	使用 Trace32	调试 SAM	9 SDRAM 代	<b>认</b> 码
文档编号	MAN3009A_CH			
文档版本	Rev. A			
文档摘要	描述了使用 Trace32 来调试运	行于 SDRAM 的	SAM9 代码的方法	和过程
关键词	Trace32 SAM9 SDRAM 程序			
创建日期	2010-01-19 创建人员	Dracula	审核人员	<u>Hotislandn</u>
文档类型	公开发布/开发板配套文件			
版权信息	Mcuzone 原创文档,转载请注	E明出处		

更新历史

版本	时间	更新	作者
Rev. A	2010-01-19	初始创建	Dracula
1	御祷山之 氏徳山マ		
	倾拴电丁	B143	
	销售 TEL: +86-571-899	08193	
	支持 TEL: 1891398916	6 13770507096	
	FAX: +86-571-8990819	3	
	www.mcuzone.com ww	/w.atarm.com	

# 1.概述

Trace32 ARM ICD(以下简称 T32)是一款高性能的 ARM 仿真器,自带调试软件环境,功能很强大。 在裸奔的情况下, SAM9 大部分用户的代码都将运行于 SDRAM,因为内部 RAM 空间有限。而代码运行于 SDRAM 时,调试就有了其特殊性。但是如果拥有了 T32,那么一切就可以迎刃而解。 本文介绍使用 T32 调试运行于 SAM9261S SDRAM 中的 ucos-II 的代码的基本过程。 Ucos-II 使用 keil 编译, keil 版本是 3.80a。

冒

# 2. 安装 T32 调试软件

## 2.1 安装软件

使用 T32 提供的安装光盘安装 T32 调试软件。 安装完成后连接上 T32 的硬件,然后安装驱动。 注意:如果是 vista 以上(含)的系统,需要安装特殊的驱动 然后在桌面创建 T32 的快捷方式:





## 3.1 修改代码

需要编译 Bootstrap 的原因有两点:一是需要使用编译生成的 bin 文件来加载正式的 SDRAM 代码,具体原因可以参考本站的 MAN3008,二是编译生成的 elf 文件可以用于 T32 初始化 SAM9261S 的外部 SDRAM,然后 再加载 ucos-II 的 axf 文件,因为 SDRAM 在初始化之前是无法使用的。

ATMEL 原始提供的 Bootstrap 源码是为 SAM9261 编写的,并不适合 SAM9261S,需要修改。

修改加载参数, ucos-II 的代码将被烧写到 data flash 的 0x8400 处,并被加载到 SDRAM 的 0x20000000 处运行:

#define IMG ADDRESS  $\rightarrow 0 \times 8400 \rightarrow$ →/\*·Image·Address·in·DataFlash·\* #define>IMG 0x40000>-→/\*·Image·Size·in·DataFlash· ·····0x350····/\*·AT91SAM9261-EK·\*/ #define MACH TYPE  $\rightarrow 0 \times 20000000 \rightarrow -$ #define JUMP ADDR →/\*·Final·Jump·Addre

修改 Makefile 中的栈顶的定义,因为 SAM9261S 只有 16KB 的内部 RAM:

RamD	Disk (J:)	▶ uSh	are 🕨	Bootst	rap-v1.14	4 <b>▶</b> bo	oard	▶ at	91sam	9261e	c ► d	ataflash	1
灵	新建文	(件夹											-
名称				^					修改	日期		类型	1
🗋 at	t91sam	9261ek	.h						201	0/1/6 2	0:28	Н文	8
	/akefile	•							201	0/1/19	20:53	文件	
board	∖at91sa	m9261	ek∖data	flash∖M	lakefile]								
		<b>•</b>	\$				<b>-</b> 4	簡 Find	<b>•</b>				$\langle \cdot \rangle$
Make	file												
16 17 18 19 20 21 22 23 24	TARGE # Boa BOARD # Lin LINK #TOP TOP 0 # Nam PROJE	T=AT91 rd·nam at91s k·Addr ADDR=0 OF_MEM F_MEMO re·of·c CT=dat	SAM926 e·(cas am9261 ess·an x30000 ORY=0x3 RY=0x3 urrent aflash	1 e'sens ek d'Top_ 0 328000 04000 'direc	itive!! of_Memo tory	!) ry							
也可以 <b>3.2</b>	<sup>以在头了</sup> 编译	<sup>文件中打</sup> <b>圣代征</b>	丁开 de 马	bug 的习	<b>支持,</b> 添	加一些	を用す	定义	的字符	串。			
运行 <b>5 ma</b> 这里伯 编译	make 即 ke CR( 使用的步 生成的)	□可: 0SS_CC 是 arm-□ 文件:	MPILE none-ea	=arm- ibi-的工	none-ea 具链。	abi-							
2 1 代码的	4516 25330 15881 約 mem	Jan 1 Jan 1 Jan 1 消耗的	9 20:9 9 20:9 9 20:9 9 20:9	58 dat 58 dat 58 dat	aflash aflash aflash	_at91 _at91 _at91	.sam .sam .sam	7261e 7261e 7261e	k.bin k.eli k.maj	n E p			
ari te 4!	m-none ext 516	e-eabi data Ø	-s12e	aatai bss 0	dec 4516	11918a 11	hex La4	filer dataf	.e1f Name Slash	_at91	.sam92	261ek.	elf

生成的 bin 文件用于烧写, elf 文件用于加载调试。

# 4. ucos-II 代码

## 4.1 编译代码

ucos-II 的代码使用 keil MDK 编译,代码的 link 地址必须在 0x2000-0000,与前面 Bootstrap 指定的一致。同时需要设定好 ram 中的异常向量。

对于 SAM9261S SDRAM 这样的应用,建议用 scatter loader file 来控制应用的 mem map。 编译完成后生成 axf 文件如下:

🛯 ucos2_sdram.axf	AXF 文件	360 KE
ucos2_sdram.bin	BIN 文件	27 KE

生成的 axf 文件用于调试, bin 文件用于烧写。

5. 调试代码

## 5.1 初始化环境

首先将 SAM9261S 板子上电,将 dbgu 连接到 PC,开启终端软件,并使板子进入 SAM boot 模式,终端收到 如下信息:

ROMBOOT

连接好 T32,包括板子侧的 20pin 电缆和 PC 侧的 USB 电缆,运行 t32 的 IDE:

TRACE32																
<u>F</u> ile <u>E</u> dit <u>V</u> i	ew Var	<u>B</u> reak <u>R</u> u	n <u>C</u> PU <u>N</u>	<u>M</u> isc <u>T</u> ra	ace <u>P</u> erf	Cov Window	v <u>H</u> elp									
H M I	e 6 1	• II   🖄	? <b>\</b> ?	•    E	333 🔳	🛃 🛃 🐻 ।	ا 🛓 🏮									
B:: TRACE32 De	evelopme	nt Suste	n (C) 19	89-2 <b>00</b> 6	i Lauterk	ach Datent	echnik									
omulato	trigger				Data		DEDE	SVStore	Ston	60	Broak	Perietor	c⊻mbol	othor	morriouc	
emulate	uiggei	device	• ua		Duid	- vai		La ratem		u				outer	previous	<b>D</b>
		_									system dow	0			MIX	<b>P</b>

使用 Trace32 调试 SAM9 SDRAM 代码

在底端的命令行依次输入如下命令:



并将其合理排列一下:

TRACE32	
Eile Edit View Var Break Run ⊆PU Misc Irace Perf Cov ARM Window Help ▶ ▶ ↓ ↓ ↓ ↓ ▶ 11   ※   ? ▶?   ◎   ※ 12 # ■ & & & & & & & & & & & & & & & & & &	
E BrDatalist	B:Var.Frame /Locals /Caller
N Step     ₩ Over     ↓ Next     ✔ Return     C Up     ▶ Go     II Breek     ⊠ Mode     Find:       addr/line     code     Label     broasen is     broasen is     broasen is	Locals ♥ Caller Tasi     Angs ♥ Locals ♥ Caller Tasi     Angliss : 8x400E84 (asp)
Course rule         Course rule         Course rule         Course rule           SR:00400F60         EFFFFC0         b         0x400E70         ^           SR:00400F60         EFFFFC0         b         0x400E70         ^	-001 SR:0x401354 (asm)
SR:00400F70         EBFFFEF8         b1         0x400B58           SR:00400F74         EAFFFFBD         b         0x400E70	end of frame
SR:00400778         E92D4010         stndb         r131,{r4,r14}           SR:00400F7C         E1A04000         mov         r4,r0           SP:00400F0C         E5000400         Ide         r0	
SR:00400F04         ES00101         Idr         r1, r0, #0x1C1           SR:00400F04         ES00101         Idr         r1, r0, #0x1C1           SR:00400F08         E3110040         tst         r1, #0x1000	-
SR:00400F8C         0A00000A         beq         0x400FBC           SR:00400F90         E3A01D40         mov         r1,#0x1000	
SR:00400154         E5001020         str         r1, Ir0, #0x20]           SR:004001598         E3801000         mvn         r1, #0x0           SR:004001591         E3801020         str         r0, #0x20]	
SR:00400FA0         E3A01000         mov         r1,#0x00           SR:00400FA4         E5801028         str         r1,[r0,#0x28]	address types inpl
SR:00400FA8         E3A01F40         mov         r1,#0x100           SR:00400FA0         E5001000         str         r1,(r0,#0x01)           SR:00400FA0         E5001000         str         r1,(r0,#0x01)	
SR:00400FD4         ES801030         str         r1,10,40x301           SR:00400FD4         ES001030         str         r1,0,40x301	
SR:00400FBC         E3110001         tst         r1,#0x1           SR:00400FC0         13A01001         movne         r1,#0x1	
SR:004001C3         Close 11,00004         strne         r1,r0,#0x20J           SR:00400FC3         11,000004         movne         r0,r4           SP:00400FC3         11,000004         hovne         r0,r4	B::Register
SR:80400FD0         ESD4004         Idth         rdy.[r4,#0x4]           SR:80400FD4         E8D8010         Idmia         r13!, (r4,pc)	N N RØ FFFA4000 R8 81A01085 SP> 00327088 A 2 - R1 00400F78 R9 00327058 +04 00401354 - R1 00400F78 R9 00327058 +04 00400578 +04 00401354 - R1 00400F78 R9 00327058 +04 00420578 +04 0040058 +04 0040058 +04 0040058 +04 0040058 +04 0040000000000000000000000000000000
SR:00400FD8         E92D43F0         stmdb         r13!,(r4-r9,r14)           SR:00400FDC         E1007000         mov         r7,r0	V _ R3 00327088 R10 4917534H *08 8086004 V _ R3 00327088 R11 6BF5C3F3 *0C 264BCBE5 I R4 00327070 R12 2000 *10 B5F64092
SR:00400FE4         E5905000         Idr         r4, Ir01, H0x8           SR:00400FE4         E5905000         Idr         r5, Ir01           SR:00400FE4         E1009002         mov         r4, r22	F F R5 00327070 R13 00327F90 +14 74577A16 T _ R6 1000 R14 00401354 +18 6B4C06A7
SR:00400FEC         E1008001         mov         r8;r1         +	J _ R7 0C028242 PC 00400F84 +1C 2CA8104E svc SPSR 10 CPSR 800000D3 +20 F4B2221F -
TRACE32 Development System (C) 1989-2006 Lauterbach Datentechnik	
emulare trigger devices trace Data Var PEHP System Step Go SR:00400F84 stop	ped MIX UP
排列好后可以将当前的设置保友到文件,	
Go Arrange Icons	
Create Duplicate window	
1F70 Clear Windows on Page	
Clear Windows on Fage	
B58	
E70 😰 Store Windows to	
Load Windows from	
下伏可以直接 load 这个 window 的 这直从 间 避 免 里 夏 设 直。	
比如 save 到这个文件:	
▶ T32 ▶ dbg_cmm	
2222	
<b>中</b> 你	
man3009.cmm	
下次左从珊翠煊止时 古拉庙田下面的人人計可以加我穷口汎罢	
Γ(八仁处理益停止凹, 旦佞使用下凹的命令肌可以加软菌口设直:	
B::do_dbg_cmm\man3009_cmm	

B::|do dbg\_cmm\man3009.cmm TRACE32 Development System (C)

## 5.2 加载 Bootstrap

使用命令加载 Bootstrap 的 elf 文件:

 $\texttt{B::} \texttt{d.load.elf_j:\weakset} are \texttt{Bootstrap-v1.14} ard \texttt{at91sam9261ek} ataflash \texttt{dataflash\_at91sam9261ek.elf} are \texttt{bootstrap-v1.14} ard \texttt{at91sam9261ek} are \texttt{bootstrap-v1.14} ard \texttt{at91sam9261ek} are \texttt{bootstrap-v1.14} are \texttt{bootstrap-v1.14}$ 

文件路径可以粘贴进去,也可以凭记忆输入,支持 tab 键自动补全。 敲回车可以看到 elf 被加载,并进入 debug 状态。 如果找不到 source 文件,可以用下面命令指定搜索路径:

#### B::SYMBOL.SPATH j:\uShare\Bootstrap-v1.14\

代码合一的效果: B::Data.List - - **- - X** N Step ] Nover 🖌 🗸 Return C Up Go Go I Break 🛛 🖄 Mode 🗍 Find: crt0\_qnu.S ↓ Next addr/line code label mnemonic comment .alion 4 reset: /\* Exception vectors (should be a branch to be detected as a valid code by the rom \*/exception\_vectors: 54 b reset\_vector /\* reset \*/ SR:0030000 0x30003 \_init\_stack ADreset h /\* Undefined Instruction \*/ undef\_vector 55 b SR:00300004 FANNNAS 0x300020 undef\_vector b 56 swi\_vector /\* Software Interrupt \*/ b SD-0000000 EU00000E ь 使用 mode 按钮可以转换为源文件和反汇编两种方式, 上图显示的是反汇编方式,也可以方便的查看源码。 使用上方的按钮可以控制程序的运行: [B::Data.List] 🗸 Return 💵 Break 📗 🎇 Mode 🕨 Step 🚽 Nover 🖌 **↓** Next Ċ Up -Go Find: addr/line code label mnemonic comment init\_stack: 81 sp,=TOP\_OF\_MEM 1dr SR:0030003C \_init\_st..:mov r13,#0x304000 同时可以观察寄存器窗 B::Register N RØ **R8** SP> ACDCEDB5 Ы FFFA4000 81A01085 00400F78 R9 00327058 +04 E38F6EFD R1 R2 0108 R10 4917B54A +08 4F6D4EB5 \_ V Ī R3 00327088 6BF5C3F3 +0C BACF7E0F R11 Ι R4 00327070 R12 2000 +10 CEEFFBD4 F **R5** 00327070 R13 00304000 +14 2EF55739 Т R6 1000 R14 00401354 +18 7974DB7B 0C028242 J R7 PC 00300040 +1C 3E6757EF \_ SPSR CPSR 8000003 +20 9428FA39 10 SVC

由于加载 Bootstrap 的目的只是为了初始化 SDRAM,所以直接在程序最后设置一个断点:

					the second se	
183	bx	$\mathbf{r}0$				
SR:00300100	E12FFF10			bx	r0	
VU100C00+05	N0204404	- In	data:	ded	00001107	
使用 Trace32 调词	式 SAM9 SDRAM 代码					

**MAN3009A** 

设置断点的方法是直接双击要设置的程序位置即可,设置完成后在断点窗口里也可以看见:

以且明点的方法 ⑧ B::Break.Lis	定且按从山安! t			口主也可以有见	:
X Delete All	🕽 Disable All 🔘 E	nable All 🛛 😕 Init	🛿 Select) 😰		
address R:00300100	types Program	impl SOFT _gov	2		
点击程序控制里	h Go 按钮,	-9-			
C Un	▶ Gn II B	reak Mode			
mnemonic		comment			
程序停在断点处	2:				$\mathbf{\Lambda}$
SR:003000FC	E1AØEØØF	·, r- m	ov r14,pc		
183 SR:00300100	bx E12FFF10	r0 b	x r0		
SR:00300104	003011A4	_lp_data: d	cd 0x3011A4		
灰条即是当前 P	C 的位置,现在	寄存器显示程序将跳结	转到 0x2000-0000 处词	室行:	
B::Register	000000 00				
Z _ R1	0350 R9	00327058 +04			
C _ R2 V _ R3 Ø	0 R10 03011A4 R1:	4917854A +08 68F5C3F3 +0C			
PC 上的终端串口	口也会有 Bootsti	rap 的 debug 输出:			
>Start AT916	Bootstrap				
此可, SDRAIM L	口空彻炉化刷角	°			
5.3 加载 ι	ıcos-II		·		
在前面的 Bootst	rap 在断点处停	下后,直接使用相同的	的命令再次加载 ucos-	II 的 axf 文件:	
B::d.load.el	f	`\M29261S\s	sam9_ucos2_mdk_sd	ram\output\uco	os2_sdram.axf
加载的效果:	$\Lambda N$				
	ENTRY Entry reset	•			
103	ь	recat vector	· recet		
SR:2000000	EA00000D	b	0x20000030	,	
104 SR:20000004	ь ЕА000005	undef_vector b	c ; Undefined 0x20000020	Instruction	
105 SR:2000008	<b>b</b> Еа000005	swi_vector h	; Software : 0x20000024	Interrupt	
106 SR:20000000	р Голалас	pabt_vector	; Prefetch A	abort	
107	LU0000000	dabt_vector	; Data Abort	t	
58:20000010 108	сниииии5 <b>b</b>	b rsvd_vector	interved; ; reserved	,	
SR:20000014	EA000005	b	0x20000030	1	

PC 指针位于 0x2000-0000 处。

源代码方式:				
addr/line	source			
	Entry_reset			
103	b	reset_vector	; reset 👘	
104	b	undef_vector	; Undefined	d Instruction
105	b	swi_vector	; Software	Interrupt
106	b	pabt_vector	; Prefetch	Abort
107	b	dabt_vector	; Data Abor	rt
108	b	rsvd_vector	; reserved	
109	b	irg_vector	; IRQ :	: read the AIC
110	b	fiq_vector	; FIQ	

#### undef\_vector

此时可以使用程序控制命令控制程序的运行:

130		ldr	r0. = TOP OF	STACK	: defined	in Makefile	
		2.02			,		
	: - Set	uv Fast	Interrupt M	ode and	set FIO M	ode Stack	
133	/	msr	CPSR_c, #(A	RM_MODE_	FIQ¦I_BIT	F_BIT)	
134		mov	r13, r0			; Init stack	FIQ
135		sub	r0, r0, #F	IO_STACK	SIZE		

## 5.4 设置断点

.

虽然可以直接找到代码所在,然后双击打断点。但是当工程中包含的文件很多时,找代码也是麻烦。在 T32 的调试环境下可以很方便的设置断点,只要知道公有函数的名字。 比如,ucos-II 的代码中有一个在 RAM 中安装异常向量的函数:

```
55
51 ; IRAM must remap to 0 for ARM execeptions
52 ER_IRAM_VECTOR 0x000000 EMPTY 0x100
53 {
54 ; OS_CPU_InitExceptVect() will install ISR here
55 }
56
```

直接在 T32 环境的命令行中输入:

B::b.s OS\_CPU\_InitExceptVect

输入过程中可以使用 tab 键自动补全。输入完成后回车,即可在那个函数处设置一个断点:

😢 B::Break.List				
🗶 Delete All 🔘	Disable All	🕽 Enable All	🛛 Init 🛛 🖉 Select 😰 Store 💈	9
address	types	impl		_
R:20004A70	Program	SOFT	OS_CPU_InitExceptVect	*

选择 Store...可以将当前的断点设置保存到文件,而使用 Load...可以加载以前的断点设置文件,可以直接使用之前的断点设置,继续调试。

#### 使用 Trace32 调试 SAM9 SDRAM 代码

#### 按运行,看到代码停在断点处:



## 5.5 查看内存

在前面的代码处停下来后,我们可以打开内存查看窗口,看看异常向量的安装。

B::d.dump 0x0 /NOASCII

使用 Trace32 调试 SAM9 SDRAM 代码

回车后会打开内存查看窗口。也可以使用菜单提供的功能:

A TRACE32	2	
File Edit	View Var Break Run	CF
N N   -	<u>R</u> egisters	
(	<u>翰 D</u> ump	
B::Dat	List Source	
🛛 🔪 Step	🐼 <u>W</u> atch	
addr/1	🛃 Re <u>f</u> erenced Var	

设置参看方式,由于是代码不需要 ascii 的显示,同时显示为 Long 类型,因为 ARM 指令是 32 位:

Address / Expr	ession ———					
0×0			🗕 🚺 🗆 HL	L	$\frown$	
Width ◎ default ◎ Byte ◎ Word ● Long 显示 data 窗口:	Access default E	Options Track Orient Ascii SpotLight	Flag Read Write	S.	3.	•
蹴 B::Data.dump ((	0x0) /Long /NoAscii	/DIALOG				, o <mark>x</mark>
C:0x0	j∄jFind M	odify Long	- E Tra	ok 🛛 📝 Hex	🗖 Ascii	
address	0 5000000 50000			4 00201104		
SD:00000020 SD:00000040 SD:00000060	EAFFFFFE EAFFF E59F00D0 E5901 E1811002 E58 <u>01</u>	603 EH000005 EH00 FFE EAFFFFE EAFF 000 E3A02001 E011 000 E59F00A8 E590	FFFE EAFFFFFE 1002 1A000009 1000 E3A02001	EAFFFFE E59F00C0 E0111002	EAFFFFFE   E3A01901   ØAFFFFFB	E3A0D9C1 = E3A02001 - E59F009C _
在安装向量的代码	马中单步, 就可以	看到 data 区域的变化				
507 (*( 508 (*(	INT32U *)0S_CPU_ INT32U *)0S_CPU_	ARM_EXCEPT_PREFETCH_ ARM_EXCEPT_PREFETCH_	ABORT_VECT_ADDR ABORT_HANDLER_A	) = DDR) = (INT	OS_CPU_ (32U)OS_CPU_	ARM_INSTR_ ARM_Except
510 (*( 511 (*(	INT32U *)OS_CPU_ INT32U *)OS_CPU_	ARM_EXCEPT_DATA_ABOR ARM_EXCEPT_DATA_ABOR	T_VECT_ADDR) T_HANDLER_ADDR)	= = (IN]	OS_CPU_ 320)0S_CPU_	ARM_INSTR_ ARM_Except
513 (*( 514 (*(	INT32U *)OS_CPU_ INT32U *)OS_CPU_	ARM_EXCEPT_ADDR_ABOR ARM_EXCEPT_ADDR_ABOR	T_VECT_ADDR) T_HANDLER_ADDR)	= = (IN]	OS_CPU_ 1320)OS_CPU_	ARM_INSTR_ ARM_Except
安装的向量:						
🚻 B::Data.dump	(0x0) /Long /NoA	scii /DIALOG				
C:0x0	ji∰j Find	Modify	<b>•</b>			
address SD:00000000 SD:00000010 SD:00000020 SD:00000030	0 ▶E59FF018 E59 EA000005 003 20004C10 200 EAFFFFFE EAF	4 8 0FF018 E59FF018 8 0011A4 EA000005 8 004C20 20004C30 2 FFFFE EAFFFFFE 8	C_ 59FF018 A000005 20004C40 3A0D9C1			

一直单步到该函数结束,可以看到代码的执行位置:



注意,断点窗口显示的断点命中状态。

这个函数计算当前系统最高优先级的任务,然后调用 OSStartHighRdy()真正的启动这个任务。

高亮调用 OSSt	artHighRdy()的那行,	右键选择"Go Till",					
void (	)SStart <mark>(v</mark> oid)						
783 if	(OSPunning == 0	E EXISE)					
705 11	OS_SchedNew();	LAUSE) (					
785	OSPrioCur =	= OSPrioHighRdy;					
786	OSTCBHighRdy =	= OSTCBPrioTb1[OS	SPrioHigh	Bdy];			
787	OSTCBCur =	<pre>= OSTCBHighRdy;</pre>					
}	obstar thrynwys	Program Add	dress				
790 }		🛨 Go Till					
/*\$PAGI	E*/	😻 Breakpoint					
/*		Breakpoints					
程序停在调用	处:						
void	d OSStart (void)	)					
{					N		
783	if (OSRunning ==	= OS_FALSE) {					
785	OSPrioCur	); = OSPrioHighRe	lv :				
786	OSTCBHighRdy	<pre>/ = OSTCBPrioTbl</pre>	[OSPrioH	ighRdy]; 🏓			
787	OSTCBCur	= OSTCBHighRdy	13				
788	OSStartHigh	Rdy();	_	/			
790 3	1			_			
打开反汇编模	式:						
SR:20001340 F	5810028		str rØ	.[p1.#0v28]			
SR:20001344 E5	5810020		str r0	,[r1,#0x20]			
788 SR:20001348 EA	OSStartHighRdy MOODF8	2 <mark>0</mark> ;	b Øx	/* Execute targ 20004830	get specific code to st ; OSStartHighRdu		
2	*\$PAGE*/				,		
单步跟入:		$\sim$					
📕 Step 🛛 🛤	Over 🔶 Next 🖌 Re	turn 🕑 Up 🕞 Go	📕 Break	Mode Find:	os_cpu_a.S		
addr/line o	code label		mnemonic		comment		
	;*************************************	*****	START MUI	**************************************	*******		
	; void OSStartHighRdy(void)						
	; Note(s) : 1) OSStar a) Cal	tHighRdy() MUST: 1 OSTaskSwHook() the	n				
	; b) Set	OSRunning to TRUE,	·,				
ŀ	; c) Switch to the highest priority task. :************************************						
	1995tartHighBdy						
	obter enrymay						
156	MSR CPSR_c. #	OS_CPU_ARM_CONTROL_	INT_DIS   OS	; Cl S_CPU_ARM_MODE_S	hange to SVC mode. SVC)		
SR:20004B30	321F0D3 OSSta	rtHighRdy:	msr cp	sr_c,#0xD3			
158	LDR R0, _OS_T	askSwHook		; 0:	STaskSwHook();		
SR:20004B34 E	159F0258 MOV LR. PC		ldr rØ	,0x20004D94			
SR:20004B38	E1AOEOOF		mov r1	4, рс			
SR:20004B3C	BX RU F12FFF10		hx rØ	1			

如果不习惯反汇编模式,可以直接看源码:

	;*****	;*************************************							
	; ; Note(; ; ; ;*****	s) : 1) OS a) b) c)	StartHighRdy() M Call OSTaskSwHo Set OSRunning t Switch to the h	UST: ok() then, o TRUE, ighest priority tasl	k. ************************************				
156	OSStart) MSR	HighRdy CPSR	⊂ #(OS CPU ARM	CONTROL INT DIS ! OS	; Change to SVC mode.				
158 159 160	LDR MOV BX	R0, _( LR, P0 R0	DS_TaskSwHook C		; OSTaskSwHook();				
162 163 164	LDR MOV STR	R0, _( R1, # B R1, []	DS_Running 1 R0]		; OSRunning = TRUE;				
单步 在执	跟踪这段 行恢复寄	代码,可以 存器之前停	看到任务是如何被 下:	<b></b> 皮启动的。	<b>v</b>				
	171 172 ;	LDR MSR LDMFD	R0, [SP], #4 SPSR_cxsf, R0 SP!, {R0-R12	, LR, PC}^					
注意	<b>176</b> 此时的寄	LDMFD LDMFD	SP!, {PC}^						
	B::Registe	r			<b>K</b>				
N _ 2 Z C C V _ I I F F T _ J _ SVC Q _	RØ R1 R2 R3 R4 R5 R6 R7 SPSR USR :	13 1 208015D8 60 20005FC4 208026BC 0 0 13	R8 R9 0032705 R10 R11 R12 R13 2080083 R14 20004B4 PC 20004B6 CPSR 6000001 FIQ:	0 SP> 30000000 58 +04 01010101 1 +08 02020202 0 +0C 03030303 0 +10 04040404 34 +14 05050505 10 +18 06060606 50 +1C 07070707 13 +20 08080808 +24 09090909 +28 10101010					
特别 按下	要注意其 单步恢复 <b>75</b>	中的 SP, 村 寄存器: LDMFD LDMFD S	这中的数据就是马_ SP!, {R0-R12, 1 P!, {R0-R12, 1 P!, {R0-R12, 1 P!, {R0-R12, 1 P!	上要恢复的数据。 LR, PC}^ LR}					

恢复	出的寄存	子器:							
	B::Registe	er			[	<u> </u>			
NZCVIFTJSVC 再执	RØ R1 R2 R3 R3 R4 R5 R6 R7 SPSR USR:	0 01010101 02020202 03030303 04040404 05050505 06060606 07070707 13	R8 R9 R10 R11 R12 R13 R14 PC CPSR FIQ: 一个任约	08080808 09090909 10101010 1111111 12121212 2080086C 14141414 20004B64 600000D3	SP> +04 +08 +0C +10 +14 +18 +1C +20 +24 +28	20000220 00000000 00000000 00000000 000000			
ist]									
	Over	↓ Next	Return	Ċ Up 🗌	🕨 Go	o 🛛 🚺 Break	Mode 🕅	Find:	main.c
<pre>3 Source 0S_STK stk_MainTask[APP_TASK_START_STK_SIZE] = {0}; 0S_STK stk_US0Task[APP_TASK_START_STK_SIZE] = {0}; static void MainTask(void *p_arg) 4</pre>									
	<pre>p_arg = p_arg;</pre>								
27	7 printf("\n\rSYS info:\n\r");								
29	9 printf("PCK = %10dHz\n\r", BOARD_MCK * 2);								
31	pri	ntf("MCK =	%10dHz`	\n\r", BOAR	D_MCK	);			
33 34 36	<pre>33 if (CP15_Is_I_CacheEnabled()) 34 printf(" I Cache Enabled\n\r"); else 26 mintf(" I Cache Dischlad\r\r");</pre>								
首先	运行——	上代石。				- /)			

单步运行一些代码:

41	<pre>printf(" D Cache Disabled\n\r");</pre>
43	if (CP15_Is_MMUEnabled())
44	printf(" MMU Enabled\n\r");
46	else printf(" MMU Disabled\n\r");
48	<pre>printf(BANNER);</pre>
50	LED_Configure(1);
53	<pre>#if OS_TASK_STAT_EN &gt; 0 OSStatInit();</pre>
54	<pre>printf("OSIdleCtrMax = %10d\r\n", OSIdleCtrMax); #endif</pre>

串口中的相应输出:

sys info:	
PCK = 198656000Hz	
MCK = 99328000Hz	
I Cache Enabled	
D Cache Disabled	
MMU Disabled	

当前还在任务 MainTask()中,直接在另一个任务 USOTask()中打个断点:

```
static void US0Task(void *p_arg)
 68
    ł
     // USARTO TXD pin definition.
    #define PIN_TXD_0 {1 << 8, AT91C_BASE_PIOC, AT91C_ID_PIOC, PIO_PERIPH_A, PIO_DEFAULT}</pre>
    // USART0 RXD pin definition.
    #define PIN_RXD_0 {1 << 9, AT91C_BASE_PIOC, AT91C_ID_PIOC, PIO_PERIPH_A, PIO_DEFAULT}</pre>
        static const Pin pinsUSART0[] = {PIN_TXD_0, PIN_RXD_0};
        p_arg = p_arg;
        // init srand
 79
        srand(RTT_GetTime(AT91C_BASE_RTTC));
 81
        LED_Configure(2);
直接 Go,停在了断点处,也就是第二个任务也被调度运行
    static void US0Task(void *p_arg)
 68
    // USARTO TXD pin definition.
    #define PIN_TXD_0
                       {1 << 8, AT91C_BASE_PIOC, AT91C_ID_PIOC, PIO_PERIPH_A, PIO_DEFAULT}
    // USARTO RXD pin definition.
    #define PIN_RXD_0 {1 << 9, AT91C_BASE_PIOC, AT91C_ID_PIOC, PIO_PERIPH_A, PIO_DEFAULT}</pre>
        static const Pin pinsUSART0[] = {PIN_TXD_0, PIN_RXD_0};
        p_arg = p_arg;
        // init srand
 79
        srand(RTT_GetTime(AT91C_BASE_RTTC));
 81
        LED_Configure(2);
注意此时 PC 的值,
       R5
            05050505
                        R13
                             20801068
                                        +0C 0000000
       R6
            06060606
                       R14
                                        +10 4F00000
                             14141414
       R7
            07070707
                       PC
                             20000404
                                        +14 6D542D53
```

位于 SDRAM 中

一般的调试过程就是断点,运行,单步的过程,根据寄存器,内存的数据判断程序的状态,排除可能存在的问题。