[AM335x启动流程（BootRom->MLO->Uboot）](http://blog.csdn.net/psvoldemort/article/details/41861959)

分类： [项目-AM335x](http://blog.csdn.net/psvoldemort/article/category/2646389)2014-12-11 08:06 112人阅读 [评论](http://blog.csdn.net/psvoldemort/article/details/41861959#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/psvoldemort/article/details/41861959#report)

http://blog.chinaunix.net/uid-28458801-id-3486399.html

参考文件：

1，AM335x ARM Cortex-A8 Microprocessors (MPUs) Technical Reference Manual.pdf；

2，am3359.pdf；

**1，am335x的cpu上电后，会跳到哪个地址去执行？**

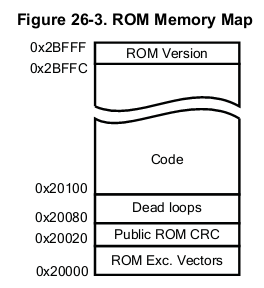
答：

芯片到uboot启动流程 ：ROM → MLO(SPL)→ uboot.img

AM335x 中bootloader被分成了 3 个部分：

**第一级 bootloader：**引导加载程序，板子上电后会自动执行这些代码，如选择哪种方式启动（NAND，SDcard，UART。。。），然后跳转转到第二级 bootloader。这些代码应该是存放在 176KB 的 ROM 中。

http://blog.chinaunix.net/attachment/201302/3/28458801_1359884692bbKj.png



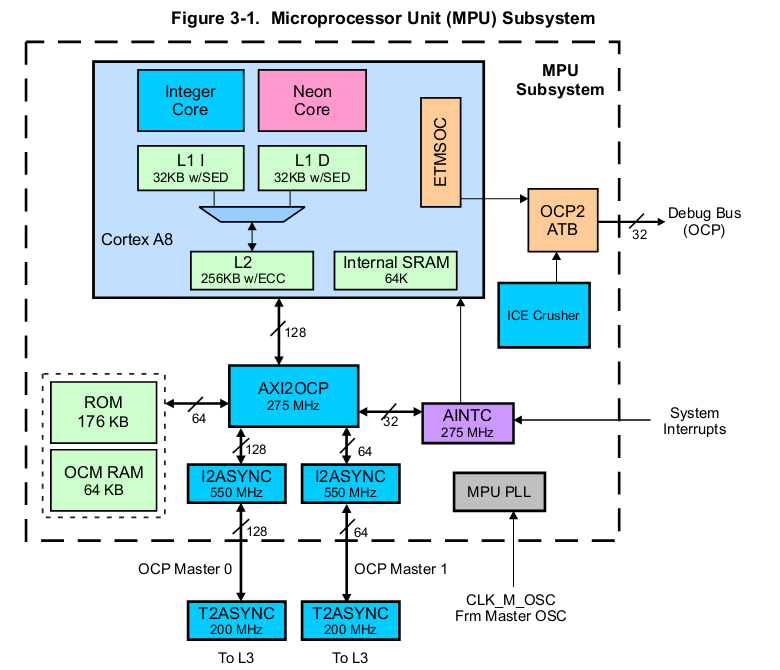
**第二级 bootloader：**MLO（SPL），用以硬件初始化：关闭看门狗，关闭中断，设置 CPU 时钟频率、速度等操作。然后会跳转到第三级bootloader。MLO文件应该会被映射到 64 KB的 Internal SRAM 中。

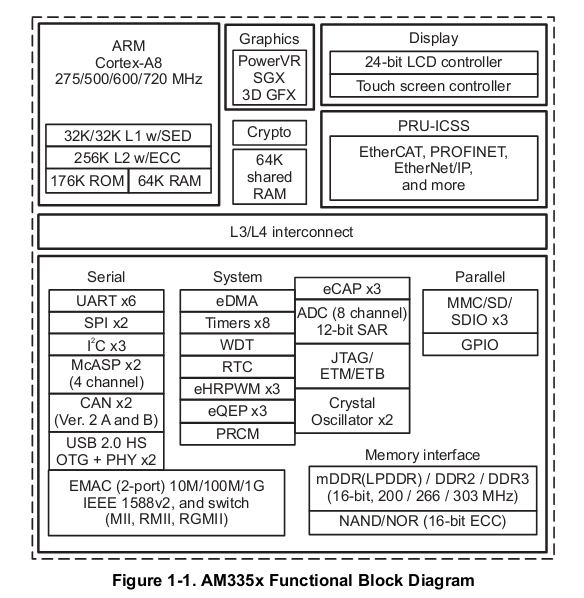
http://blog.chinaunix.net/attachment/201302/3/28458801_1359884707t6fk.png

**第三级 bootloader：**uboot.img，C代码的入口。

http://blog.chinaunix.net/attachment/201302/3/28458801_135988479506fn.png

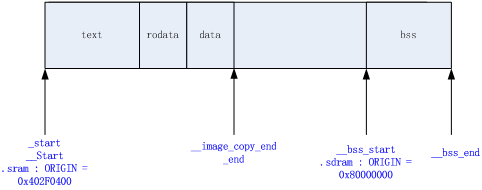
其中第一级 bootloader 是板子固化的，第二级和第三级是通过编译 uboot 所得的。





**2，第二级 bootloader：MLO（SPL）做了哪些事情？**

**MLO(SPL)内存分布如下：**

****

**SPL内存重映射：**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | < PATH : /arch/arm/cpu/armv7/omap-common/u-boot-spl.lds >  MEMORY { .sram : ORIGIN = CONFIG\_SPL\_TEXT\_BASE,\          LENGTH = CONFIG\_SPL\_MAX\_SIZE }  MEMORY { .sdram : ORIGIN = CONFIG\_SPL\_BSS\_START\_ADDR, \          LENGTH = CONFIG\_SPL\_BSS\_MAX\_SIZE }    OUTPUT\_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")  OUTPUT\_ARCH(arm)  ENTRY(\_start)  SECTIONS  {      .text      :      {      \_\_start = .;        arch/arm/cpu/armv7/start.o    (.text)        \*(.text\*)      } >.sram        . = ALIGN(4);      .rodata : { \*(SORT\_BY\_ALIGNMENT(.rodata\*)) } >.sram        . = ALIGN(4);      .data : { \*(SORT\_BY\_ALIGNMENT(.data\*)) } >.sram      . = ALIGN(4);      \_\_image\_copy\_end = .;      \_end = .;        .bss :      {          . = ALIGN(4);          \_\_bss\_start = .;          \*(.bss\*)          . = ALIGN(4);          \_\_bss\_end\_\_ = .;      } >.sdram  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <PATH : /include/configs/am335x\_evm.h>  #define CONFIG\_SPL\_TEXT\_BASE        0x402F0400  #define CONFIG\_SPL\_MAX\_SIZE     (46 \* 1024)  #define CONFIG\_SPL\_STACK        LOW\_LEVEL\_SRAM\_STACK    #define CONFIG\_SPL\_BSS\_START\_ADDR   0x80000000  #define CONFIG\_SPL\_BSS\_MAX\_SIZE     0x80000     /\* 512 KB \*/ <span style="font-size:16px;color:#003399;"><strong></strong></span> |

**@1@ 保存启动参数 bl    save\_boot\_params**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <PATH : /arch/arm/cpu/armv7/start.S>  /\*   \* the actual reset code   \*/    reset:      bl  save\_boot\_params |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <PATH : /arch/arm/cpu/armv7/omap-common/lowlevel\_init.S>  .global save\_boot\_params  save\_boot\_params:      /\*       \* See if the rom code passed pointer is valid:       \* It is not valid if it is not in non-secure SRAM       \* This may happen if you are booting with the help of       \* debugger       \*/      ldr     r2, =NON\_SECURE\_SRAM\_START      cmp r2, r0      bgt 1f      ldr r2, =NON\_SECURE\_SRAM\_END      cmp r2, r0      blt 1f        /\*       \* store the boot params passed from rom code or saved       \* and passed by SPL       \*/      cmp r0, #0      beq 1f      ldr r1, =boot\_params      str r0, [r1] |

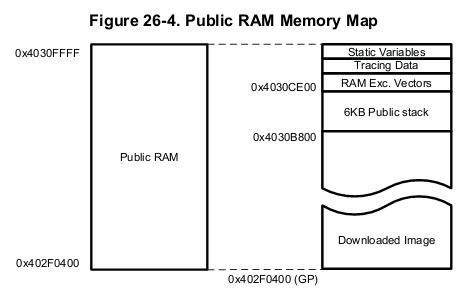
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | /\*《PATH: /arch/arm/include/asm/arch-ti81xx/omap.h》   \* Non-secure SRAM Addresses   \* Non-secure RAM starts at 0x40300000 for GP devices. But we keep SRAM\_BASE   \* at 0x40304000(EMU base) so that our code works for both EMU and GP   \*/  #define NON\_SECURE\_SRAM\_START   0x40304000  #define NON\_SECURE\_SRAM\_END 0x4030E000  #define LOW\_LEVEL\_SRAM\_STACK    0x4030B7FC |

**问题：这些参数是保存在哪里的？大概有哪些参数？**

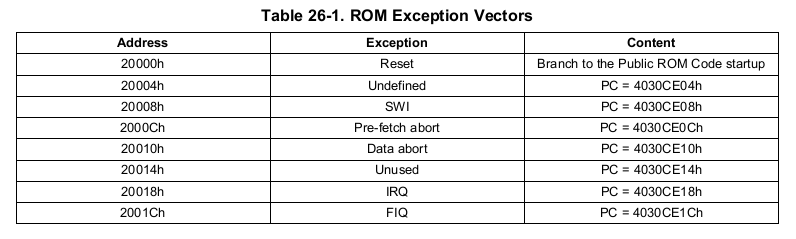
答：

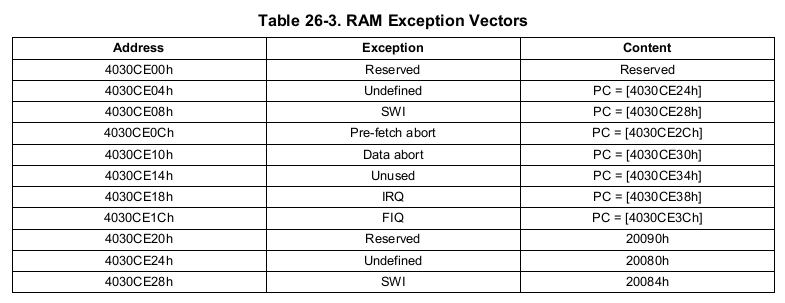
这些参数保存的内存地址为 64 KB 的 OCM RAM 中：

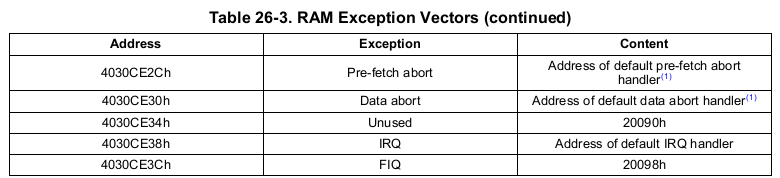
http://blog.chinaunix.net/attachment/201302/5/28458801_13600331869XZu.png

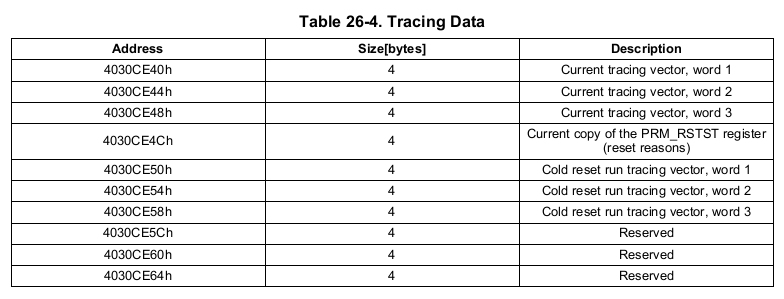


**注：Dowloaded Image 区域：是用来保存 MLO（SPL） 文件的，其最大可达到 109 KB**









**@a2@ 设置 CPU 为** **SVC32 模式**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <PATH : /arch/arm/cpu/armv7/start.S>          /\*       \* set the cpu to SVC32 mode       \*/      mrs r0, cpsr      bic r0, r0, #0x1f      orr r0, r0, #0xd3      msr cpsr,r0 |

**CPSR**:程序状态寄存器(current program status register)(当前程序状态寄存器)，在任何处理器模式下被访问。它包含了条件标志位、中断禁止位、当前处理器模式标志以及其他的一些控制和状态位。  
CPSR在用户级编程时用于存储条件码。

**SPSR:**程序状态保存寄存器（saved program statusregister）,每一种处理器模式下都有一个状态寄存器SPSR,SPSR用于保存CPSR的状态，以便异常返回后恢复异常发生时的工作状态。当特定的异常中断发生时，这个寄存器用于存放当前程序状态寄存器的内容。在异常中断退出时，可以用SPSR来恢复CPSR。由于用户模式和系统模式不是异常中断模式，所以他没有SPSR。当用户在用户模式或系统模式访问SPSR，将产生不可预知的后果。

CPSR格式如下所示。SPSR和CPSR格式相同。  
31 30 29 28 27 26 7 6 5 4 3 2 1 0  
N Z C V Q DNM(RAZ) I F T M4 M3 M2 M1 M0

**详解：[http://blog.chinaunix.net/uid-28458801-id-3487199.html](http://blog.chinaunix.net/uid-28458801-id-3487199.html" \t "_blank)**

**@a3@ CPU的初始化**

|  |  |
| --- | --- |
| 1  2  3  4  5 | 《PATH : /arch/arm/cpu/armv7/start.S》      /\* the mask ROM code should have PLL and others stable \*/  #ifndef CONFIG\_SKIP\_LOWLEVEL\_INIT      bl  cpu\_init\_crit  #endif |

#ifndef CONFIG\_SKIP\_LOWLEVEL\_INIT

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* CPU\_init\_critical registers

\*

\* setup important registers

\* setup memory timing

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ENTRY(cpu\_init\_crit)

/\*

\* Jump to board specific initialization...

\* The Mask ROM will have already initialized

\* basic memory. Go here to bump up clock rate and handle

\* wake up conditions.

\*/

b lowlevel\_init @ go setup pll,mux,memory

ENDPROC(cpu\_init\_crit)

#endif

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <PATH : /arch/arm/cpu/armv7/omap-common/lowlevel\_init.S>  .globl lowlevel\_init  lowlevel\_init:      /\*       \* Setup a temporary stack       \*/      ldr sp, =LOW\_LEVEL\_SRAM\_STACK        /\*       \* Save the old lr(passed in ip) and the current lr to stack       \*/      push    {ip, lr}        /\*       \* go setup pll, mux, memory       \*/      bl  s\_init      pop {ip, pc} |

**问题：CPU的初始化有哪些内容？**

答：

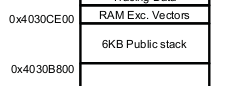
**@b1@ 首先要设置堆栈区，因为将会调用 C函数来实现CPU的初始化**

**问题：这个堆栈在什么位置，其内存大小是多少？**

答

|  |  |
| --- | --- |
| 1  2 | 《PATH ：/arch/arm/include/asm/arch-ti81xx/omap.h》  #define LOW\_LEVEL\_SRAM\_STACK    0x4030B7FC<strong></strong> |

**http://blog.chinaunix.net/attachment/201302/5/28458801_1360036744DPZ4.png**

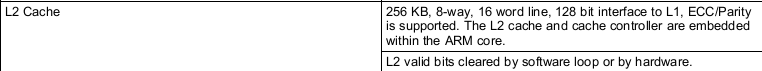


**@b2@ 执行 s\_init() 函数，实现 CPU 的初始化**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | <PATH : /board/ti/am335x/evm.c>  /\*   \* early system init of muxing and clocks.   \*/  void s\_init(void)  {      /\* Can be removed as A8 comes up with L2 enabled \*/      l2\_cache\_enable();        /\* WDT1 is already running when the bootloader gets control       \* Disable it to avoid "random" resets       \*/      \_\_raw\_writel(0xAAAA, WDT\_WSPR);      while(\_\_raw\_readl(WDT\_WWPS) != 0x0);      \_\_raw\_writel(0x5555, WDT\_WSPR);      while(\_\_raw\_readl(WDT\_WWPS) != 0x0);    #ifdef CONFIG\_SPL\_BUILD      /\* Setup the PLLs and the clocks for the peripherals \*/      pll\_init();        /\* Enable RTC32K clock \*/      rtc32k\_enable();        /\* UART softreset \*/      u32 regVal;      u32 uart\_base = DEFAULT\_UART\_BASE;        enable\_uart0\_pin\_mux();      /\* IA Motor Control Board has default console on UART3\*/      /\* XXX: This is before we've probed / set board\_id \*/      if (board\_id == IA\_BOARD) {          uart\_base = UART3\_BASE;      }        regVal = \_\_raw\_readl(uart\_base + UART\_SYSCFG\_OFFSET);      regVal |= UART\_RESET;      \_\_raw\_writel(regVal, (uart\_base + UART\_SYSCFG\_OFFSET) );      while ((\_\_raw\_readl(uart\_base + UART\_SYSSTS\_OFFSET) &              UART\_CLK\_RUNNING\_MASK) != UART\_CLK\_RUNNING\_MASK);        /\* Disable smart idle \*/      regVal = \_\_raw\_readl((uart\_base + UART\_SYSCFG\_OFFSET));      regVal |= UART\_SMART\_IDLE\_EN;      \_\_raw\_writel(regVal, (uart\_base + UART\_SYSCFG\_OFFSET));        /\* Initialize the Timer \*/      init\_timer();        preloader\_console\_init();        printf("\nlocation /board/ti/am335x\n");        //@@  /\*@@\*/  //  led();  /\*@@\*/        config\_am335x\_ddr();    #endif  } |

**@c1@ 使能第二级缓冲区**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\* Can be removed as A8 comes up with L2 enabled \*/      l2\_cache\_enable();    <PATH : /arch/arm/cpu/armv7/ti81xx/cache.S>  l2\_cache\_enable:      push    {r0, r1, r2, lr}      mrc 15, 0, r3, cr1, cr0, 1      orr r3, r3, #2      mcr 15, 0, r3, cr1, cr0, 1      pop {r1, r2, r3, pc} |

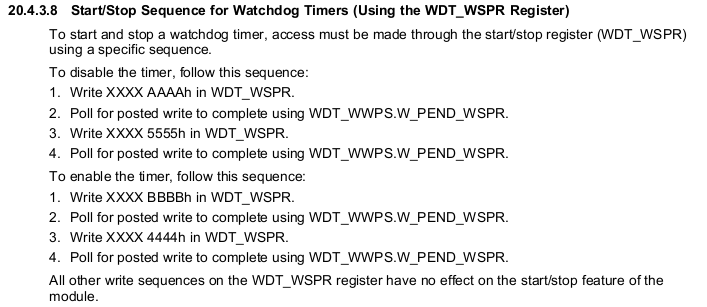


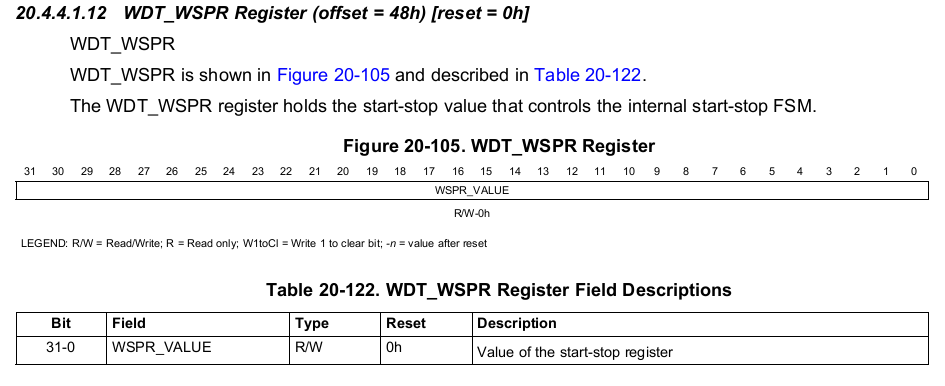
**@c2@ 关闭看门狗（WDT）**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | /\* WDT1 is already running when the bootloader gets control   \* Disable it to avoid "random" resets   \*/  \_\_raw\_writel(0xAAAA, WDT\_WSPR);  while(\_\_raw\_readl(WDT\_WWPS) != 0x0);  \_\_raw\_writel(0x5555, WDT\_WSPR);  while(\_\_raw\_readl(WDT\_WWPS) != 0x0); |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <PATH : /arch/arm/include/asm/arch-ti81xx/cpu.h>  #define WDT\_WSPR    (WDT\_BASE + 0x048)      <PATH : /arch/arm/include/asm/arch-ti81xx/hardware.h>  /\* Watchdog Timer \*/  #ifdef CONFIG\_AM335X  #define WDT\_BASE            0x44E35000  #else  #define WDT\_BASE            0x480C2000  #endif |

**http://blog.chinaunix.net/attachment/201302/5/28458801_1360050929HZ1Y.png**

****

****

**@c3@ 给外设设置好 PLL 和 时钟频率等**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | /\* Setup the PLLs and the clocks for the peripherals \*/      pll\_init();      <PATH : /board/ti/am335x/pll.c>  /\*   \* Configure the PLL/PRCM for necessary peripherals   \*/  void pll\_init()  {      mpu\_pll\_config(MPUPLL\_M\_500);      core\_pll\_config();      per\_pll\_config();      ddr\_pll\_config();      /\* Enable the required interconnect clocks \*/      interface\_clocks\_enable();      /\* Enable power domain transition \*/      power\_domain\_transition\_enable();      /\* Enable the required peripherals \*/      per\_clocks\_enable();  } |

****

**@c4@ 使能 32-KHz 频率的实时时钟**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | /\* Enable RTC32K clock \*/      rtc32k\_enable();      《PATH : /board/ti/am335x/evm.c》  static void rtc32k\_enable(void)  {      /\* Unlock the rtc's registers \*/      \_\_raw\_writel(0x83e70b13, (AM335X\_RTC\_BASE + RTC\_KICK0\_REG));      \_\_raw\_writel(0x95a4f1e0, (AM335X\_RTC\_BASE + RTC\_KICK1\_REG));        /\* Enable the RTC 32K OSC \*/      \_\_raw\_writel(0x48, (AM335X\_RTC\_BASE + RTC\_OSC\_REG));  }    <PATH : /arch/arm/include/asm/arch-ti81xx/hardware.h>  /\* RTC base address \*/  #define AM335X\_RTC\_BASE            0x44E3E000    <PATH : /board/ti/am335x/evm.c>  #define RTC\_KICK0\_REG        0x6c  #define RTC\_KICK1\_REG        0x70  #define RTC\_OSC\_REG        0x54 |

**http://blog.chinaunix.net/attachment/201302/5/28458801_1360054158I10F.png**

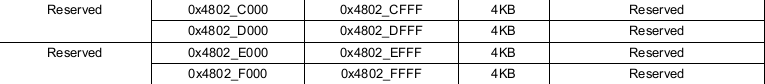
**@c5@ 使能UART0**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | /\* UART softreset \*/      u32 regVal;      u32 uart\_base = DEFAULT\_UART\_BASE;        enable\_uart0\_pin\_mux();      /\* IA Motor Control Board has default console on UART3\*/      /\* XXX: This is before we've probed / set board\_id \*/      if (board\_id == IA\_BOARD) {          uart\_base = UART3\_BASE;      }        regVal = \_\_raw\_readl(uart\_base + UART\_SYSCFG\_OFFSET);      regVal |= UART\_RESET;      \_\_raw\_writel(regVal, (uart\_base + UART\_SYSCFG\_OFFSET) );      while ((\_\_raw\_readl(uart\_base + UART\_SYSSTS\_OFFSET) &              UART\_CLK\_RUNNING\_MASK) != UART\_CLK\_RUNNING\_MASK);        /\* Disable smart idle \*/      regVal = \_\_raw\_readl((uart\_base + UART\_SYSCFG\_OFFSET));      regVal |= UART\_SMART\_IDLE\_EN;      \_\_raw\_writel(regVal, (uart\_base + UART\_SYSCFG\_OFFSET));      <PATH : /arch/arm/include/asm/arch-ti81xx/cpu.h>  #ifdef CONFIG\_AM335X  #define DEFAULT\_UART\_BASE       UART0\_BASE  #endif    <PATH : /arch/arm/include/asm/arch-ti81xx/hardware.h>  #ifdef CONFIG\_AM335X  #define UART0\_BASE          0x44E09000  #else  #define UART0\_BASE          0x48020000  #endif |

http://blog.chinaunix.net/attachment/201302/5/28458801_1360054569U66b.png

**@c6@ 初始化 定时器**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | /\* Initialize the Timer \*/      init\_timer();      <PATH : /board/ti/am335x/evm.c>  static void init\_timer(void)  {      /\* Reset the Timer \*/      \_\_raw\_writel(0x2, (DM\_TIMER2\_BASE + TSICR\_REG));        /\* Wait until the reset is done \*/      while (\_\_raw\_readl(DM\_TIMER2\_BASE + TIOCP\_CFG\_REG) & 1);        /\* Start the Timer \*/      \_\_raw\_writel(0x1, (DM\_TIMER2\_BASE + TCLR\_REG));  }    <PATH : /arch/arm/include/asm/arch-ti81xx/hardware.h>  /\* DM Timer base addresses \*/  #define DM\_TIMER0\_BASE          0x4802C000  #define DM\_TIMER1\_BASE          0x4802E000  #define DM\_TIMER2\_BASE          0x48040000  #define DM\_TIMER3\_BASE          0x48042000  #define DM\_TIMER4\_BASE          0x48044000  #define DM\_TIMER5\_BASE          0x48046000  #define DM\_TIMER6\_BASE          0x48048000  #define DM\_TIMER7\_BASE          0x4804A000 |





**@c7@ 初始化控制台，通过UART可以查看相关信息**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | preloader\_console\_init();    《PATH : /arch/arm/cpu/armv7/omap-common/spl.c》  /\* This requires UART clocks to be enabled \*/  void preloader\_console\_init(void)  {      const char \*u\_boot\_rev = U\_BOOT\_VERSION;      char rev\_string\_buffer[50];        gd = &gdata;      gd->bd = &bdata;      gd->flags |= GD\_FLG\_RELOC;      gd->baudrate = CONFIG\_BAUDRATE;        serial\_init();      /\* serial communications setup \*/        /\* Avoid a second "U-Boot" coming from this string \*/      u\_boot\_rev = &u\_boot\_rev[7];        printf("\nU-Boot SPL %s (%s - %s)\n", u\_boot\_rev, U\_BOOT\_DATE,          U\_BOOT\_TIME);      omap\_rev\_string(rev\_string\_buffer);      printf("Texas Instruments %s\n", rev\_string\_buffer);  } <span style="font-size:14px;color:#003399;"></span> |

**@c8@ 配置 DDR**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | config\_am335x\_ddr();    《PATH ：》  /\*  void DDR2\_EMIF\_Config(void); \*/  static void config\_am335x\_ddr(void)  {      int data\_macro\_0 = 0;      int data\_macro\_1 = 1;        enable\_ddr\_clocks();        config\_vtp();        Cmd\_Macro\_Config();        Data\_Macro\_Config(data\_macro\_0);      Data\_Macro\_Config(data\_macro\_1);        \_\_raw\_writel(PHY\_RANK0\_DELAY, DATA0\_RANK0\_DELAYS\_0);      \_\_raw\_writel(PHY\_RANK0\_DELAY, DATA1\_RANK0\_DELAYS\_0);        \_\_raw\_writel(DDR\_IOCTRL\_VALUE, DDR\_CMD0\_IOCTRL);      \_\_raw\_writel(DDR\_IOCTRL\_VALUE, DDR\_CMD1\_IOCTRL);      \_\_raw\_writel(DDR\_IOCTRL\_VALUE, DDR\_CMD2\_IOCTRL);      \_\_raw\_writel(DDR\_IOCTRL\_VALUE, DDR\_DATA0\_IOCTRL);      \_\_raw\_writel(DDR\_IOCTRL\_VALUE, DDR\_DATA1\_IOCTRL);        \_\_raw\_writel(\_\_raw\_readl(DDR\_IO\_CTRL) & 0xefffffff, DDR\_IO\_CTRL);      \_\_raw\_writel(\_\_raw\_readl(DDR\_CKE\_CTRL) | 0x00000001, DDR\_CKE\_CTRL);        config\_emif\_ddr2();  }      《PATH : /arm/include/asm/arch-ti81xx/cpu.h》  #define DATA0\_RANK0\_DELAYS\_0        (DDR\_PHY\_BASE\_ADDR + 0x134)  #define DATA1\_RANK0\_DELAYS\_0        (DDR\_PHY\_BASE\_ADDR + 0x1D8)    /\* DDR offsets \*/  #define DDR\_PHY\_BASE\_ADDR       0x44E12000  #define DDR\_IO\_CTRL         0x44E10E04  #define DDR\_CKE\_CTRL            0x44E1131C  #define CONTROL\_BASE\_ADDR       0x44E10000 |

http://blog.chinaunix.net/attachment/201302/5/28458801_1360055466Bjn9.png

**@c DONE@**

**@b DONE@**

**@a4@ 设置 internal RAM 内存空间的栈指针，调用 board\_init\_f()函数**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | /\* Set stackpointer in internal RAM to call board\_init\_f \*/  call\_board\_init\_f:      ldr sp, =(CONFIG\_SYS\_INIT\_SP\_ADDR)      bic sp, sp, #7 /\* 8-byte alignment for ABI compliance \*/      ldr r0,=0x00000000      bl  board\_init\_f |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | <PATH: include/configs/am335x\_evm.h>  #define CONFIG\_SYS\_INIT\_SP\_ADDR     (CONFIG\_SYS\_INIT\_RAM\_ADDR + \                       CONFIG\_SYS\_INIT\_RAM\_SIZE - \                       GENERATED\_GBL\_DATA\_SIZE)    #define CONFIG\_SYS\_INIT\_RAM\_ADDR    SRAM0\_START  #define CONFIG\_SYS\_INIT\_RAM\_SIZE    SRAM0\_SIZE    <PATH : /arch/arm/include/asm/arch-ti81xx/hardware.h : begin>  #ifdef CONFIG\_AM335X  #define SRAM0\_START         0x402F0400  #else  #define SRAM0\_START         0x40300000  #endif  <PATH : /arch/arm/include/asm/arch-ti81xx/hardware.h : end>    <PATH : /arch/arm/include/asm/arch-ti81xx/cpu.h : begin>  #if defined(CONFIG\_AM335X) || defined(CONFIG\_TI814X)  #define SRAM0\_SIZE          (0x1B400) /\* 109 KB \*/  #define SRAM\_GPMC\_STACK\_SIZE        (0x40)  #endif  <PATH : /arch/arm/include/asm/arch-ti81xx/cpu.h : end>    <PATH : /am335x/include/generated/generic-asm-offsets.h : begin>  #define GENERATED\_GBL\_DATA\_SIZE (128) /\* (sizeof(struct global\_data) + 15) & ~15 \*/  <PATH : /am335x/include/generated/generic-asm-offsets.h : end> |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <PATH : /arch/arm/cpu/armv7/omap-common/spl.c'>  void board\_init\_f(ulong dummy)  {      /\*       \* We call relocate\_code() with relocation target same as the       \* CONFIG\_SYS\_SPL\_TEXT\_BASE. This will result in relocation getting       \* skipped. Instead, only .bss initialization will happen. That's       \* all we need       \*/      debug(">>board\_init\_f()\n");      relocate\_code(CONFIG\_SPL\_STACK, &gdata, CONFIG\_SPL\_TEXT\_BASE);  }    <PATH : /arch/arm/cpu/armv7/omap-common/spl.c : begin>  #define CONFIG\_SPL\_TEXT\_BASE        0x402F0400  #define CONFIG\_SPL\_MAX\_SIZE     (46 \* 1024)  #define CONFIG\_SPL\_STACK        LOW\_LEVEL\_SRAM\_STACK  <PATH : /arch/arm/cpu/armv7/omap-common/spl.c : end>    <PATH : /arch/arm/include/asm/arch-ti81xx/omap.h : begin>  #define LOW\_LEVEL\_SRAM\_STACK    0x4030B7FC  <PATH : /arch/arm/include/asm/arch-ti81xx/omap.h : end> |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <PATH : /arch/arm/cpu/armv7/start.S>  /\*   \* void relocate\_code (addr\_sp, gd, addr\_moni)   \*   \* This "function" does not return, instead it continues in RAM   \* after relocating the monitor code.   \*   \*/      .globl  relocate\_code  relocate\_code:      mov r4, r0  /\* save addr\_sp \*/      mov r5, r1  /\* save addr of gd \*/      mov r6, r2  /\* save addr of destination 0x402F0400\*/ |

**@a5@ 代码重定位**

**代码重定向，它首先检测自己(MLO)是否已经在内存中：**

**如果是直接跳到下面的堆栈初始化代码 clear\_bss。**

**如果不是就将自己从Nor Flash中拷贝到内存中。**

**Nor Flash 和Nand Flash 本质区别就在于是否进行代码拷贝，也就是下面代码所表述：无论  
是Nor Flash 还是Nand Flash，核心思想就是将 uboot 代码搬运到内存中去运行，但是没有拷  
贝bss 后面这段代码，只拷贝bss 前面的代码，bss 代码是放置全局变量的。Bss 段代码是为  
了清零，拷贝过去再清零重复操作。**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | /\* Set up the stack                         \*/  stack\_setup:      mov sp, r4        adr r0, \_start      cmp r0, r6      moveq   r9, #0      /\* no relocation. relocation offset(r9) = 0 \*/      beq clear\_bss       /\* skip relocation \*/      mov r1, r6          /\* r1 <- scratch for copy\_loop \*/      ldr r3, \_image\_copy\_end\_ofs      add r2, r0, r3      /\* r2 <- source end address      \*/    copy\_loop:                              /\* 自拷贝 \*/      ldmia   r0!, {r9-r10}       /\* copy from source address [r0]    \*/      stmia   r1!, {r9-r10}       /\* copy to   target address [r1]    \*/      cmp r0, r2          /\* until source end address [r2]    \*/      blo copy\_loop |

**@a6@ 清空 bss 段**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | clear\_bss:        ldr r0, \_bss\_start\_ofs      ldr r1, \_bss\_end\_ofs      mov r4, r6          /\* reloc addr \*/      add r0, r0, r4      add r1, r1, r4        mov r2, #0x00000000     /\* clear                \*/    clbss\_l:str r2, [r0]        /\* clear loop...            \*/      add r0, r0, #4      cmp r0, r1      bne clbss\_l    /\*   \* These are defined in the board-specific linker script.   \*/  .globl \_bss\_start\_ofs  \_bss\_start\_ofs:      .word \_\_bss\_start - \_start          /\* \_\_bss\_start = 0x80000000 \*/ |

**@a7@ 调用函数 board\_init\_r，用以完成 MLO（SPI）阶段的所有初始化，并跳转到 uboot.img 阶段**

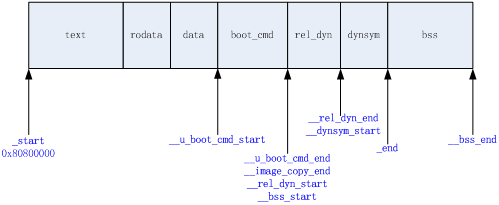
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | /\*   \* We are done. Do not return, instead branch to second part of board   \* initialization, now running from RAM.   \*/  jump\_2\_ram:  /\*   \* If I-cache is enabled invalidate it   \*/  #ifndef CONFIG\_SYS\_ICACHE\_OFF      mcr p15, 0, r0, c7, c5, 0   @ invalidate icache      mcr     p15, 0, r0, c7, c10, 4  @ DSB      mcr     p15, 0, r0, c7, c5, 4   @ ISB  #endif      ldr r0, \_board\_init\_r\_ofs      adr r1, \_start      add lr, r0, r1      add lr, lr, r9      /\* setup parameters for board\_init\_r \*/      mov r0, r5      /\* gd\_t \*/      mov r1, r6      /\* dest\_addr \*/      /\* jump to it ... \*/      mov pc, lr    \_board\_init\_r\_ofs:      .word board\_init\_r - \_start |

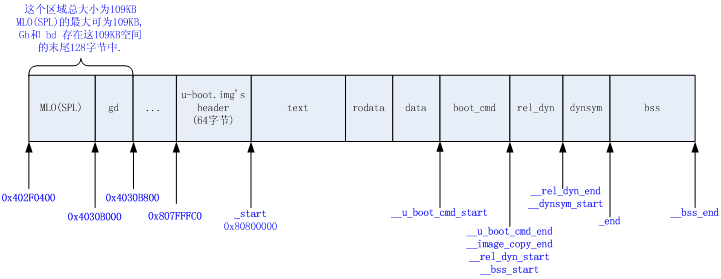
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | 《PATH : /arch/arm/cpu/armv7/omap-common/spl.c 》  void board\_init\_r(gd\_t \*id, ulong dummy)  {      u32 boot\_device;      debug(">>spl:board\_init\_r()\n");        timer\_init();      i2c\_init(CONFIG\_SYS\_I2C\_SPEED, CONFIG\_SYS\_I2C\_SLAVE);    #ifdef CONFIG\_SPL\_BOARD\_INIT      spl\_board\_init();  #endif        boot\_device = omap\_boot\_device();      debug("boot device - %d\n", boot\_device);      switch (boot\_device) {  #ifdef CONFIG\_SPL\_MMC\_SUPPORT      case BOOT\_DEVICE\_MMC1:      case BOOT\_DEVICE\_MMC2:          spl\_mmc\_load\_image();          break;  #endif  #ifdef CONFIG\_SPL\_NAND\_SUPPORT      case BOOT\_DEVICE\_NAND:          spl\_nand\_load\_image();          break;  #endif  #ifdef CONFIG\_SPL\_YMODEM\_SUPPORT      case BOOT\_DEVICE\_UART:          spl\_ymodem\_load\_image();          break;  #endif      default:          printf("SPL: Un-supported Boot Device - %d!!!\n", boot\_device);          hang();          break;      }        switch (spl\_image.os) {      case IH\_OS\_U\_BOOT:          debug("Jumping to U-Boot\n");          jump\_to\_image\_no\_args();          break;      default:          puts("Unsupported OS image.. Jumping nevertheless..\n");          jump\_to\_image\_no\_args();      }  } |

**@a DONE@**

**3，第三级 bootloader：uboot.img 做了哪些事情？**

**uboot.img 内存分布如下：**





**访问 /arch/arm/lib/board.c 中** **的 board\_init\_f() 函数：**

在 uboot.img 运行过程中，有两个非常重要的结构体：gd\_t 和 bd\_t 。

其中 **gd\_t** ：global\_data 数据结构的定义，位于：/arch/arm/include/asm/global\_data.h 中。

                 其成员主要是一些全局的系统初始化参数。

其中 **bd\_t** ：bd\_info 数据结构的定义，位于：/arch/arm/include/asm/u-boot.h 中。

                 其成员是开发板的相关参数。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70 | <PATH : /arch/arm/include/asm/global\_data.h >  /\*   \* The following data structure is placed in some memory which is   \* available very early after boot (like DPRAM on MPC8xx/MPC82xx, or   \* some locked parts of the data cache) to allow for a minimum set of   \* global variables during system initialization (until we have set   \* up the memory controller so that we can use RAM).   \*   \* Keep it \*SMALL\* and remember to set GENERATED\_GBL\_DATA\_SIZE > sizeof(gd\_t)   \*/    typedef struct  global\_data {      bd\_t        \*bd;      unsigned long   flags;      unsigned long   baudrate;      unsigned long   have\_console;   /\* serial\_init() was called \*/      unsigned long   env\_addr;   /\* Address  of Environment struct \*/      unsigned long   env\_valid;  /\* Checksum of Environment valid? \*/      unsigned long   fb\_base;    /\* base address of frame buffer \*/  #ifdef CONFIG\_FSL\_ESDHC      unsigned long   sdhc\_clk;  #endif  #ifdef CONFIG\_AT91FAMILY      /\* "static data" needed by at91's clock.c \*/      unsigned long   cpu\_clk\_rate\_hz;      unsigned long   main\_clk\_rate\_hz;      unsigned long   mck\_rate\_hz;      unsigned long   plla\_rate\_hz;      unsigned long   pllb\_rate\_hz;      unsigned long   at91\_pllb\_usb\_init;  #endif  #ifdef CONFIG\_ARM      /\* "static data" needed by most of timer.c on ARM platforms \*/      unsigned long   timer\_rate\_hz;      unsigned long   tbl;      unsigned long   tbu;      unsigned long long  timer\_reset\_value;      unsigned long   lastinc;  #endif  #ifdef CONFIG\_IXP425      unsigned long   timestamp;  #endif      unsigned long   relocaddr;  /\* Start address of U-Boot in RAM \*/      phys\_size\_t ram\_size;   /\* RAM size \*/      unsigned long   mon\_len;    /\* monitor len \*/      unsigned long   irq\_sp;     /\* irq stack pointer \*/      unsigned long   start\_addr\_sp;  /\* start\_addr\_stackpointer \*/      unsigned long   reloc\_off;  #if !(defined(CONFIG\_SYS\_ICACHE\_OFF) && defined(CONFIG\_SYS\_DCACHE\_OFF))      unsigned long   tlb\_addr;  #endif      void        \*\*jt;       /\* jump table \*/      char        env\_buf[32];    /\* buffer for getenv() before reloc. \*/  } gd\_t;    #define DECLARE\_GLOBAL\_DATA\_PTR     register volatile gd\_t \*gd asm ("r8")      <PATH : /arch/arm/include/asm/u-boot.h >  typedef struct bd\_info {      int         bi\_baudrate;    /\* serial console baudrate \*/      unsigned long   bi\_ip\_addr; /\* IP Address \*/      ulong           bi\_arch\_number; /\* unique id for this board \*/      ulong           bi\_boot\_params; /\* where this board expects params \*/      struct              /\* RAM configuration \*/      {      ulong start;      ulong size;      }           bi\_dram[CONFIG\_NR\_DRAM\_BANKS];  } bd\_t; |

其中 **DECLARE\_GLOBAL\_DATA\_PTR** 宏定义在系统初始化过程中会被频繁调用，

**其的作用是，声明gd这么一个全局的指针，这个指针指向gd\_t结构体类型，并且这个gd指针是保存在ARM的r8这个寄存器里面的。**

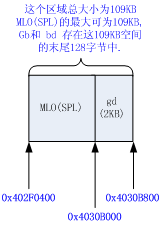
uboot.img 第一个运行的文件还是 start.o，其在运行访问的 board\_init\_f() 函数定义在 /arch/arm/lib/board.c 中：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <PATH : /arch/arm/lib/board.c >  void board\_init\_f(ulong bootflag)  {      bd\_t \*bd;      init\_fnc\_t \*\*init\_fnc\_ptr;      gd\_t \*id;      ulong addr, addr\_sp;        /\* Pointer is writable since we allocated a register for it \*/      gd = (gd\_t \*) ((CONFIG\_SYS\_INIT\_SP\_ADDR) & ~0x07);      /\* compiler optimization barrier needed for GCC >= 3.4 \*/      \_\_asm\_\_ \_\_volatile\_\_("": : :"memory");        memset((void \*)gd, 0, sizeof(gd\_t));            ...  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <PATH : /include/configs/am335x\_evm>  #define CONFIG\_SYS\_INIT\_RAM\_ADDR    SRAM0\_START  #define CONFIG\_SYS\_INIT\_RAM\_SIZE    SRAM0\_SIZE  #define CONFIG\_SYS\_INIT\_SP\_ADDR     (CONFIG\_SYS\_INIT\_RAM\_ADDR + \                       CONFIG\_SYS\_INIT\_RAM\_SIZE - \                       GENERATED\_GBL\_DATA\_SIZE)    <PATH : /arch/arm/include/asm/arch-ti81xx/hardware.h>  #define SRAM0\_START         0x402F0400    <PATH : /arch/arm/include/asm/arch-ti81xx/cpu.h>  #define SRAM0\_SIZE          (0x1B400) /\* 109 KB \*/    <PATH : /am335x/include/generated/generic-asm-offsets.h>  #define GENERATED\_GBL\_DATA\_SIZE (128) /\* (sizeof(struct global\_data) + 15) & ~15 \*/ |

因此，系统初始化参数将会被保存在 （保存 MLO（SPL）文件的内存空间的）末尾 2 KB 处。

通过计算的 gb 指针指向的内存空间地址为 gb = 0x4030B000



gb\_t 结构体中某些元素的值是来自于 uboot.img's header，这个header的数据保存在内存的0x807FFFCO，大小为 64字节

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <PATH : /include/image.h>  /\*   \* Legacy format image header,   \* all data in network byte order (aka natural aka bigendian).   \*/  typedef struct image\_header {      uint32\_t    ih\_magic;   /\* Image Header Magic Number    \*/      uint32\_t    ih\_hcrc;    /\* Image Header CRC Checksum    \*/      uint32\_t    ih\_time;    /\* Image Creation Timestamp \*/      uint32\_t    ih\_size;    /\* Image Data Size      \*/      uint32\_t    ih\_load;    /\* Data  Load  Address      \*/      uint32\_t    ih\_ep;      /\* Entry Point Address      \*/      uint32\_t    ih\_dcrc;    /\* Image Data CRC Checksum  \*/      uint8\_t     ih\_os;      /\* Operating System     \*/      uint8\_t     ih\_arch;    /\* CPU architecture     \*/      uint8\_t     ih\_type;    /\* Image Type           \*/      uint8\_t     ih\_comp;    /\* Compression Type     \*/      uint8\_t     ih\_name[IH\_NMLEN];  /\* Image Name       \*/  } image\_header\_t; |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <PATH : /include/configs/am335x\_evm.h>  /\*   \* 8MB into the SDRAM to allow for SPL's bss at the beginning of SDRAM.   \* 64 bytes before this address should be set aside for u-boot.img's   \* header. That is 0x807FFFC0--0x80800000 should not be used for any   \* other needs.   \*/  #define CONFIG\_SYS\_TEXT\_BASE        0x80800000 |