.exe的环境变量。  
   set path=C:\Program Files (x86)\GnuWin32\bin;%path%  
   set XDCTOOLS\_JAVA\_HOME=c:\ti\ccsv7\eclipse\jre  
   D.不退出CMD，进入pdk\_dra7xx\_1\_0\_10\packages目录，执行make uart命令。  
   默认编译release，在pdk\_dra7xx\_1\_0\_10\packages\ti\build中的Rules.make文件中改debug。然后再.cfg文件中该引用选项。
4. 更新文件系统命令  
   system\_image\_burn -f -t UPDATE\_CRAMFS -i /mnt/firmware/  
   system\_image\_burn -h可以查询命令
5. 中断标志寄存器 IFR
6. 设置测试点  
   **while((volatile int)flag);**
7. 默认堆**Memory.defaultHeapInstance = HeapMem.create(heapMemParams);**

直接调用malloc（）则在该堆上分配。

1. 中断相关  
   ./omapconf dump crossbar irq dsp1  
   CSL\_XBAR\_INST\_DSP1\_IRQ\_48 为index48，CSL\_XBAR\_UART2\_IRQ为68，也是写在0x4A002968的值为68。
2. DSP的timer也需要在linux的设备树开启。
3. 高温重启不来，和晶振有关，无源晶振换有源晶振后解决。
4. you must update the address of PHYS\_MEM\_IPC\_VRING to be the same base address as your corresponding CMA in linux .dts file.
5. 查看DSP1的MMU0、MMU1  
   cat /sys/kernel/debug/omap\_iommu/40d01000.mmu/pagetable  
   cat /sys/kernel/debug/omap\_iommu/40d02000.mmu/pagetable
6. ti.sysbios.heaps.HeapMem: line 221: out of memory: handle=0x953f1ea0, size=65536  
   var HeapMem = xdc.useModule('ti.sysbios.heaps.HeapMem'); task的栈在HeapMem分配。
7. 查看DSP中断  
   ./omapconf dump crossbar irq mpu
8. Task与Swi区别  
   Swi是运行至完成的非阻塞线程，只能被更高优先级的Swi或Hwi（中断）抢占。Swi可以在发布并在ISR（系统）堆栈上运行时抢占Task线程（即，它们没有自己的堆栈）。

Task，任务线程是阻塞线程，可以被更高优先级的任务或Swi / Hwi抢占。任务通常有一个while循环，只要应用程序需要它，就可以使任务在系统中连续执行。同样，任务在它们自己的独立堆栈上运行。

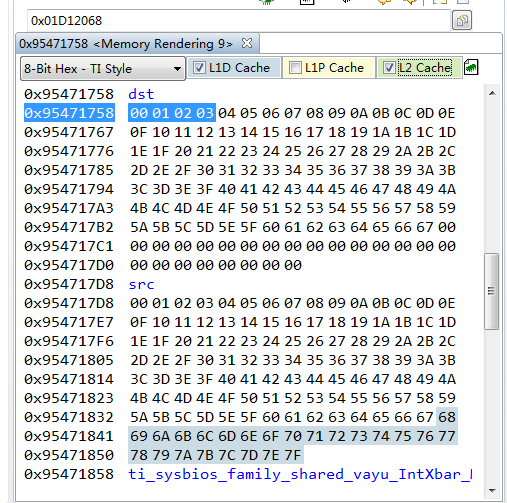
1. 由于EDMA是运行在DSP的1/3分频的频率下的，即在197MHZ的频率下，所以EDMA在数据搬运上是慢于CPU直接进行数据的拷贝的，但是，EDMA在数据搬运中是不需要CPU干涉的，实际上是不消耗cycle的,所以说EDMA还是大大的提高了CPU的工作效率的。  
   但是一旦目的地址是DDR就设置到DDR和SRAM一致性的问题。

Cache无效：由于EDMA是在CPU不干扰的情况下进行数据搬运，但是CPU读取数据的时候是通过Cache来读取数据的，当EDMA将目的地址设置为DDR时，CPU并不知道DDR的数据已经改变，当然也不会去更新Cache中的内容，于是当CPU读取该块DDR内容的时候仍然是Cache中原来没有更新的数据，这种情况的时候需要将原来的DDR那块数据对应的Cache置位无效，从新写入Cache的值，就可以保证DDR和Cache的一致性了。而对应的Cache无效函数是：Cache\_Invalid（）；

Cache 回写：同样DMA在搬运数据的时候还可能存在另一种情况，处理器在处理数据的时候会将数据先存放到Cache 中，此时Cache中的数据可能还没有来的急写入到内存中去，这时候DMA要向外设写入由处理器提供的数据的时候，DMA会直接从DDR中获得数据，搬运到外设，而这个数据并不是想要搬运的数据，这种情况下要调用Cache回写函数，在DMA进行数据搬运之前要将Cache中的数据回写到DDR中，保证DDR和Cache 的数据的一致性。

1. edma3\_lld\_2\_12\_05\_30C  
   默认EDMA为system EDMA，需要改。
2. EDMA test  
   //edma test

#pragma DATA\_SECTION(src, ".tt\_sect\_ddr");

**unsigned** **char** src[256]={0};  
如上对src指定data段后，EDMA传输不全，应该是cache问题，待确定  
src数组太小也EDMA搬运不成功，待确定  
上面两个问题均是cache问题，实际上EDMA均搬运成功，如下图。调用cache\_inv、cache\_wb解决。

1. System\_printf  
   默认也会输出到remoteproc/trace0中，但是不会在console中显示出。需要如下配置：  
   var System = xdc.useModule('xdc.runtime.System');

var SysStd = xdc.useModule('xdc.runtime.SysStd');

System.SupportProxy = SysStd; //output to console

1. Memory and Heap  
   **HeapMin.** Very small code footprint implementation. Supports non blocking memory allocation, but does not support freeing memory.

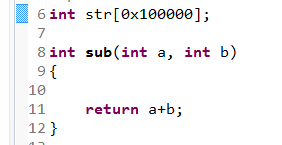
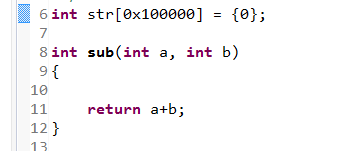
**HeapMem.** Allocate variable-size blocks and uses Gate module to protect allocation and freeing of memory. typically Slower and non-deterministic

**HeapBuf.** Allocate fixed-size blocks. Fast deterministic and non-blocking as allocation uses same size.

**HeapMultiBuf.** Specify variable-size allocation, but internally allocate from a variety of fixed-size blocks. Good tradeoff for HeapMem and HeapBuf

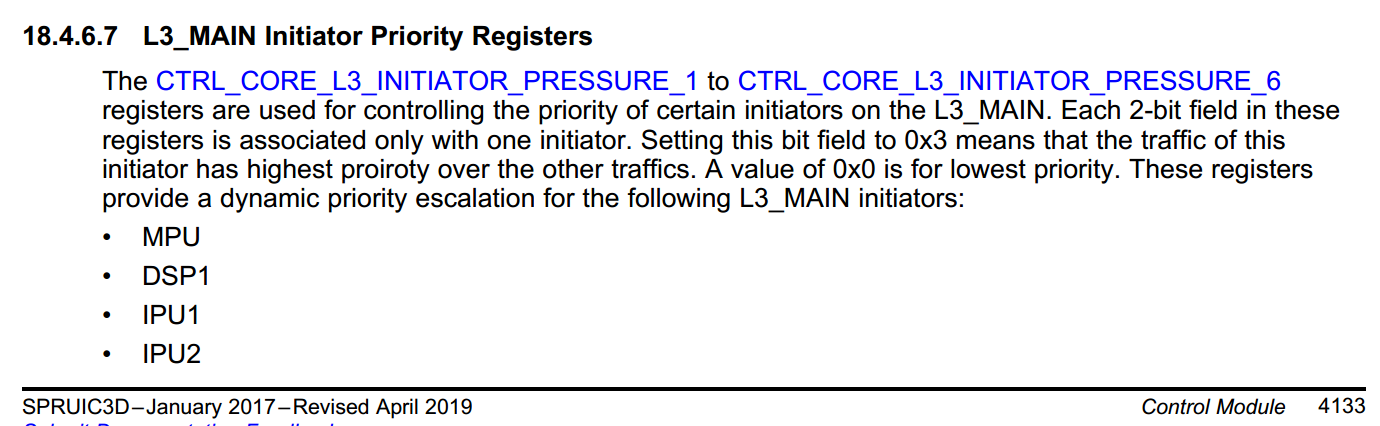
**HeapTrack.** Used to detect memory allocation and deallocation problems. Good for debugging as it detects memory leaks and buffer overflows.

1. Code段大小问题



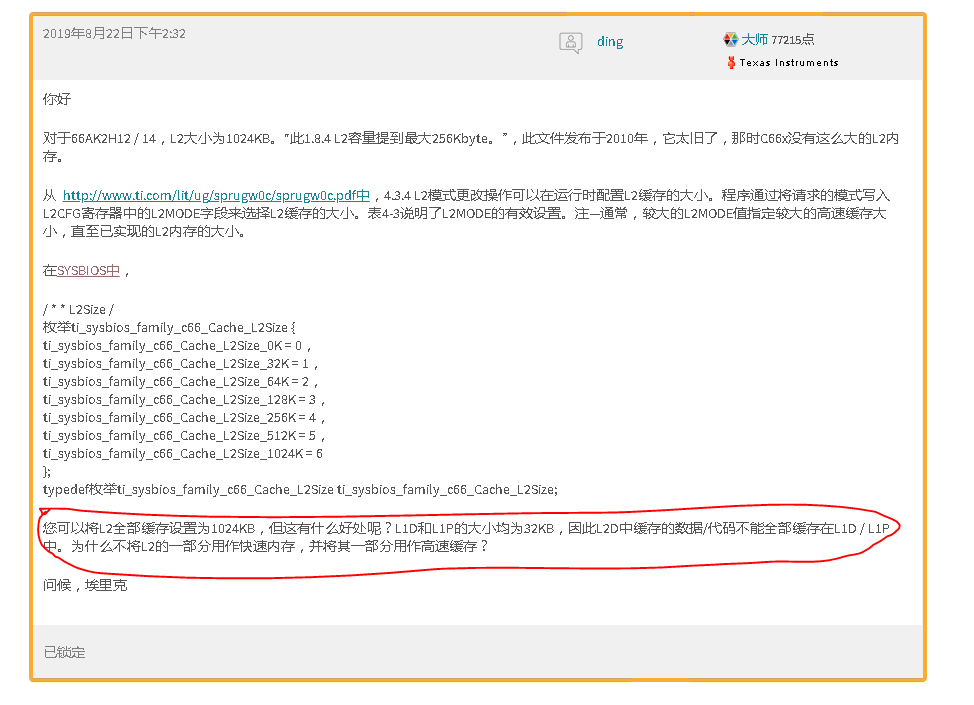
生成的lib大小，6k

生成的lib大小，4103k

1. cat /dev/ttyACM0  
   microcom -s 460800 /dev/ttyS4  
   microcom -s 460800 /dev/ttyACM0
2. 内存竞争
3. dmesg -n 4 关闭打印
4. Cache相关  
   <https://e2e.ti.com/support/processors/f/791/t/679199>

捕获  

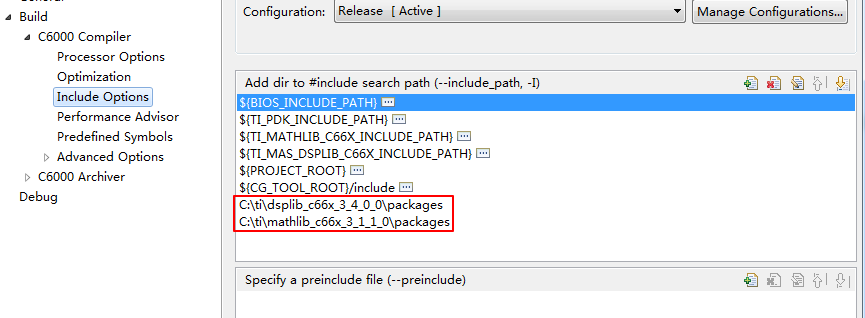


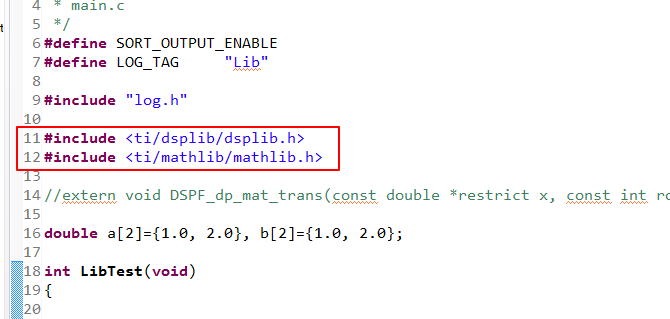

  


1. touch /mnt/platform/var/debug
2. DSPLIB  
   <https://e2e.ti.com/support/processors/f/791/t/931167?tisearch=e2e-sitesearch&keymatch=dsplib>



引用dsplib库的设置





1. CPU load配置  
   <https://e2e.ti.com/support/legacy_forums/embedded/tirtos/f/355/t/244763?tisearch=e2e-sitesearch&keymatch=Load_calculateLoad>
2. 原子操作  
   key = Hwi\_disable();  
   ......  
   Hwi\_restore(key);
3. 性能测试



1. SecureCRT与目标板传文件，借助lrz工具。
2. **Benchmark**  
   <https://e2e.ti.com/support/processors/f/791/p/918667/3419150?tisearch=e2e-sitesearch&keymatch=DSP%20benchmark%20code#pi320966=2>https://www.ti.com.cn/zh-cn/processors/digital-signal-processors/core-benchmarks/core-benchmarks.html  
   
3. killall StreamManager monitor
4. A15(Linux)  
   76.1 安装linux下SDK。  
   76.2 配置环境变量，添加交叉编译工具，编译uboot、内核、应用程序需要。（工具链存在SDK中）。
5. Sys/bios支持POSIX，需要在脚本文件中做如下配置。



1. 在ARM920T平台上int类型在对齐时读写是原子的（编译默认是对齐的），在非对齐时读写不是原子的。
2. CORS（Continuously Operating Reference Stations）就是网络基准站，通过网络收发GPS差分数据。用户访问CORS后，不用单独架设GPS基准站，即可实现GPS流动站的差分定位。访问CORS系统，就需要网络通讯协议。NTRIP（ Networked Transport of RTCM via Internet Protocol）是CORS系统的通讯协议之一。