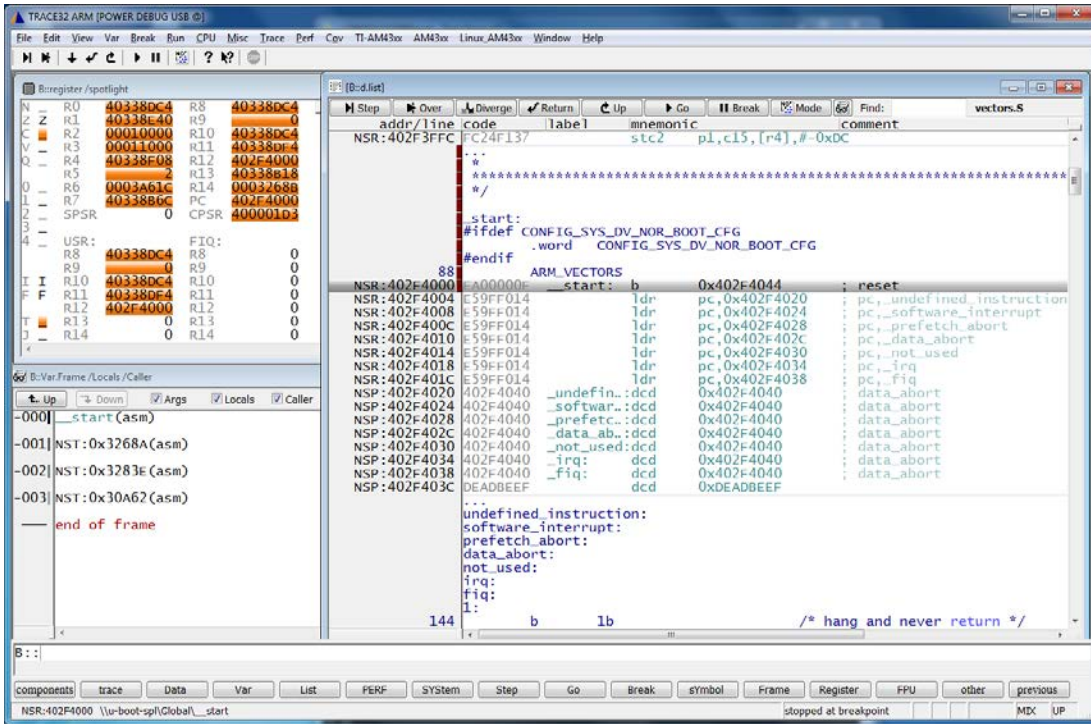
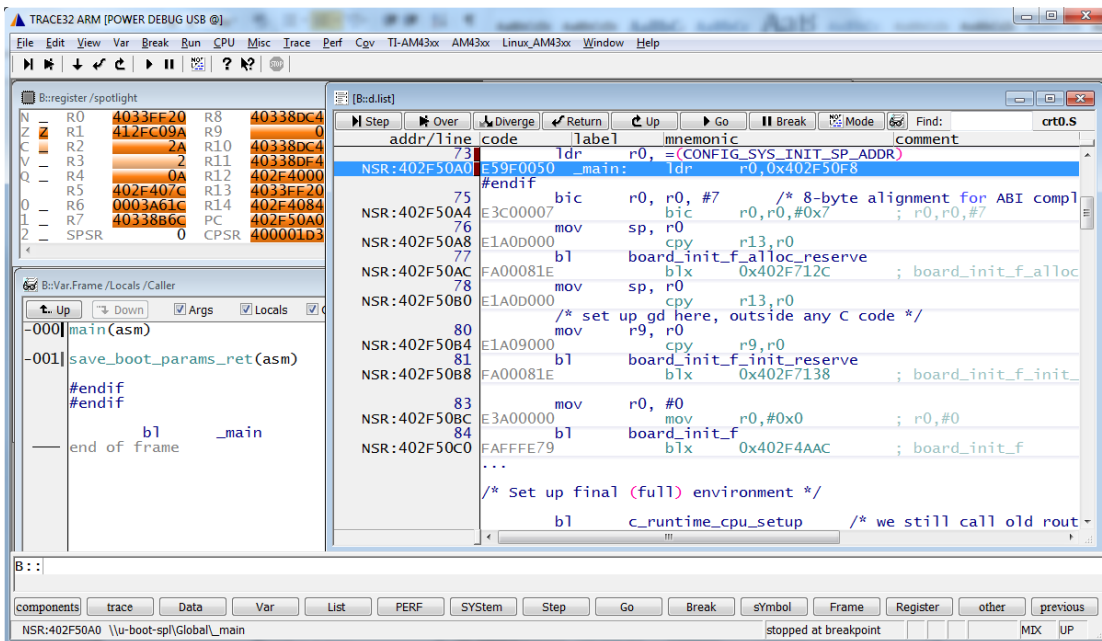


SPL Boot Flow (AM437x GP is used as an example)

1. At the SPL code entry @0x402F4000



2. At the entry of ENTRY(_main) in crt0.S (arch/arm/lib/crt0.S)



3. arch/arm/lib/crt0.S (a good summary of SPL boot flow)

```
/*  
  
* This file handles the target-independent stages of the U-Boot  
* start-up where a C runtime environment is needed. Its entry point  
* is _main and is branched into from the target's start.S file.  
*  
* _main execution sequence is:  
*  
* 1. Set up initial environment for calling board_init_f().  
* This environment only provides a stack and a place to store  
* the GD ('global data') structure, both located in some readily  
* available RAM (SRAM, locked cache...). In this context, VARIABLE  
* global data, initialized or not (BSS), are UNAVAILABLE; only  
* CONSTANT initialized data are available. GD should be zeroed  
* before board_init_f() is called.  
*  
* 2. Call board_init_f(). This function prepares the hardware for  
* execution from system RAM (DRAM, DDR...) As system RAM may not  
* be available yet, board_init_f() must use the current GD to  
* store any data which must be passed on to later stages. These  
* data include the relocation destination, the future stack, and  
* the future GD location.  
*  
* 3. Set up intermediate environment where the stack and GD are the  
* ones allocated by board_init_f() in system RAM, but BSS and  
* initialized non-const data are still not available.  
*  
* 4a. For U-Boot proper (not SPL), call relocate_code(). This function  
* relocates U-Boot from its current location into the relocation  
* destination computed by board_init_f().  
*  
* 4b. For SPL, board_init_f() just returns (to crt0). There is no  
* code relocation in SPL.  
*  
* 5. Set up final environment for calling board_init_r(). This  
* environment has BSS (initialized to 0), initialized non-const  
* data (initialized to their intended value), and stack in system  
* RAM (for SPL moving the stack and GD into RAM is optional - see  
* CONFIG_SPL_STACK_R). GD has retained values set by board_init_f().  
*  
* 6. For U-Boot proper (not SPL), some CPUs have some work left to do  
* at this point regarding memory, so call c_runtime_cpu_setup.  
*  
* 7. Branch to board_init_r().  
*  
* For more information see 'Board Initialisation Flow' in README.  
*/
```

4. Boot flow details

a) arch/arm/lib/crt0.S

```
main (asm)
{
board_init_f()
board_init_r()
}
```

b) core board file (/arch/arm/mach-omap2/am33xx/board.c)

```
board_init_f()
{
early_system_init();
board_early_init_f();
sdram_init(); /* DDR */
gd->ram_size = get_ram_size();
}
```

```
board_early_init_f()
{
prcm_init(); /* PMIC, DPLL */
set_mux_conf_regs(); /* PIN_MUX */
}
```

c) AM43xx board file (/board/ti/am43xx/board.c)

```
sdram_init()-> config_ddr()
```

d) core DDR files

```
/arch/arm/mach-omap2/am33xx/ddr.c
/arch/arm/mach-omap2/am33xx/emif4.c
config_ddr()
```

e) common SPL file (common/spl/spl.c)

```
board_init_r()-> spl_board_init() ..... load/run u-boot @0x80800000 in DDR
```

f) (arch/arm/mach-omap2/boot-common.c)

```
spl_board_init()
{
preloader_console_init(); /* SPL banner output */
}
```

5. At the entry of board_init_r()

The screenshot shows the TRACE32 ARM debugger interface. The main window displays the assembly code for the `board_init_r` function. The register window on the left shows the state of registers R0 through R15, with R0 containing `81FFFFFF20`. The local variable window shows the arguments passed to the function: `dummy1 = 0x81FFFFFF20` and `dummy2 = 0x0`. The assembly code is as follows:

```
NST:402F5F32 0000      movs    r0,r0
;
; #endif
566 void board_init_r(gd_t *dummy1, ulong dummy2)
{
568      u32 spl_boot_list[] = {
568      board_init_r:  push    {r7,r14}
NST:402F5F34 B090      sub     sp,sp,#0x40
NST:402F5F36 B214      movs   r2,#0x14
NST:402F5F38 2100      movs   r1,#0x0           ; dummy2,#0
NST:402F5F3C A802      add    r0,sp,#0x8       ; dummy1,sp,#8
NST:402F5F3E F7FF8EF  bl     0x402F5120       ; memset
;
; BOOT_DEVICE_NONE,
; BOOT_DEVICE_NONE,
; BOOT_DEVICE_NONE,
; BOOT_DEVICE_NONE,
; BOOT_DEVICE_NONE,
;
576      struct spl_image_info spl_image;
578      int ret;
;
; debug(">>" SPL_TPL_PROMPT "board_init_r()\n");
580      spl_set_bd();
NST:402F5F42 F7FFFA3  bl     0x402F5E8C       ; spl_set_bd
;
; #if !(defined(CONFIG_SYS_ICACHE_OFF) && defined(CONFIG_SYS_DCACHE_
;
; #endif
;
; }
```

6. Right before switching to u-boot, still in board_init_r()

The screenshot shows the TRACE32 ARM debugger interface. The main window displays the assembly code for the `board_init_r` function. The register window on the left shows the state of registers R0 through R15, with R0 containing `81FFFFFF4`. The local variable window shows the arguments passed to the function: `dummy1 = 0x81FFFFFF4` and `dummy2 = 0x701FE019`. The assembly code is as follows:

```
690      debug("loaded - jumping to U-Boot...\n");
691      spl_board_prepare_for_boot();
NST:402F5FD2 F7FFFA0  bl     0x402F5DEA       ; spl_board_prepare_for
692      jump_to_image_no_args(&spl_image);
NST:402F5FD6 A807      add    r0,sp,#0x1c
NST:402F5FD8 F7FFFE7A  bl     0x402F4CD0       ; jump_to_image_no_arg
530      printf("Trying to boot from %s\n", loader->name);
NST:402F5DC 6829      ldr   r1,[r5]
NST:402F5DE 4815      ldr   r0,0x402F6034
NST:402F5FE0 F006AB7  bl     0x402FC552       ; printf
;
; }
; /* Not found */
; return NULL;
;
500 static int spl_load_image(struct spl_image_info *spl_image,
; struct spl_image_loader *loader)
; }
```

7. At the entry of u-boot @0x8080000 in DDR

The screenshot displays the TRACE32 ARM debugger interface. The main window shows the disassembly of the code starting at address 0x8080000. The register window on the left shows the current state of the registers, with R0 through R14 and SPSR. The stack window at the bottom shows the current stack frame, which is the entry point of the u-boot image copy.

Register / Spotlight:

Register	Value
R0	40338DC4
R1	701FE019
R2	40337A04
R3	80800000
R4	4032599C
R5	0
R6	81FFFFFF00
R7	403259FC
SPSR	0
USR	0
R8	0
R9	81FFFFFF20
R10	0
R11	0
R12	81FFFFFF9C
R13	0
R14	0
SPSR	0

Disassembly:

```
#ifndef CONFIG_SYS_DV_NOR_BOOT_CFG
.word CONFIG_SYS_DV_NOR_BOOT_CFG
#endif

NSR:80800000 EA0000BE b reset image_start 0x80800300 : reset
NSR:80800004 E59FF014 ldr pc, _undefined_instruction : pc,_undefined_instruction
NSR:80800008 E59FF014 ldr pc, _software_interrupt : pc,_software_interrupt
NSR:8080000C E59FF014 ldr pc, _prefetch_abort : pc,_prefetch_abort
NSR:80800010 E59FF014 ldr pc, _data_abort : pc,_data_abort
NSR:80800014 E59FF014 ldr pc, _not_used : pc,_not_used
NSR:80800018 E59FF014 ldr pc, _irq : pc,0x80800030 : pc,_irq
NSR:8080001C E59FF014 ldr pc, _fiq : pc,0x80800034 : pc,_fiq
NSP:80800020 80800060 _undefined:dcd 0x80800060 : undefined_instruction
NSP:80800024 808000C0 _software:dcd 0x808000C0 : software_interrupt
NSP:80800028 80800120 _prefetc:dcd 0x80800120 : prefetch_abort
NSP:8080002C 80800180 _data_ab:dcd 0x80800180 : data_abort
NSP:80800030 808001E0 _not_used:dcd 0x808001E0 : not_used
NSP:80800034 80800240 _irq:dcd 0x80800240 : irq
NSP:80800038 808002A0 _fiq:dcd 0x808002A0 : fiq
NSP:8080003C DEADBEEF dcd 0xDEADBEEF :
NSP:80800040 0BAD00DE dcd 0x0BAD00DE : IRQ_STAC...
NSR:80800044 E32F0000 nop
NSR:80800048 E32F0000 nop
NSR:8080004C E32F0000 nop
NSR:80800050 E32F0000 nop
NSR:80800054 E32F0000 nop
```

Stack:

```
B:Var.Frame/Locals/Caller
-000|__image_copy_start(asm)
-001|NST:0x403026B6(asm)
end of frame
```