Freescale Semiconductor
Application Note

Document Number: AN4876

CodeWarrior U-Boot Debugging

1. Introduction

This document describes the steps required for U-Boot debugging using the CodeWarrior IDE.

This document includes the following sections:

- Configuring and building U-Boot.
- Creating a CodeWarrior project to debug U-boot.
- Specifying the launch configuration settings.
- Debugging U-Boot from NOR, NAND, SPI, and SD card flash devices for low-end and high-end Power Architecture CPU.

Contents

2. Preliminary background

U-Boot resides in flash memory on target systems and boots an embedded Linux image or other OS image (vxworks) or an elf, developed for those systems.

Before debugging U-Boot on a target system, follow these steps:

1. Install the Board Support Package (BSP) for a target system you want to debug on the host Linux machine.
2. Configure the BSP U-Boot package to place

debugger symbolic information in the U-Boot binary executable file.

3. Configure hardware to use the U-Boot image. (For more information, see Chapter 7.5.3 of Targeting PA Processor.pdf)
4. Create a new CodeWarrior project that you will use to debug U-Boot on the target system.

3. Configuring and building U-Boot

After installing BSP, configure and build U-Boot images for CodeWarrior debug. For more information on configuring and building U-Boot with CodeWarrior debugger support, see the SDK User Manual.

4. Configuring a CodeWarrior project

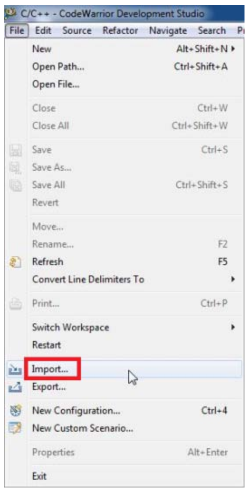
This section covers:

- Creating a CodeWarrior project
- Configure initialization file of project for debugging
- Board hardware configuration

4.1. Creating a CodeWarrior project

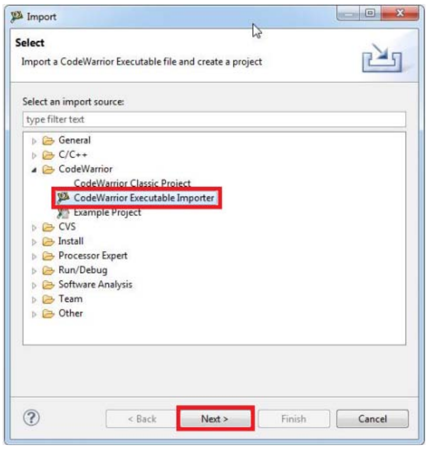
1. Open CodeWarrior IDE.
2. Choose **File** > **Import**, to import the U-Boot .elf file generated during the U-Boot compilation. It can be found in u-boot folder.

Figure 1. CodeWarrior File menu



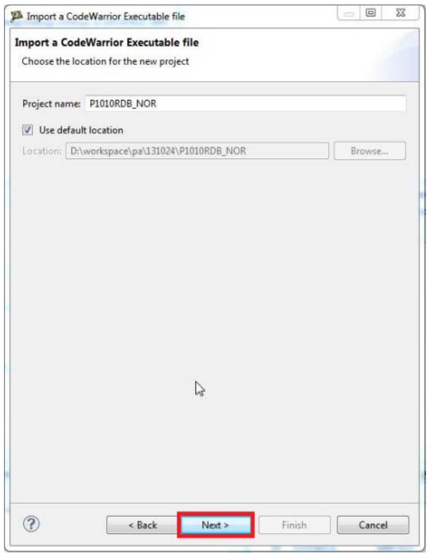
3. Choose the source to import and select **Next**.

Figure 2. Import executable file dialog



4. Specify **Project name** and **Location**, or use the default location and select **Next**.

Figure 3. Import executable file dialog


Import a CodeWarrior Executable file
Import a CodeWarrior Executable file
Choose the location for the new project
< Back
Next >
Finish
Cancel
Browse...

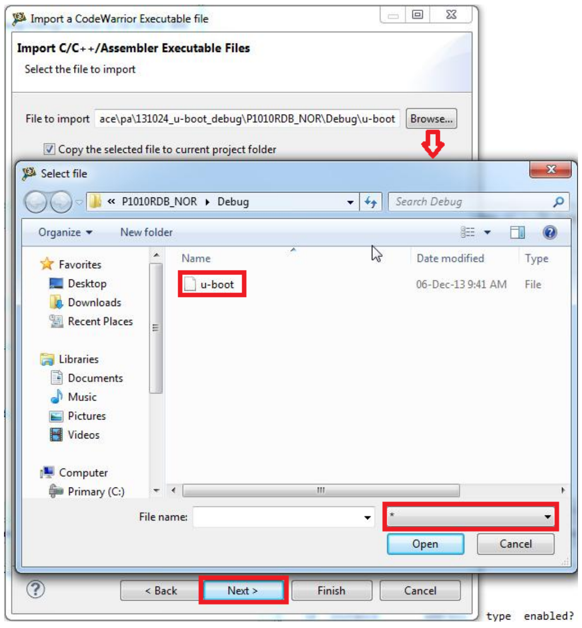
Project name: P1010RDB_NOR

☑ Use default location

Location: D:\workspace\pa\131024\P1010RDB_NOR

5. Browse to the U-Boot .elf file and select open. By default, CodeWarrior looks for an .elf extension, so change the file type in lower right corner of select file dialog, as shown in Figure 4.

Figure 4. Select U-Boot elf file



File to import: ace\pa\131024_u-boot_debug\P1010RDB_NOR\Debug\u-boot

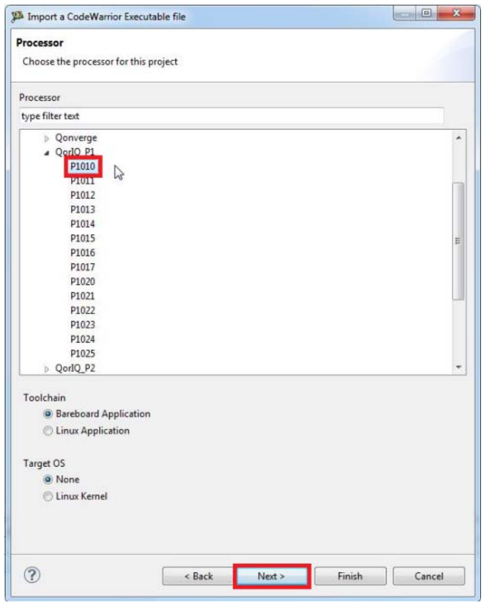
[✓] Copy the selected file to current project folder

u-boot   06-Dec-13 9:41 AM   File

File type: *

6. Select processor type for the project and select **Next**.

Figure 5. Select processor type



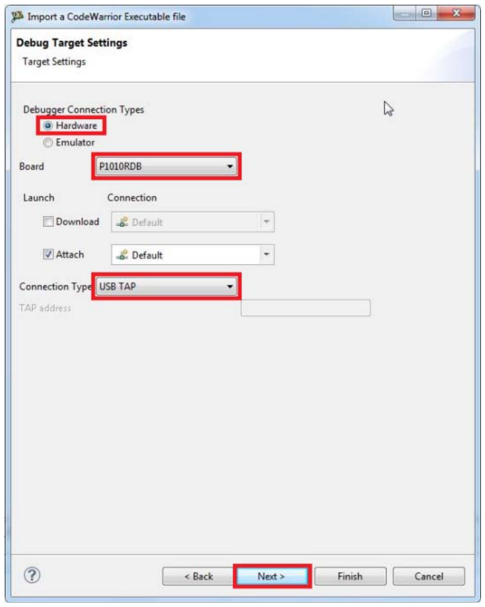
Processor: P1010

Toolchain: [● Bareboard Application] ○ Linux Application

Target OS: [● None] ○ Linux Kernel

7. Select **Debugger Connection Types**, **Board**, and **Connection Type**.

Figure 6. Target settings dialog



Debug Target Settings
Target Settings

Debugger Connection Types
[✓ Hardware]
Emulator

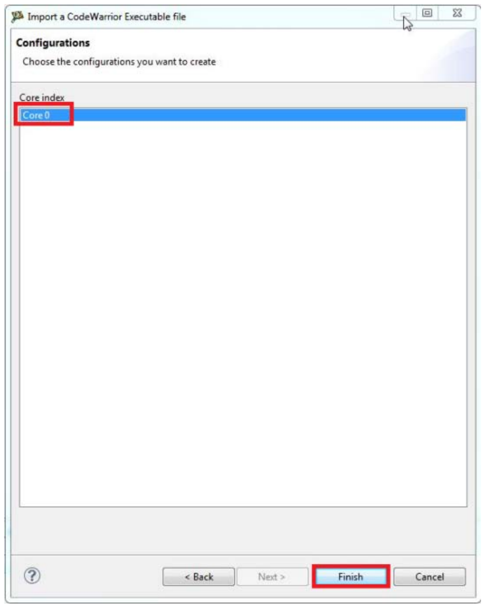
Board P1010RDB

Launch Connection
Download Default
[✓ Attach] Default

Connection Type USB TAP
TAP address

8. Select the configuration that you want to create and then, select **Finish** to close the wizard.

Figure 7. Select configuration dialog



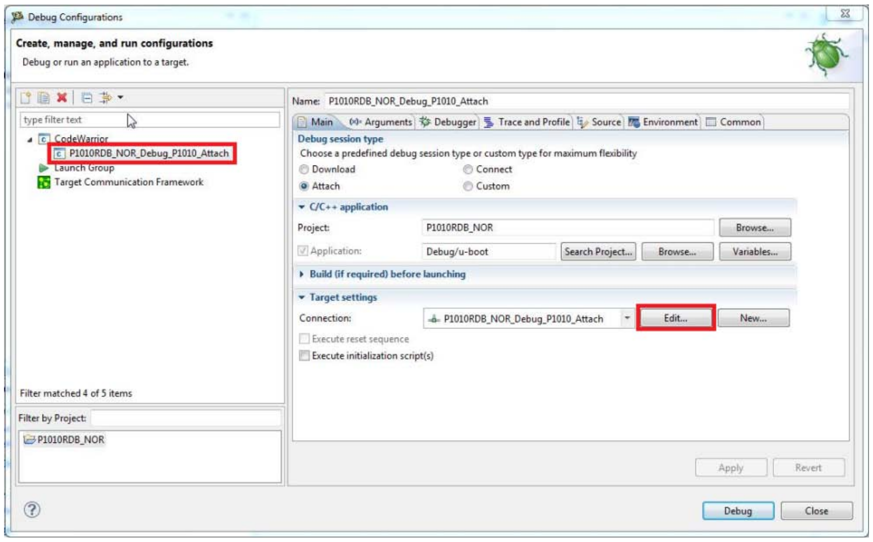
Configurations
Choose the configurations you want to create

Core index
[✓ Core 0]

4.2. Configure initialization file of project for debugging

1. Choose **Run** > **Debug configurations**, to open the **Debug configurations** dialog.
2. Select Project name from the left pane and from the right pane, under Main tab – **Target settings**, select **Edit**, as shown in Figure 8.

Figure 8. Debug Configurations dialog



Name: P1010RDB_NOR_Debug_P1010_Attach

Debug session type

Choose a predefined debug session type or custom type for maximum flexibility

Download / Connect / [✓ Attach] / Custom

C/C++ application

Project: P1010RDB_NOR

[✓] Application: Debug/u-boot

Build (if required) before launching

Target settings

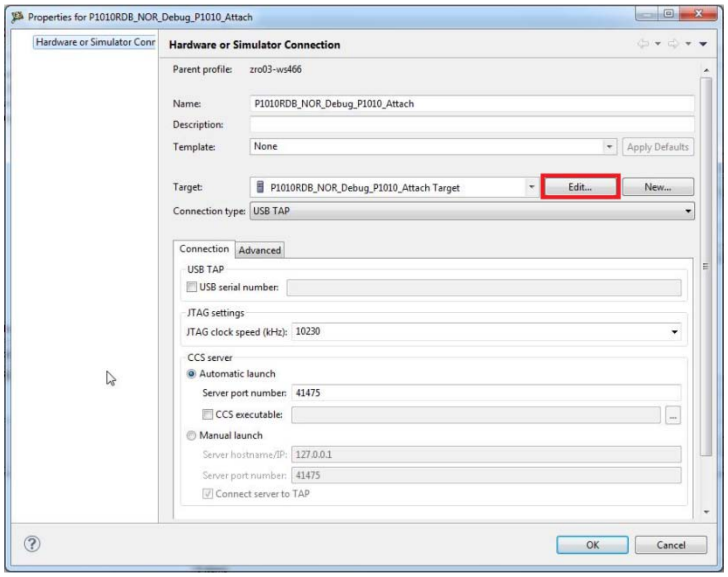
Connection: P1010RDB_NOR_Debug_P1010_Attach

[ ] Execute reset sequence

[ ] Execute initialization script(s)

3. From the **Hardware or Simulator Connection** dialog, select **Edit** to edit the target as shown in the Figure 9.

Figure 9. Hardware or Simulator Connection dialog


Properties for P1010RDB_NOR_Debug_P1010_Attach
Hardware or Simulator Conr
Hardware or Simulator Connection
Apply Defaults
Edit...
New...
Connection
Advanced
USB TAP
JTAG settings
CCS server
OK
Cancel

Parent profile: zro03-ws466

Name: P1010RDB_NOR_Debug_P1010_Attach

Description:

Template: None

Target: P1010RDB_NOR_Debug_P1010_Attach Target

Connection type: USB TAP

[ ] USB serial number:

JTAG clock speed (kHz): 10230

[✓ Automatic launch]

Server port number: 41475

[ ] CCS executable:

Manual launch

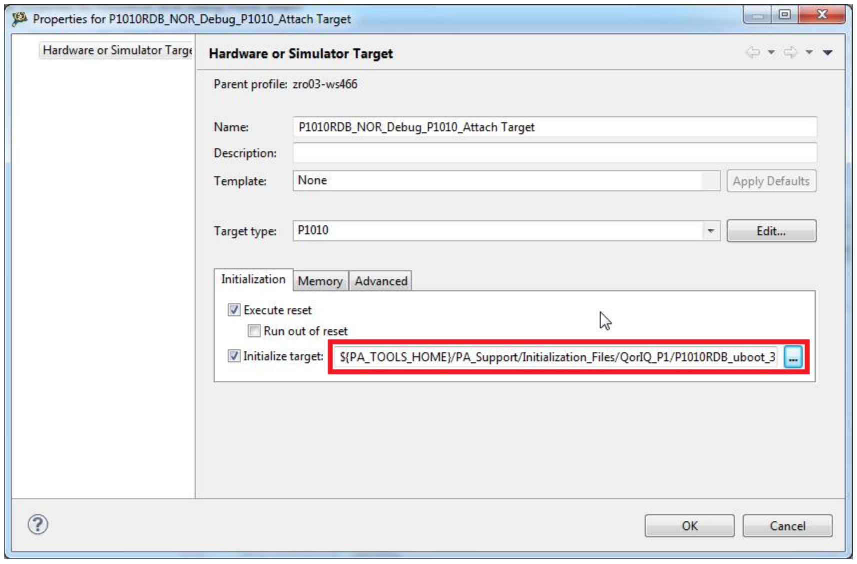
Server hostname/IP: 127.0.0.1

Server port number: 41475

[✓] Connect server to TAP

4. From the **Initialization** tab, browse to the location of U_Boot initialization file and add its location in the **Initialize target**, as shown in Figure 10.

Figure 10. Hardware or Simulator Target dialog



Properties for P1010RDB_NOR_Debug_P1010_Attach Target

Hardware or Simulator Target

Parent profile: zro03-ws466

Name: P1010RDB_NOR_Debug_P1010_Attach Target

Description:

Template: None

Target type: P1010

Initialization | Memory | Advanced

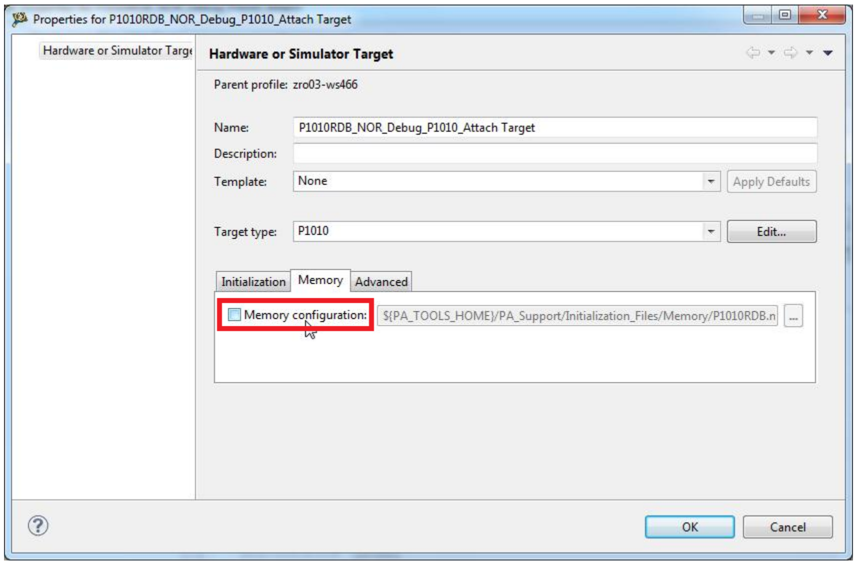
☑ Execute reset

☐ Run out of reset

☑ Initialize target: ${PA_TOOLS_HOME}/PA_Support/Initialization_Files/QorIQ_P1/P1010RDB_uboot_3

5. Navigate to **Memory** tab and deselect **Memory configuration**.

Figure 11. Hardware or Simulator Target dialog



Properties for P1010RDB_NOR_Debug_P1010_Attach Target

Hardware or Simulator Target

Parent profile: zro03-ws466

Name: P1010RDB_NOR_Debug_P1010_Attach Target

Description:

Template: None

Target type: P1010

Initialization | Memory | Advanced

☐ Memory configuration: ${PA_TOOLS_HOME}/PA_Support/Initialization_Files/Memory/P1010RDB.m

6. Select **OK** to exit the **Debug configurations** dialog.

4.3. Board hardware configuration

See the SDK User Guide for the correct board configuration and switch settings.

NOTE U-Boot debug is JTAG-based and a probe needs to be connected to the board.

4.4. Useful hints and tips

Refer to Chapter 12, for useful hints and tips.

5. Debugging U-Boot from NOR for e500v2

5.1. Debug environment

Use the following setup for U-Boot debugging on e500v2 core:

- P1010RDB board.
- Compiled U-Boot for the NOR FLASH target.
- Flash U-Boot on the target board. (See SDK documentation, for more information on how to program the U-Boot to NOR flash.)
- Switches set for NOR boot. (See SDK documentation, for more information on how to set switches.)
- Latest release of CodeWarrior IDE.
- P1010RDB_uboot_32.tcl initialization file.
- USB TAP or other probe.

5.2. U-Boot NOR debugging

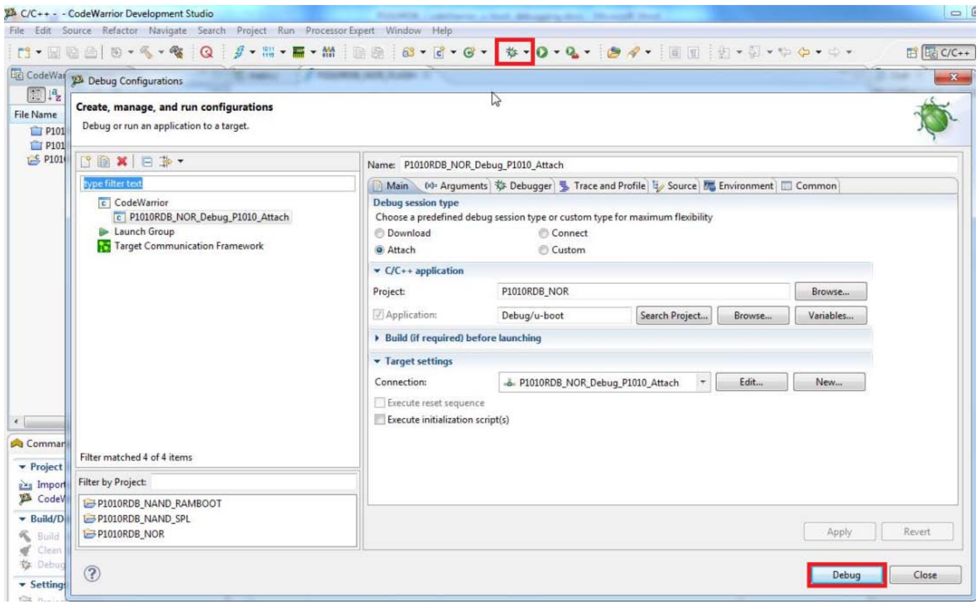
The U-Boot .elf file generated during the U-Boot compilation should be imported as CodeWarrior project. (See Configuring a CodeWarrior project, for more information.)

5.2.1. Stage 0 – Connect CodeWarrior to a board

Before debugging, run the board in the debug mode.

1. Choose **Run** > **Debug configurations**, to open **Debug configurations** dialog and select **Debug**, as shown in Figure 12.

Figure 12. Debug configurations dialog


Debug Configurations

Create, manage, and run configurations

Debug or run an application to a target.

- CodeWarrior
  - P1010RDB_NOR_Debug_P1010_Attach
- Launch Group
- Target Communication Framework

Filter matched 4 of 4 items

Filter by Project:
- P1010RDB_NAND_RAMBOOT
- P1010RDB_NAND_SPL
- P1010RDB_NOR

Name: P1010RDB_NOR_Debug_P1010_Attach

Main | Arguments | Debugger | Trace and Profile | Source | Environment | Common

Debug session type

Choose a predefined debug session type or custom type for maximum flexibility

( ) Download ( ) Connect

(●) Attach ( ) Custom

C/C++ application

Project: P1010RDB_NOR

Application: Debug/u-boot

Build (if required) before launching

Target settings

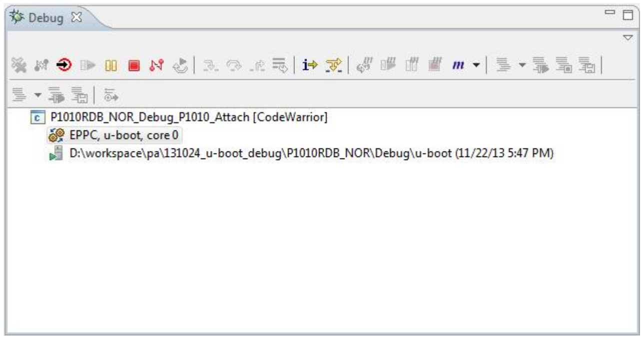
Connection: P1010RDB_NOR_Debug_P1010_Attach

[ ] Execute reset sequence

[ ] Execute initialization script(s)

2. The connection initializes and configures the TAP, and then it will attach to board.

Figure 13. Debug window


Debug

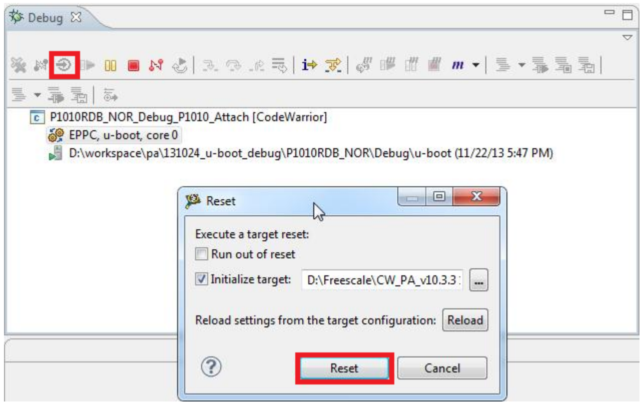
P1010RDB_NOR_Debug_P1010_Attach [CodeWarrior]

EPPC, u-boot, core 0

D:\workspace\pa\131024_u-boot_debug\P1010RDB_NOR\Debug\u-boot (11/22/13 5:47 PM)

3. To reinitialize the target from CodeWarrior, select **Reset**, as shown in Figure 14.

Figure 14. Reset dialog


Debug

P1010RDB_NOR_Debug_P1010_Attach [CodeWarrior]

EPPC, u-boot, core 0

D:\workspace\pa\131024_u-boot_debug\P1010RDB_NOR\Debug\u-boot (11/22/13 5:47 PM)

Reset

Execute a target reset:

[ ] Run out of reset

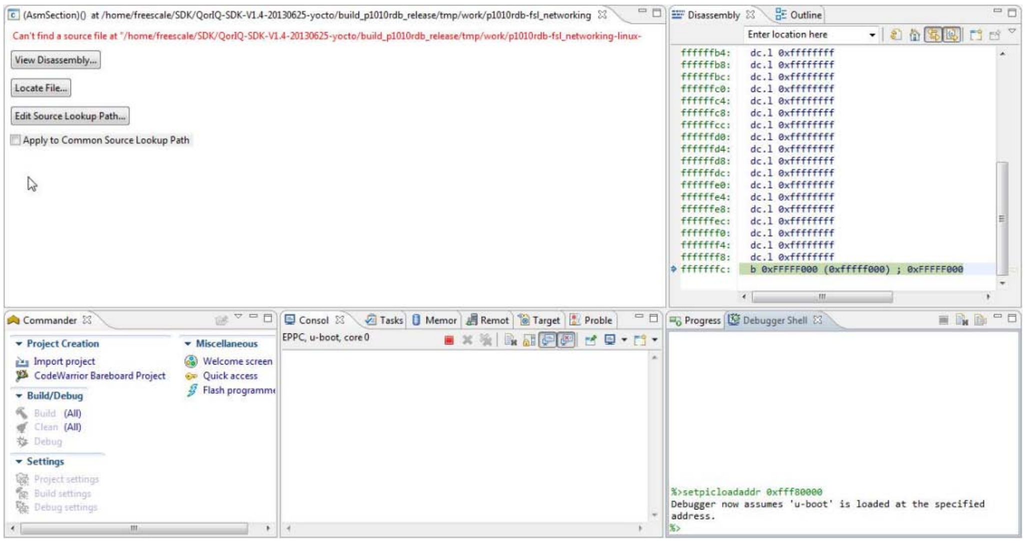
[✓] Initialize target: D:\Freescale\CW_PA_v10.3.3

Reload settings from the target configuration: Reload

5.2.2. Stage 1 – Debug NOR for AS0

1. Set PIC load address as `0xfff80000`, using Debugger Shell command `setpicloadaddr 0xfff80000`.

Figure 15. File location



(AsmSection)() at /home/freescale/SDK/QorIQ-SDK-V1.4-20130625-yocto/build_p1010rdb_release/tmp/work/p1010rdb-fsl_networking

Can't find a source file at "/home/freescale/SDK/QorIQ-SDK-V1.4-20130625-yocto/build_p1010rdb_release/tmp/work/p1010rdb-fsl_networking-linux-

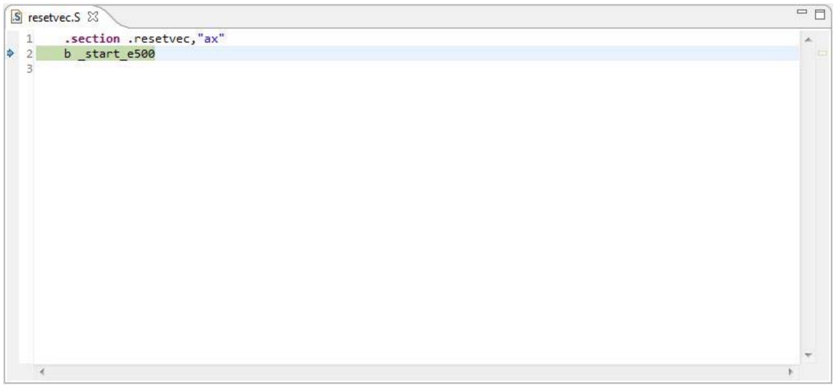
```
ffffffb4:   dc.l 0xffffffff
ffffffb8:   dc.l 0xffffffff
ffffffbc:   dc.l 0xffffffff
ffffffc0:   dc.l 0xffffffff
ffffffc4:   dc.l 0xffffffff
ffffffc8:   dc.l 0xffffffff
ffffffcc:   dc.l 0xffffffff
ffffffd0:   dc.l 0xffffffff
ffffffd4:   dc.l 0xffffffff
ffffffd8:   dc.l 0xffffffff
ffffffdc:   dc.l 0xffffffff
ffffffe0:   dc.l 0xffffffff
ffffffe4:   dc.l 0xffffffff
ffffffe8:   dc.l 0xffffffff
ffffffec:   dc.l 0xffffffff
fffffff0:   dc.l 0xffffffff
fffffff4:   dc.l 0xffffffff
fffffff8:   dc.l 0xffffffff
fffffffc:   b 0xFFFFF000 (0xfffff000) ; 0xFFFFF000
```

EPPC, u-boot, core 0

```
%>setpicloadaddr 0xfff80000
Debugger now assumes 'u-boot' is loaded at the specified
address.
%>
```

2. After the path is provided, source will become available in CodeWarrior. (See Figure 15, for more details.)

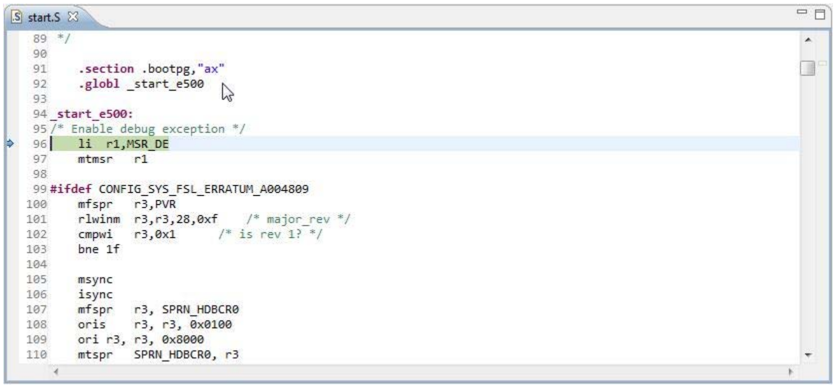
Figure 16. File editor



```
    .section .resetvec,"ax"
    b _start_e500
```

3. Now debugging (step, run, or breakpoint) can be done before switching to AS1.

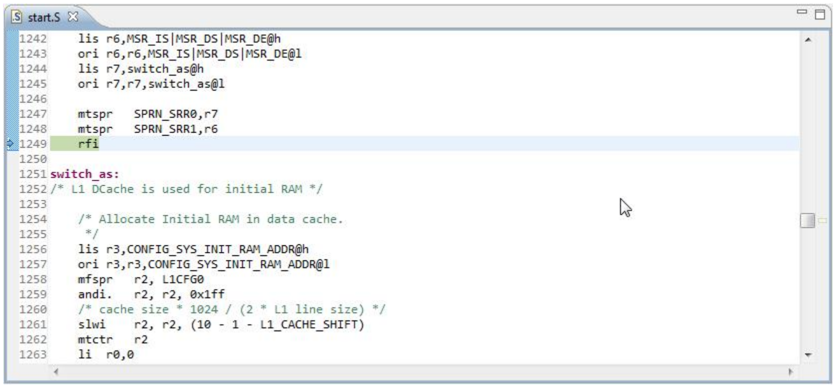
Figure 17. File editor



```
*/

    .section .bootpg,"ax"
    .globl _start_e500

_start_e500:
/* Enable debug exception */
    li  r1,MSR_DE
    mtmsr   r1

#ifdef CONFIG_SYS_FSL_ERRATUM_A004809
    mfspr   r3,PVR
    rlwinm  r3,r3,28,0xf    /* major_rev */
    cmpwi   r3,0x1      /* is rev 1? */
    bne 1f

    msync
    isync
    mfspr   r3, SPRN_HDBCR0
    oris    r3, r3, 0x0100
    ori r3, r3, 0x8000
    mtspr   SPRN_HDBCR0, r3
```

4. In file start.S, last instruction before moving to AS1 is `rfi` before `switch_as`. (See Stage 2 – Debug NOR for AS1, for more information.)

Figure 18. File editor

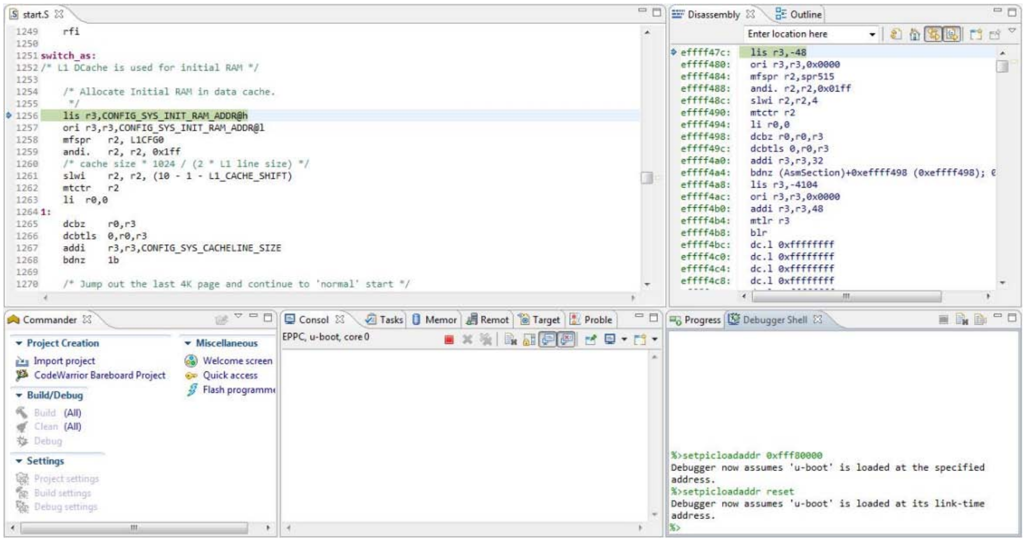


```
    lis r6,MSR_IS|MSR_DS|MSR_DE@h
    ori r6,r6,MSR_IS|MSR_DS|MSR_DE@l
    lis r7,switch_as@h
    ori r7,r7,switch_as@l

    mtspr   SPRN_SRR0,r7
    mtspr   SPRN_SRR1,r6
    rfi

switch_as:
/* L1 DCache is used for initial RAM */

    /* Allocate Initial RAM in data cache.
     */
    lis r3,CONFIG_SYS_INIT_RAM_ADDR@h
    ori r3,r3,CONFIG_SYS_INIT_RAM_ADDR@l
    mfspr   r2, L1CFG0
    andi.   r2, r2, 0x1ff
    /* cache size * 1024 / (2 * L1 line size) */
    slwi    r2, r2, (10 - 1 - L1_CACHE_SHIFT)
    mtctr   r2
    li  r0,0
```

5.2.3. Stage 2 – Debug NOR for AS1

1. Step Into `rfi` instruction.
2. Reset PIC load address, using Debugger Shell command `setpicloadaddr reset.`

Figure 19. Debugger shell view



```
    rfi

switch_as:
/* L1 DCache is used for initial RAM */

    /* Allocate Initial RAM in data cache.
     */
    lis r3,CONFIG_SYS_INIT_RAM_ADDR@h
    ori r3,r3,CONFIG_SYS_INIT_RAM_ADDR@l
    mfspr   r2, L1CFG0
    andi.   r2, r2, 0x1ff
    /* cache size * 1024 / (2 * L1 line size) */
    slwi    r2, r2, (10 - 1 - L1_CACHE_SHIFT)
    mtctr   r2
    li  r0,0
1:
    dcbz    r0,r3
    dcbtls  0,r0,r3
    addi    r3,r3,CONFIG_SYS_CACHELINE_SIZE
    bdnz    1b

    /* Jump out the last 4K page and continue to 'normal' start */
```

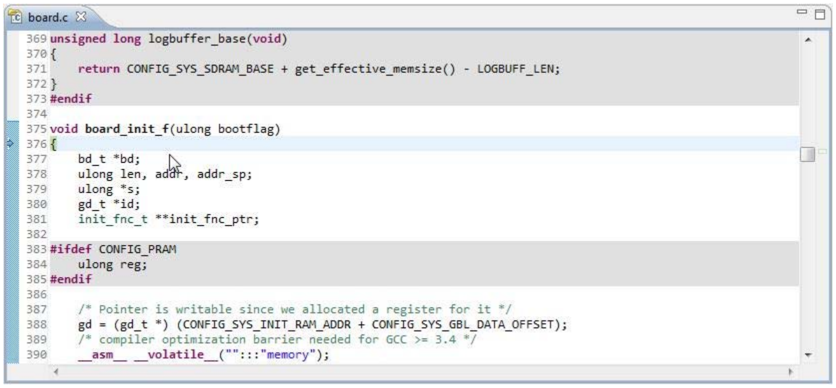
```
effff47c:  lis r3,-48
effff480:  ori r3,r3,0x0000
effff484:  mfspr r2,spr515
effff488:  andi. r2,r2,0x01ff
effff48c:  slwi r2,r2,4
effff490:  mtctr r2
effff494:  li r0,0
effff498:  dcbz r0,r0,r3
effff49c:  dcbtls 0,r0,r3
effff4a0:  addi r3,r3,32
effff4a4:  bdnz (AsmSection)+0xeffff498 (0xeffff498); 0
effff4a8:  lis r3,-4104
effff4ac:  ori r3,r3,0x0000
effff4b0:  addi r3,r3,48
effff4b4:  mtlr r3
effff4b8:  blr
effff4bc:  dc.l 0xffffffff
effff4c0:  dc.l 0xffffffff
effff4c4:  dc.l 0xffffffff
effff4c8:  dc.l 0xffffffff
```

EPPC, u-boot, core 0

```
%>setpicloadaddr 0xfff80000
Debugger now assumes 'u-boot' is loaded at the specified
address.
%>setpicloadaddr reset
Debugger now assumes 'u-boot' is loaded at its link-time
address.
%>
```

3. Debugging (step, run, or breakpoint) can be done until code is relocated in DDR.
   a) Run to Line: `board_init_f` and Step Into.

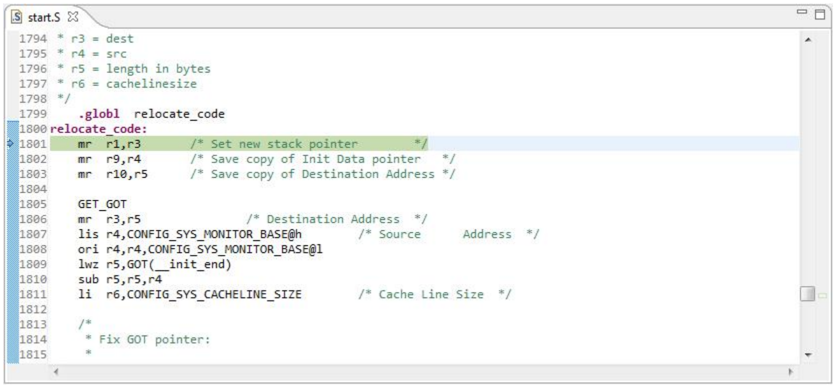
Figure 20. File editor



```
unsigned long logbuffer_base(void)
{
    return CONFIG_SYS_SDRAM_BASE + get_effective_memsize() - LOGBUFF_LEN;
}
#endif

void board_init_f(ulong bootflag)
{
    bd_t *bd;
    ulong len, addr, addr_sp;
    ulong *s;
    gd_t *id;
    init_fnc_t **init_fnc_ptr;

#ifdef CONFIG_PRAM
    ulong reg;
#endif

    /* Pointer is writable since we allocated a register for it */
    gd = (gd_t *) (CONFIG_SYS_INIT_RAM_ADDR + CONFIG_SYS_GBL_DATA_OFFSET);
    /* compiler optimization barrier needed for GCC >= 3.4 */
    __asm__ __volatile__("":::"memory");
```

   b) Run to Line: `relocate_code` and Step Into.

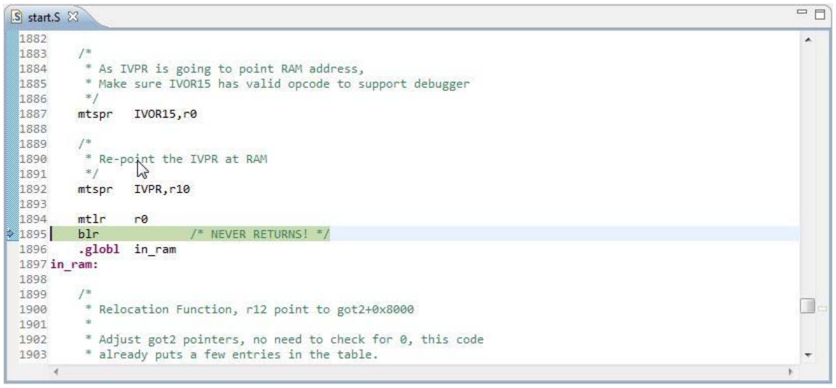
Figure 21. File editor



```
 * r3 = dest
 * r4 = src
 * r5 = length in bytes
 * r6 = cachelinesize
 */
    .globl  relocate_code
relocate_code:
    mr   r1,r3      /* Set new stack pointer        */
    mr   r9,r4      /* Save copy of Init Data pointer   */
    mr   r10,r5     /* Save copy of Destination Address */

    GET_GOT
    mr   r3,r5              /* Destination Address  */
    lis r4,CONFIG_SYS_MONITOR_BASE@h        /* Source      Address  */
    ori r4,r4,CONFIG_SYS_MONITOR_BASE@l
    lwz r5,GOT(__init_end)
    sub r5,r5,r4
    li   r6,CONFIG_SYS_CACHELINE_SIZE       /* Cache Line Size  */

    /*
     * Fix GOT pointer:
     *
```

c) In start.S, last instruction before relocate to DDR is `relocate_code`.

Figure 22. File editor



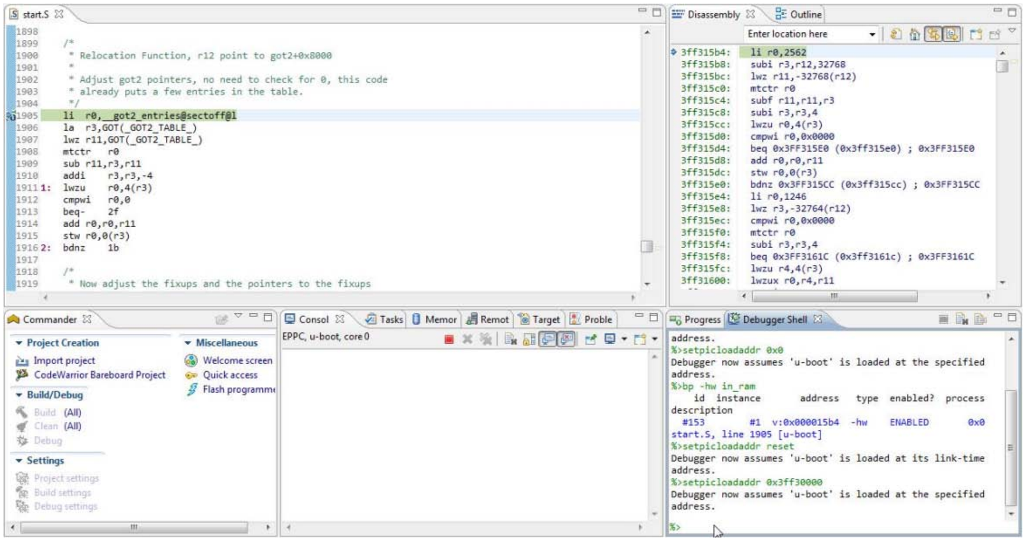
```
    /*
     * As IVPR is going to point RAM address,
     * Make sure IVOR15 has valid opcode to support debugger
     */
    mtspr   IVOR15,r0

    /*
     * Re-point the IVPR at RAM
     */
    mtspr   IVPR,r10

    mtlr    r0
    blr                 /* NEVER RETURNS! */
    .globl  in_ram
in_ram:

    /*
     * Relocation Function, r12 point to got2+0x8000
     *
     * Adjust got2 pointers, no need to check for 0, this code
     * already puts a few entries in the table.
```

d) Step Into blr, it shows the code in assembly. (See Stage 3 – Debug in DDR's higher address, for more information.)

5.2.4. Stage 3 – Debug in DDR's higher address

1. Set the PIC load address as `0x3ff30000` using Debugger Shell command `setpicloadaddr 0x3ff30000`. (See How to calculate PIC load address, for more information.)

Figure 23. Debugger shell view



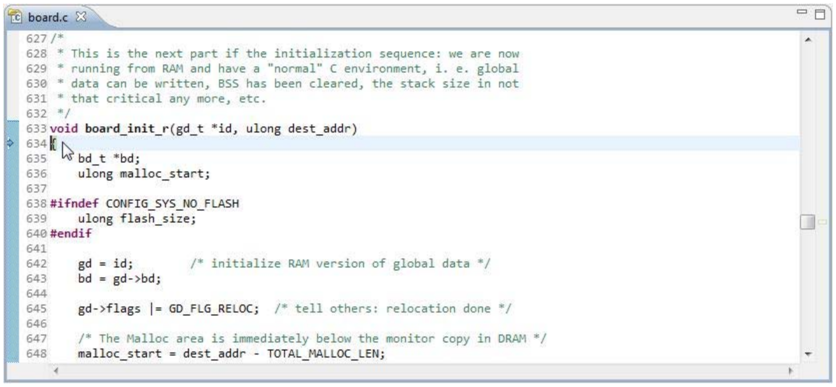
```
1898
1899        /*
1900         * Relocation Function, r12 point to got2+0x8000
1901         *
1902         * Adjust got2 pointers, no need to check for 0, this code
1903         * already puts a few entries in the table.
1904         */
1905        li  r0,__got2_entries@sectoff@l
1906        la  r3,GOT(_GOT2_TABLE_)
1907        lwz r11,GOT(_GOT2_TABLE_)
1908        mtctr   r0
1909        sub r11,r3,r11
1910        addi    r3,r3,-4
1911 1:     lwzu    r0,4(r3)
1912        cmpwi   r0,0
1913        beq-    2f
1914        add r0,r0,r11
1915        stw r0,0(r3)
1916 2:     bdnz    1b
1917
1918        /*
1919         * Now adjust the fixups and the pointers to the fixups
```

```
3ff315b4:   li r0,2562
3ff315b8:   subi r3,r12,32768
3ff315bc:   lwz r11,-32768(r12)
3ff315c0:   mtctr r0
3ff315c4:   subf r11,r11,r3
3ff315c8:   subi r3,r3,4
3ff315cc:   lwzu r0,4(r3)
3ff315d0:   cmpwi r0,0x0000
3ff315d4:   beq 0x3FF315E0 (0x3ff315e0) ; 0x3FF315E0
3ff315d8:   add r0,r0,r11
3ff315dc:   stw r0,0(r3)
3ff315e0:   bdnz 0x3FF315CC (0x3ff315cc) ; 0x3FF315CC
3ff315e4:   li r0,1246
3ff315e8:   lwz r3,-32764(r12)
3ff315ec:   cmpwi r0,0x0000
3ff315f0:   mtctr r0
3ff315f4:   subi r3,r3,4
3ff315f8:   beq 0x3FF3161C (0x3ff3161c) ; 0x3FF3161C
3ff315fc:   lwzu r4,4(r3)
3ff31600:   lwzux r0,r4,r11
```

```
address.
%>setpicloadaddr 0x0
Debugger now assumes 'u-boot' is loaded at the specified
address.
%>bp -hw in_ram
    id  instance       address   type  enabled?  process
description
  #153        #1  v:0x000015b4  -hw    ENABLED       0x0
start.S, line 1905 [u-boot]
%>setpicloadaddr reset
Debugger now assumes 'u-boot' is loaded at its link-time
address.
%>setpicloadaddr 0x3ff30000
Debugger now assumes 'u-boot' is loaded at the specified
address.

%>
```

2. You can debug until U-Boot is running.
   a) Run to Line: `board_init_r` and do Step into.

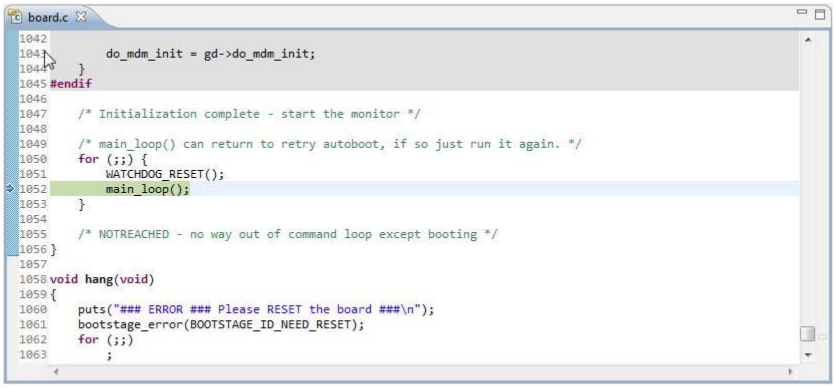
Figure 24. File editor



```
/*
 * This is the next part if the initialization sequence: we are now
 * running from RAM and have a "normal" C environment, i. e. global
 * data can be written, BSS has been cleared, the stack size in not
 * that critical any more, etc.
 */
void board_init_r(gd_t *id, ulong dest_addr)
{
    bd_t *bd;
    ulong malloc_start;

#ifndef CONFIG_SYS_NO_FLASH
    ulong flash_size;
#endif

    gd = id;        /* initialize RAM version of global data */
    bd = gd->bd;

    gd->flags |= GD_FLG_RELOC;  /* tell others: relocation done */

    /* The Malloc area is immediately below the monitor copy in DRAM */
    malloc_start = dest_addr - TOTAL_MALLOC_LEN;
```

b) Run to Line: `main_loop()`

Figure 25. File editor



```

            do_mdm_init = gd->do_mdm_init;
    }
#endif

    /* Initialization complete - start the monitor */

    /* main_loop() can return to retry autoboot, if so just run it again. */
    for (;;) {
        WATCHDOG_RESET();
        main_loop();
    }

    /* NOTREACHED - no way out of command loop except booting */
}

void hang(void)
{
    puts("### ERROR ### Please RESET the board ###\n");
    bootstage_error(BOOTSTAGE_ID_NEED_RESET);
    for (;;)
        ;
```

6. Debugging U-Boot from NAND for e500v2

U-Boot NAND boot is a 2-stage booting process:

- First stage (U-Boot NAND SPL) – when turned on and on reset, U-Boot NAND SPL gets the control. It runs from IFC's internal SRAM and it copies U-Boot RAMBOOT to DDR and transfers control to it.
- Second stage (U-Boot NAND RAMBOOT) – RAMBOOT code.

Depending upon the booting stage, U-Boot NAND debugging can be classified into two modes:

- U-Boot NAND SPL debugging
- U-Boot NAND RAMBOOT debugging

6.1. Debug environment

Use the following setup for U-Boot NAND debugging on e500v2 core:

1. P1010RBD board.
2. Compiled U-Boot for the NAND FLASH target.

3. Flash U-Boot on the target board. (See SDK documentation, for more information on how to program the U-Boot to NAND flash.)
4. Switches set for NAND boot (See SDK documentation, for more information on how to set switches.)
5. Latest release of CodeWarrior IDE.
6. P1010RDB_uboot_32.tcl initialization file.
7. USB TAP or other probe.

6.2. U-Boot NAND SPL debugging

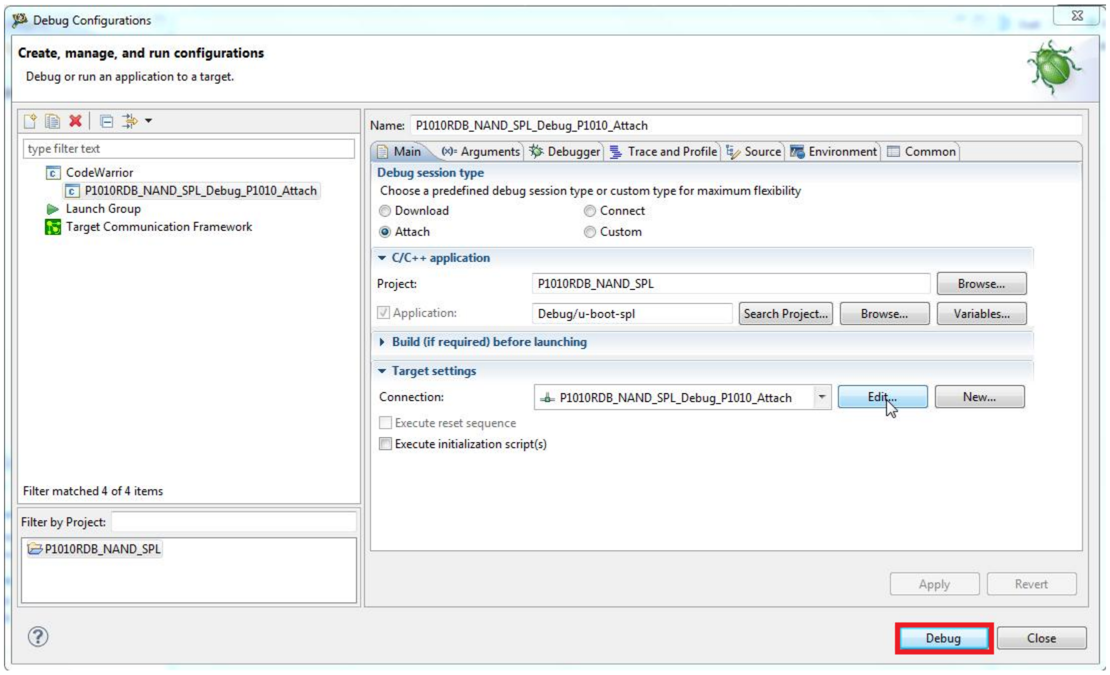
For this stage the U-Boot-spl elf file generated during U-Boot compilation should be imported as a CodeWarrior project. (See Configuring a CodeWarrior project, for more details.)

6.2.1. Stage 0 – Connect CodeWarrior to board

Before starting debugging, run the project in debug mode.

1. Choose **Run** > **Debug** configurations, to open **Debug configurations** dialog and select **Debug**.

Figure 26. Debug configurations dialog



Debug Configurations

Create, manage, and run configurations

Debug or run an application to a target.

type filter text

- CodeWarrior
  - P1010RDB_NAND_SPL_Debug_P1010_Attach
- Launch Group
- Target Communication Framework

Filter matched 4 of 4 items

Filter by Project:

P1010RDB_NAND_SPL

Name: P1010RDB_NAND_SPL_Debug_P1010_Attach

Debug session type

Choose a predefined debug session type or custom type for maximum flexibility

Download | Connect
[✓ Attach] | Custom

C/C++ application

Project: P1010RDB_NAND_SPL

Application: Debug/u-boot-spl

Build (if required) before launching

Target settings

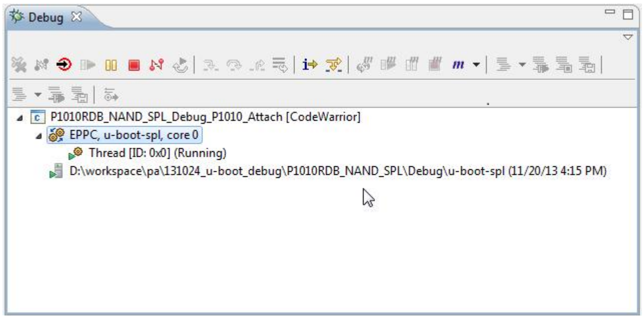
Connection: P1010RDB_NAND_SPL_Debug_P1010_Attach

Execute reset sequence

Execute initialization script(s)

2. This initializes and configures the TAP, then attaches to board.

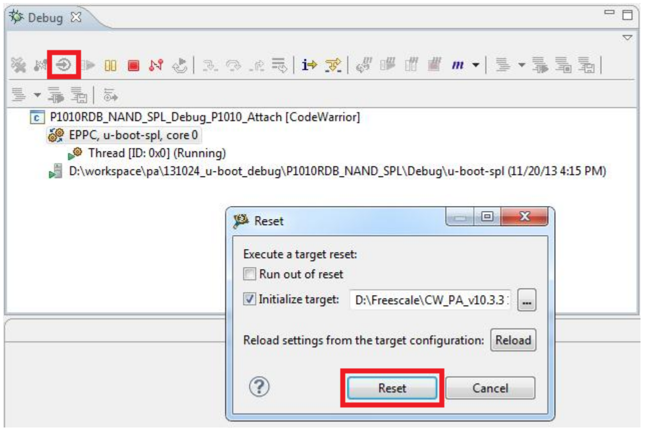
Figure 27. Debug view



P1010RDB_NAND_SPL_Debug_P1010_Attach [CodeWarrior]
EPPC, u-boot-spl, core 0
Thread [ID: 0x0] (Running)
D:\workspace\pa\131024_u-boot_debug\P1010RDB_NAND_SPL\Debug\u-boot-spl (11/20/13 4:15 PM)

3. Reinitialize the target from CodeWarrior, using the U-Boot initialization file.

Figure 28. Reset dialog



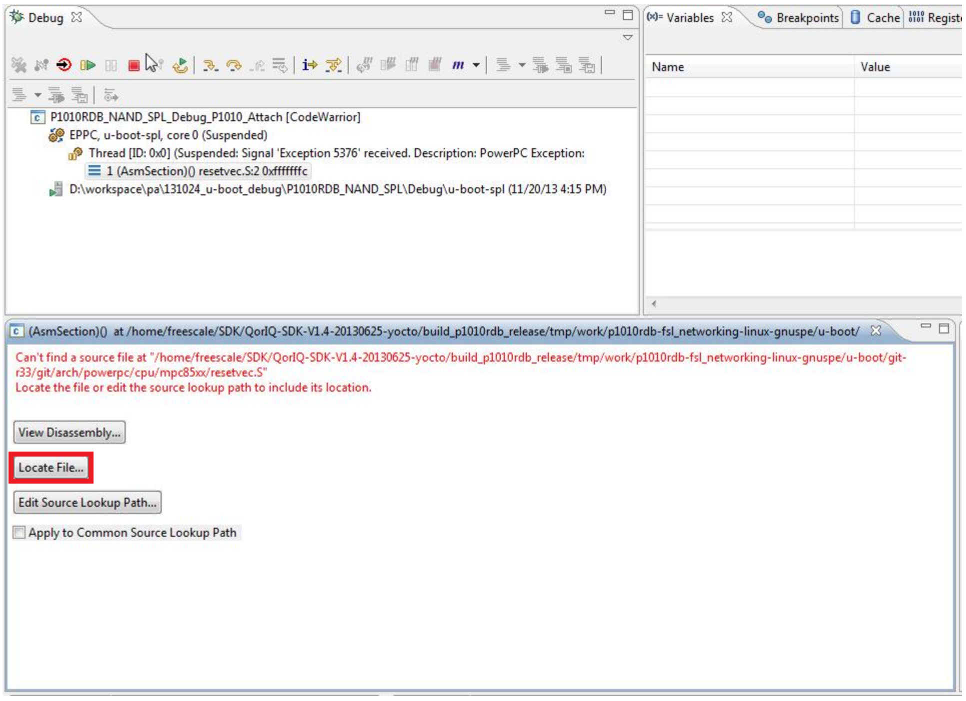
P1010RDB_NAND_SPL_Debug_P1010_Attach [CodeWarrior]
EPPC, u-boot-spl, core 0
Thread [ID: 0x0] (Running)
D:\workspace\pa\131024_u-boot_debug\P1010RDB_NAND_SPL\Debug\u-boot-spl (11/20/13 4:15 PM)

Execute a target reset:
☐ Run out of reset
☑ Initialize target: D:\Freescale\CW_PA_v10.3.3
Reload settings from the target configuration:

6.2.2. Stage 1 – Debug NAND SPL in IFC SRAM for AS0

1. Browse to the location, where the source file is saved, as shown in Figure 29.

Figure 29. Debug view

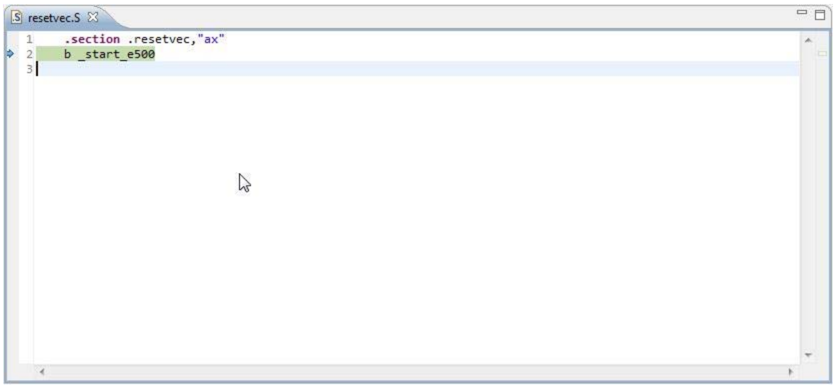


```
P1010RDB_NAND_SPL_Debug_P1010_Attach [CodeWarrior]
  EPPC, u-boot-spl, core 0 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Exception 5376' received. Description: PowerPC Exception:
      1 (AsmSection)() resetvec.S:2 0xfffffffc
  D:\workspace\pa\131024_u-boot_debug\P1010RDB_NAND_SPL\Debug\u-boot-spl (11/20/13 4:15 PM)
```

```
(AsmSection)() at /home/freescale/SDK/QorIQ-SDK-V1.4-20130625-yocto/build_p1010rdb_release/tmp/work/p1010rdb-fsl_networking-linux-gnuspe/u-boot/
Can't find a source file at "/home/freescale/SDK/QorIQ-SDK-V1.4-20130625-yocto/build_p1010rdb_release/tmp/work/p1010rdb-fsl_networking-linux-gnuspe/u-boot/git-r33/git/arch/powerpc/cpu/mpc85xx/resetvec.S"
Locate the file or edit the source lookup path to include its location.
```

2. After the path is specified, the source is available in CodeWarrior.

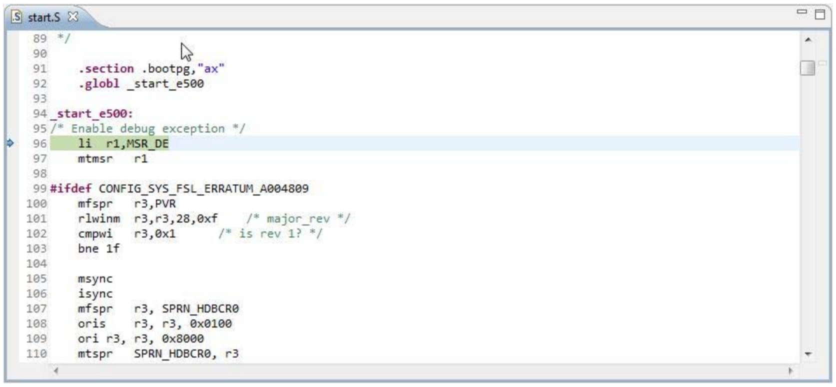
Figure 30. File editor



```
    .section .resetvec,"ax"
    b _start_e500
```

3. Now, debugging (step, run, or breakpoint) can be done before switching to AS1.

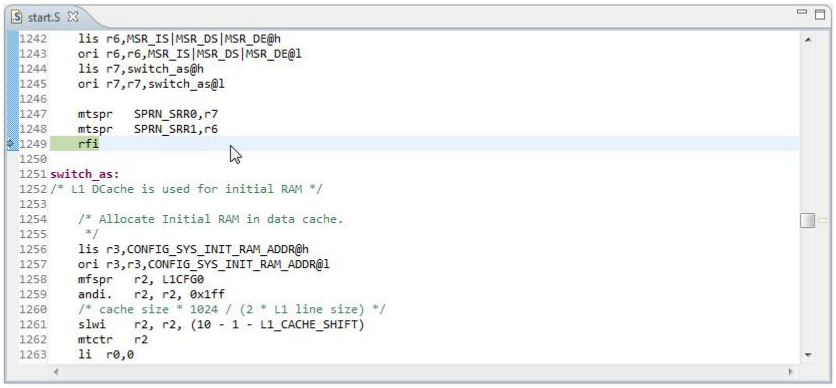
Figure 31. File editor



```
*/

    .section .bootpg,"ax"
    .globl _start_e500

_start_e500:
/* Enable debug exception */
    li  r1,MSR_DE
    mtmsr   r1

#ifdef CONFIG_SYS_FSL_ERRATUM_A004809
    mfspr   r3,PVR
    rlwinm  r3,r3,28,0xf    /* major_rev */
    cmpwi   r3,0x1      /* is rev 1? */
    bne 1f

    msync
    isync
    mfspr   r3, SPRN_HDBCR0
    oris    r3, r3, 0x0100
    ori r3, r3, 0x8000
    mtspr   SPRN_HDBCR0, r3
```

4. In file start.S, last instruction before moving to AS1 is `rfi` before `switch_as`. (See Stage 2 – Debug NAND SPL in IFC SRAM for AS1, for more information.)

Figure 32. File editor

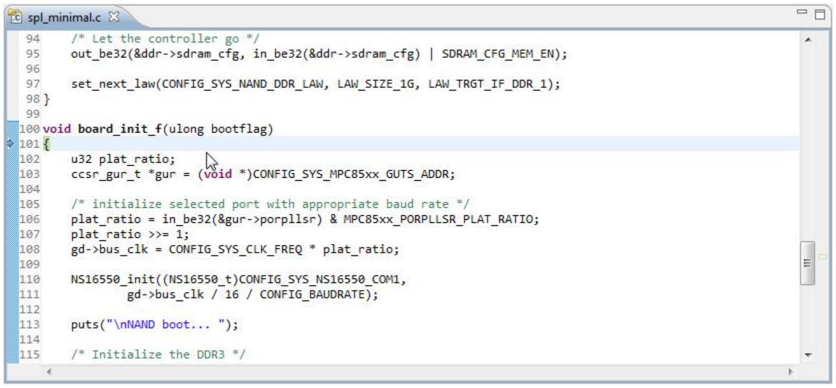


```
    lis r6,MSR_IS|MSR_DS|MSR_DE@h
    ori r6,r6,MSR_IS|MSR_DS|MSR_DE@l
    lis r7,switch_as@h
    ori r7,r7,switch_as@l

    mtspr   SPRN_SRR0,r7
    mtspr   SPRN_SRR1,r6
    rfi

switch_as:
/* L1 DCache is used for initial RAM */

    /* Allocate Initial RAM in data cache.
     */
    lis r3,CONFIG_SYS_INIT_RAM_ADDR@h
    ori r3,r3,CONFIG_SYS_INIT_RAM_ADDR@l
    mfspr   r2, L1CFG0
    andi.   r2, r2, 0x1ff
    /* cache size * 1024 / (2 * L1 line size) */
    slwi    r2, r2, (10 - 1 - L1_CACHE_SHIFT)
    mtctr   r2
    li  r0,0
```

6.2.3. Stage 2 – Debug NAND SPL in IFC SRAM for AS1

1. Step Into this instruction.
2. Debugging is possible until the code is relocated to DDR.
   a) Run to Line: `board_init_f` and Step Into: `board_init_f.`

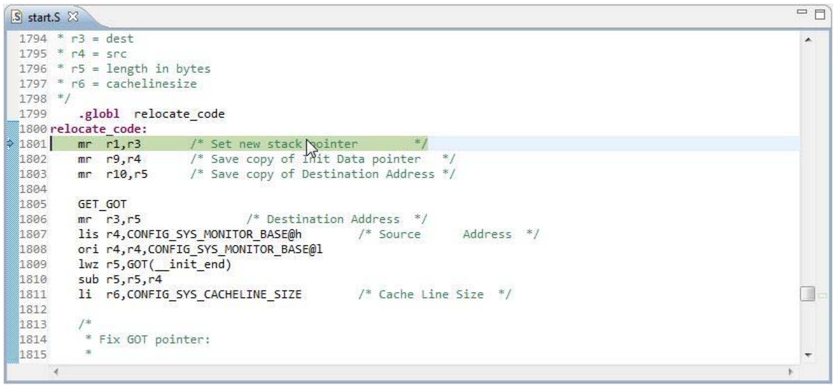
Figure 33. File editor



```
    /* Let the controller go */
    out_be32(&ddr->sdram_cfg, in_be32(&ddr->sdram_cfg) | SDRAM_CFG_MEM_EN);

    set_next_law(CONFIG_SYS_NAND_DDR_LAW, LAW_SIZE_1G, LAW_TRGT_IF_DDR_1);
}

void board_init_f(ulong bootflag)
{
    u32 plat_ratio;
    ccsr_gur_t *gur = (void *)CONFIG_SYS_MPC85xx_GUTS_ADDR;

    /* initialize selected port with appropriate baud rate */
    plat_ratio = in_be32(&gur->porpllsr) & MPC85xx_PORPLLSR_PLAT_RATIO;
    plat_ratio >>= 1;
    gd->bus_clk = CONFIG_SYS_CLK_FREQ * plat_ratio;

    NS16550_init((NS16550_t)CONFIG_SYS_NS16550_COM1,
            gd->bus_clk / 16 / CONFIG_BAUDRATE);

    puts("\nNAND boot... ");

    /* Initialize the DDR3 */
```

b) Run to Line: `relocate_code` and do Step Into.

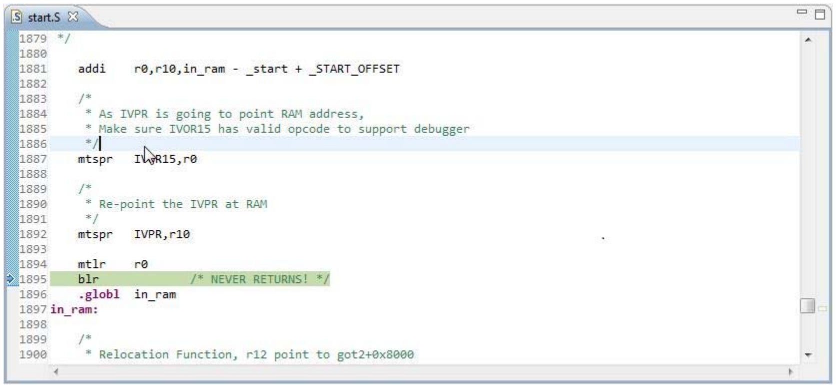
Figure 34. File editor



```
 * r3 = dest
 * r4 = src
 * r5 = length in bytes
 * r6 = cachelinesize
 */
    .globl  relocate_code
relocate_code:
    mr  r1,r3       /* Set new stack pointer        */
    mr  r9,r4       /* Save copy of Init Data pointer    */
    mr  r10,r5      /* Save copy of Destination Address */

    GET_GOT
    mr  r3,r5               /* Destination Address  */
    lis r4,CONFIG_SYS_MONITOR_BASE@h        /* Source      Address  */
    ori r4,r4,CONFIG_SYS_MONITOR_BASE@l
    lwz r5,GOT(__init_end)
    sub r5,r5,r4
    li  r6,CONFIG_SYS_CACHELINE_SIZE        /* Cache Line Size  */

    /*
     * Fix GOT pointer:
     *
```

c) In file start.S, last code before relocate to DDR is `relocate_code`.

Figure 35. File editor



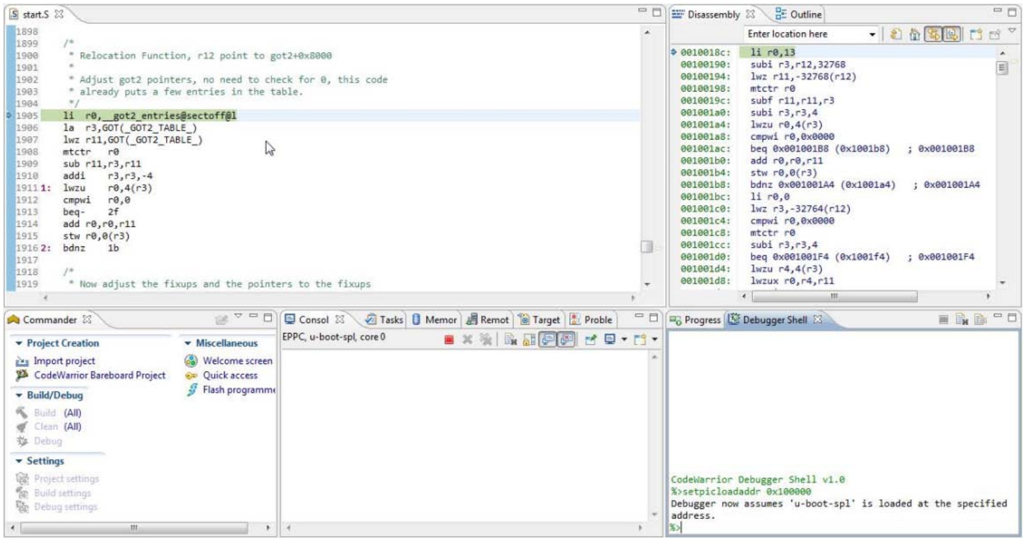
```
 */

    addi    r0,r10,in_ram - _start + _START_OFFSET

    /*
     * As IVPR is going to point RAM address,
     * Make sure IVOR15 has valid opcode to support debugger
     */
    mtspr   IVOR15,r0

    /*
     * Re-point the IVPR at RAM
     */
    mtspr   IVPR,r10

    mtlr    r0
    blr             /* NEVER RETURNS! */
    .globl  in_ram
in_ram:

    /*
     * Relocation Function, r12 point to got2+0x8000
```

3. Step Into: blr, it shows code in assembly. (See Stage 3 – Debug in RAM, for more information.)

6.2.4. Stage 3 – Debug in RAM

1. Set PIC load address as `0x100000` in Debugger Shell using `setpicloadaddr 0x100000` command.

Figure 36. Debugger shell view



```
    /*
     * Relocation Function, r12 point to got2+0x8000
     *
     * Adjust got2 pointers, no need to check for 0, this code
     * already puts a few entries in the table.
     */
    li  r0,__got2_entries@sectoff@l
    la  r3,GOT(_GOT2_TABLE_)
    lwz r11,GOT(_GOT2_TABLE_)
    mtctr   r0
    sub r11,r3,r11
    addi    r3,r3,-4
1:  lwzu    r0,4(r3)
    cmpwi   r0,0
    beq-    2f
    add r0,r0,r11
    stw r0,0(r3)
2:  bdnz    1b

    /*
     * Now adjust the fixups and the pointers to the fixups
```

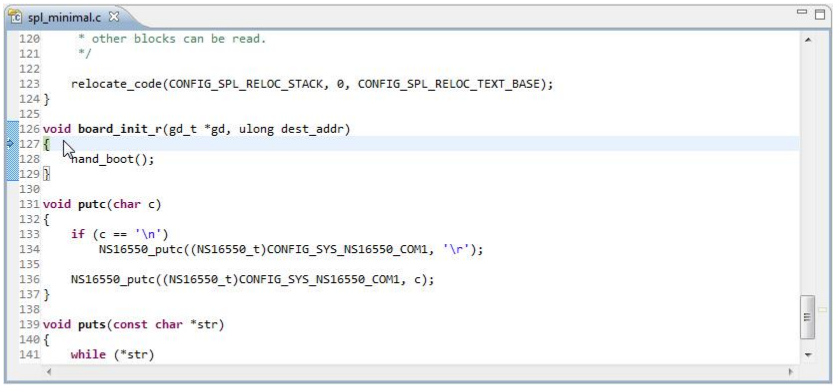
```
0010018c:   li r0,13
00100190:   subi r3,r12,32768
00100194:   lwz r11,-32768(r12)
00100198:   mtctr r0
0010019c:   subf r11,r11,r3
001001a0:   subi r3,r3,4
001001a4:   lwzu r0,4(r3)
001001a8:   cmpwi r0,0x0000
001001ac:   beq 0x001001B8 (0x1001b8)   ; 0x001001B8
001001b0:   add r0,r0,r11
001001b4:   stw r0,0(r3)
001001b8:   bdnz 0x001001A4 (0x1001a4)   ; 0x001001A4
001001bc:   li r0,0
001001c0:   lwz r3,-32764(r12)
001001c4:   cmpwi r0,0x0000
001001c8:   mtctr r0
001001cc:   subi r3,r3,4
001001d0:   beq 0x001001F4 (0x1001f4)   ; 0x001001F4
001001d4:   lwzu r4,4(r3)
001001d8:   lwzux r0,r4,r11
```

```
CodeWarrior Debugger Shell v1.0
%>setpicloadaddr 0x100000
Debugger now assumes 'u-boot-spl' is loaded at the specified
address.
%>
```

2. Debug until U-Boot RAMBOOT code is copied from NAND to RAM and control is transferred to it.

   a) Run to Line: `board_init_r` and do Step Into.

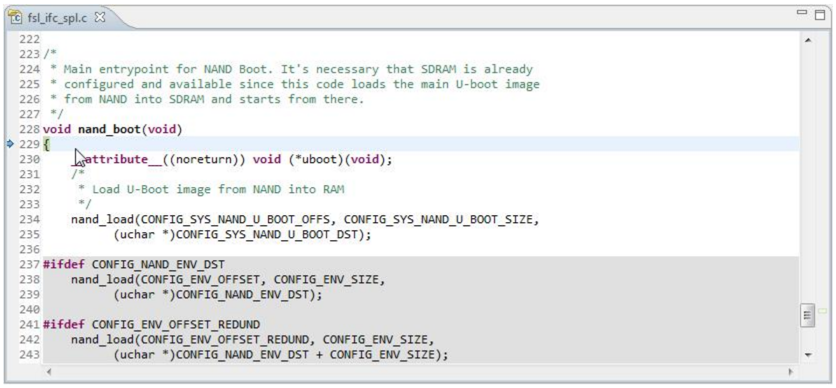
Figure 37. File editor



```
     * other blocks can be read.
     */

    relocate_code(CONFIG_SPL_RELOC_STACK, 0, CONFIG_SPL_RELOC_TEXT_BASE);
}

void board_init_r(gd_t *gd, ulong dest_addr)
{
    nand_boot();
}

void putc(char c)
{
    if (c == '\n')
        NS16550_putc((NS16550_t)CONFIG_SYS_NS16550_COM1, '\r');

    NS16550_putc((NS16550_t)CONFIG_SYS_NS16550_COM1, c);
}

void puts(const char *str)
{
    while (*str)
```

   b) Step Into: `nand_boot()` function.

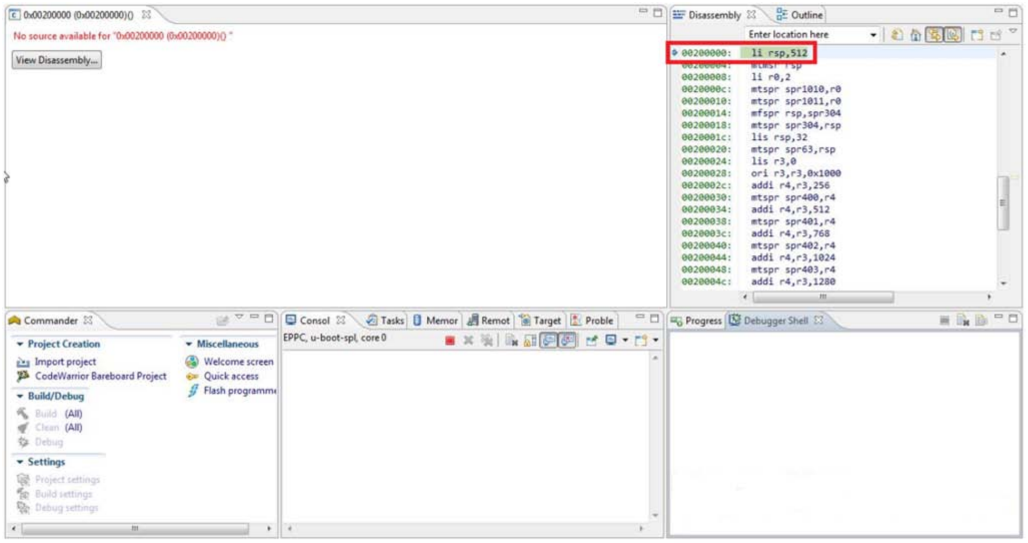
Figure 38.



```
/*
 * Main entrypoint for NAND Boot. It's necessary that SDRAM is already
 * configured and available since this code loads the main U-boot image
 * from NAND into SDRAM and starts from there.
 */
void nand_boot(void)
{
    __attribute__((noreturn)) void (*uboot)(void);
    /*
     * Load U-Boot image from NAND into RAM
     */
    nand_load(CONFIG_SYS_NAND_U_BOOT_OFFS, CONFIG_SYS_NAND_U_BOOT_SIZE,
          (uchar *)CONFIG_SYS_NAND_U_BOOT_DST);

#ifdef CONFIG_NAND_ENV_DST
    nand_load(CONFIG_ENV_OFFSET, CONFIG_ENV_SIZE,
          (uchar *)CONFIG_NAND_ENV_DST);

#ifdef CONFIG_ENV_OFFSET_REDUND
    nand_load(CONFIG_ENV_OFFSET_REDUND, CONFIG_ENV_SIZE,
          (uchar *)CONFIG_NAND_ENV_DST + CONFIG_ENV_SIZE);
```

c) This is the last function before control is transferred to u-boot RAMBOOT. Run to Line: `uboot()` and do Step Into. As soon as we Step Into `uboot()` function, control is transferred to U-Boot NAND RAMBOOT, that is, `0x00200000`. This address is used to set PIC load address for U-Boot NAND RAMBOOT debug.

Figure 39.



No source available for "0x00200000 (0x00200000)() "

```
00200000:   li rsp,512
00200004:   mtmsr rsp
00200008:   li r0,2
0020000c:   mtspr spr1010,r0
00200010:   mtspr spr1011,r0
00200014:   mfspr rsp,spr304
00200018:   mtspr spr304,rsp
0020001c:   lis rsp,32
00200020:   mtspr spr63,rsp
00200024:   lis r3,0
00200028:   ori r3,r3,0x1000
0020002c:   addi r4,r3,256
00200030:   mtspr spr400,r4
00200034:   addi r4,r3,512
00200038:   mtspr spr401,r4
0020003c:   addi r4,r3,768
00200040:   mtspr spr402,r4
00200044:   addi r4,r3,1024
00200048:   mtspr spr403,r4
0020004c:   addi r4,r3,1280
```

d) Further debugging is not possible with this u-boot-spl.elf and a new project needs to be created for U-Boot NAND debugging. (See U-Boot NAND RAMBOOT debugging, for more information.)

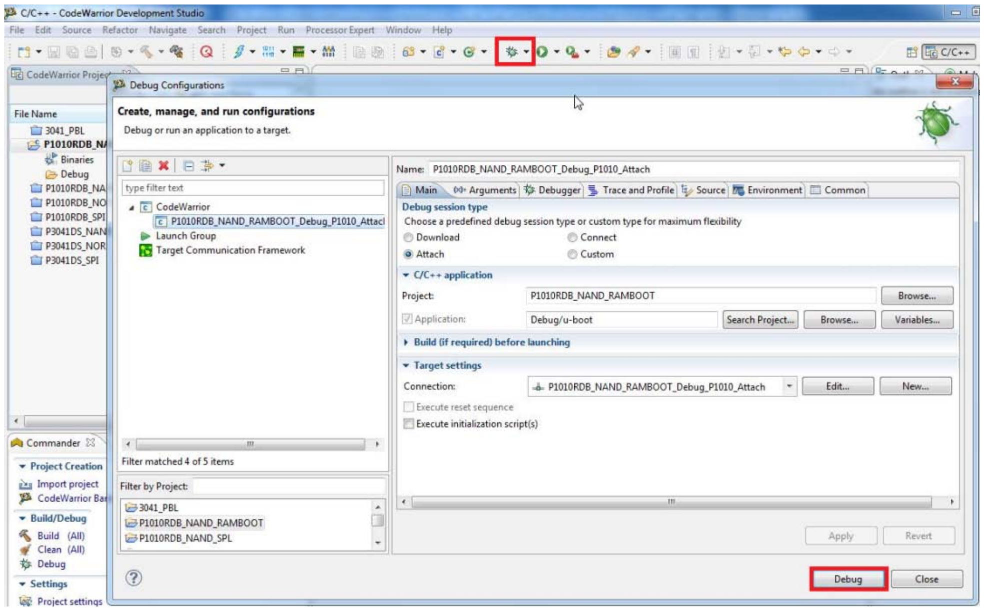
6.3. U-Boot NAND RAMBOOT debugging

For this stage, the U-Boot elf file generated during U-Boot compilation should be imported as a CodeWarrior project. (See Configuring a CodeWarrior project, for more information.)

6.3.1. Stage 0 – Connect CodeWarrior to board

1. Restart the board. U-Boot starts and relocates itself into RAM.
2. Before Debugging, run the board in Debug mode.

Figure 40. Debug configurations dialog



Debug Configurations

Create, manage, and run configurations

Debug or run an application to a target.

Name: P1010RDB_NAND_RAMBOOT_Debug_P1010_Attach

Debug session type

Choose a predefined debug session type or custom type for maximum flexibility

( ) Download ( ) Connect

(●) Attach ( ) Custom

C/C++ application

Project: P1010RDB_NAND_RAMBOOT

Application: Debug/u-boot

Target settings

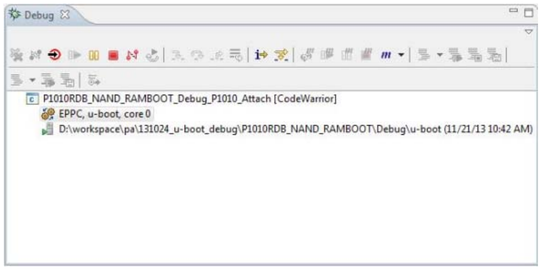
Connection: P1010RDB_NAND_RAMBOOT_Debug_P1010_Attach

[ ] Execute reset sequence

[ ] Execute initialization script(s)

3. This initializes and configures the TAP, then attaches to the board.

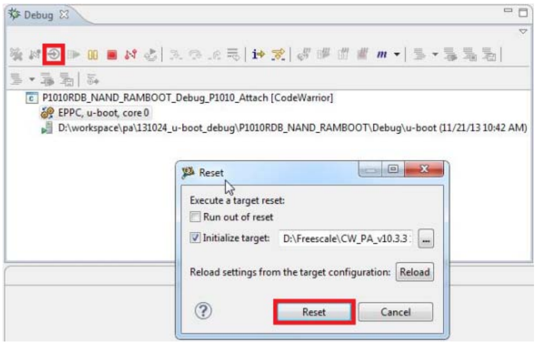
Figure 41. Debug view



```
P1010RDB_NAND_RAMBOOT_Debug_P1010_Attach [CodeWarrior]
  EPPC, u-boot, core 0
    D:\workspace\pa\131024_u-boot_debug\P1010RDB_NAND_RAMBOOT\Debug\u-boot (11/21/13 10:42 AM)
```

4. Reinitialize the target from CodeWarrior, using the U-Boot initialization file.

Figure 42. Reset dialog



P1010RDB_NAND_RAMBOOT_Debug_P1010_Attach [CodeWarrior]
EPPC, u-boot, core 0
D:\workspace\pa\131024_u-boot_debug\P1010RDB_NAND_RAMBOOT\Debug\u-boot (11/21/13 10:42 AM)

Execute a target reset:
[ ] Run out of reset
[✓] Initialize target: D:\Freescale\CW_PA_v10.3.3
Reload settings from the target configuration: Reload
Reset Cancel

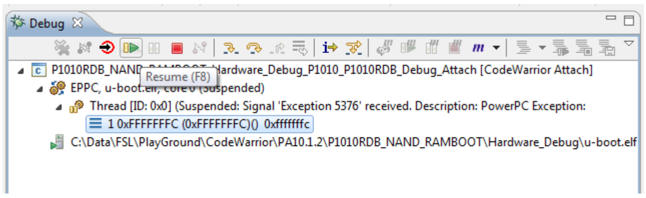
6.3.2. Stage 1 – Debug NAND RAMBOOT until U-Boot is relocated to DDR's higher address

1. Set PIC load address as `0x00200000` in Debugger Shell, using `setpicloadaddr 0x00200000` command.
2. Set break point at address `0x00200008` in Debugger Shell using `bp -hw 0x00200008` command.

NOTE The break point's address is needed to be offset with 8 bytes because it is required to jump over the instructions that enables the MSR[DE] bit, otherwise the break point will not hit.

3. Resume core operation.

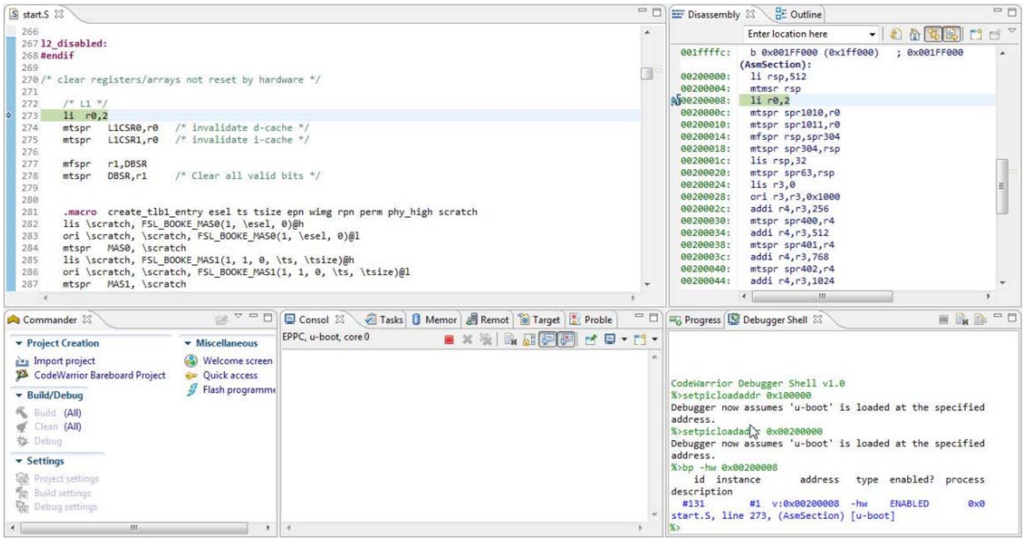
Figure 43. Debug view



P1010RDB_NAND_RAMBOOT_Hardware_Debug_P1010_P1010RDB_Debug_Attach [CodeWarrior Attach]
EPPC, u-boot.elf, core 0 (Suspended)
Thread [ID: 0x0] (Suspended: Signal 'Exception 5376' received. Description: PowerPC Exception:
1 0xFFFFFFFC (0xFFFFFFFC)() 0xfffffffc
C:\Data\FSL\PlayGround\CodeWarrior\PA10.1.2\P1010RDB_NAND_RAMBOOT\Hardware_Debug\u-boot.elf

4. When break point is hit, source code location is asked by CodeWarrior. After the path is specified, it shows the source code in CodeWarrior.

Figure 44. File editor



```
l2_disabled:
#endif

/* clear registers/arrays not reset by hardware */

    /* L1 */
    li  r0,2
    mtspr   L1CSR0,r0   /* invalidate d-cache */
    mtspr   L1CSR1,r0   /* invalidate i-cache */

    mfspr   r1,DBSR
    mtspr   DBSR,r1     /* Clear all valid bits */


    .macro  create_tlb1_entry esel ts tsize epn wimg rpn perm phy_high scratch
    lis \scratch, FSL_BOOKE_MAS0(1, \esel, 0)@h
    ori \scratch, \scratch, FSL_BOOKE_MAS0(1, \esel, 0)@l
    mtspr   MAS0, \scratch
    lis \scratch, FSL_BOOKE_MAS1(1, 1, 0, \ts, \tsize)@h
    ori \scratch, \scratch, FSL_BOOKE_MAS1(1, 1, 0, \ts, \tsize)@l
    mtspr   MAS1, \scratch
```

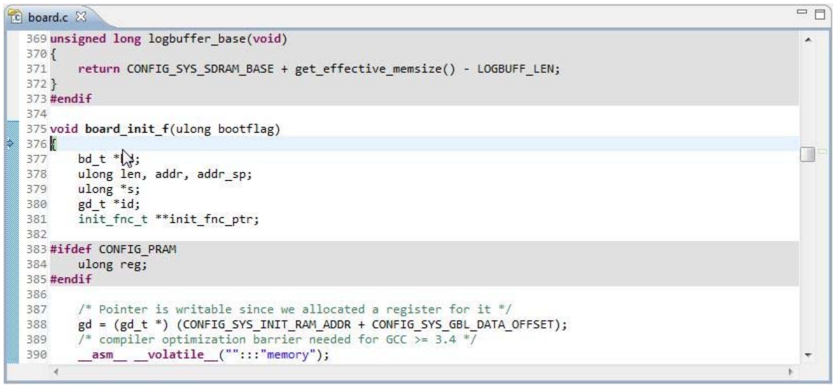
```
001ffffc:   b 0x001FF000 (0x1ff000)   ; 0x001FF000
          (AsmSection):
00200000:   li rsp,512
00200004:   mtmsr rsp
00200008:   li r0,2
0020000c:   mtspr spr1010,r0
00200010:   mtspr spr1011,r0
00200014:   mfspr rsp,spr304
00200018:   mtspr spr304,rsp
0020001c:   lis rsp,32
00200020:   mtspr spr63,rsp
00200024:   lis r3,0
00200028:   ori r3,r3,0x1000
0020002c:   addi r4,r3,256
00200030:   mtspr spr400,r4
00200034:   addi r4,r3,512
00200038:   mtspr spr401,r4
0020003c:   addi r4,r3,768
00200040:   mtspr spr402,r4
00200044:   addi r4,r3,1024
```

```
CodeWarrior Debugger Shell v1.0
%>setpicloadaddr 0x100000
Debugger now assumes 'u-boot' is loaded at the specified
address.
%>setpicloadaddr 0x00200000
Debugger now assumes 'u-boot' is loaded at the specified
address.
%>bp -hw 0x00200008
    id  instance       address   type  enabled?  process
description
  #131        #1  v:0x00200008  -hw    ENABLED       0x0
start.S, line 273, (AsmSection) [u-boot]
%>
```

5. Now debugging (step, run, or breakpoint) can be done until U-Boot code is relocated to the higher address of DDR.

   a) Run to Line: `board_init_f` and do Step Into.

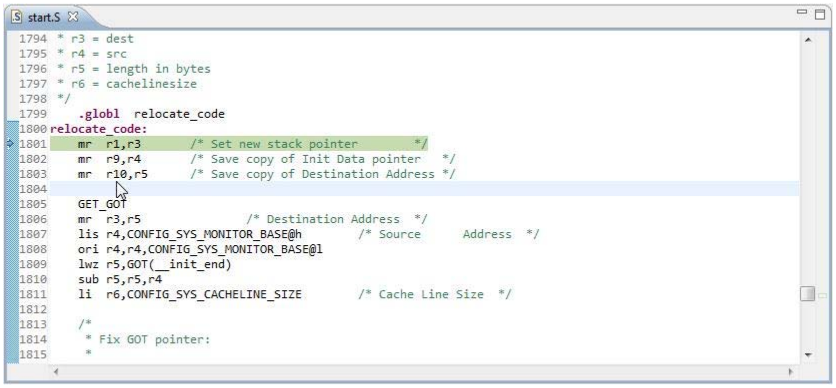
Figure 45. File editor



```
unsigned long logbuffer_base(void)
{
    return CONFIG_SYS_SDRAM_BASE + get_effective_memsize() - LOGBUFF_LEN;
}
#endif

void board_init_f(ulong bootflag)
{
    bd_t *bd;
    ulong len, addr, addr_sp;
    ulong *s;
    gd_t *id;
    init_fnc_t **init_fnc_ptr;

#ifdef CONFIG_PRAM
    ulong reg;
#endif

    /* Pointer is writable since we allocated a register for it */
    gd = (gd_t *) (CONFIG_SYS_INIT_RAM_ADDR + CONFIG_SYS_GBL_DATA_OFFSET);
    /* compiler optimization barrier needed for GCC >= 3.4 */
    __asm__ __volatile__("":::"memory");
```

   b) Run to Line: `relocate_code` and do Step Into.

Figure 46. File editor

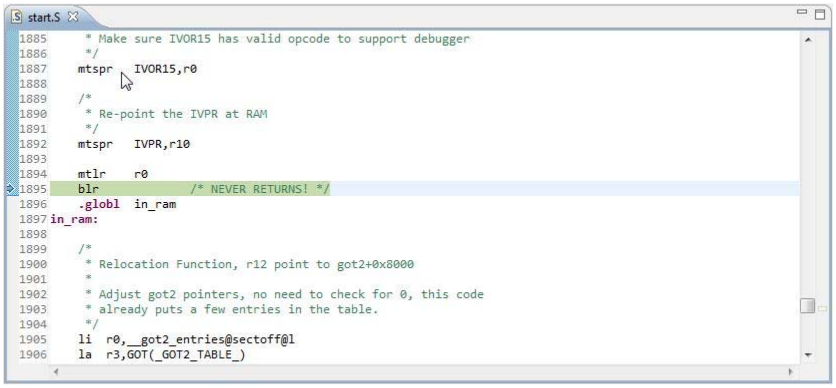


start.S

```
 * r3 = dest
 * r4 = src
 * r5 = length in bytes
 * r6 = cachelinesize
 */
    .globl  relocate_code
relocate_code:
    mr   r1,r3          /* Set new stack pointer         */
    mr   r9,r4          /* Save copy of Init Data pointer   */
    mr   r10,r5         /* Save copy of Destination Address */

    GET_GOT
    mr   r3,r5                  /* Destination Address  */
    lis r4,CONFIG_SYS_MONITOR_BASE@h        /* Source      Address  */
    ori r4,r4,CONFIG_SYS_MONITOR_BASE@l
    lwz r5,GOT(__init_end)
    sub r5,r5,r4
    li  r6,CONFIG_SYS_CACHELINE_SIZE        /* Cache Line Size  */

    /*
     * Fix GOT pointer:
     *
```

6. In file *start.S*, last instruction before moving to the higher address of DDR is `relocate_code`.

Figure 47. File editor



start.S

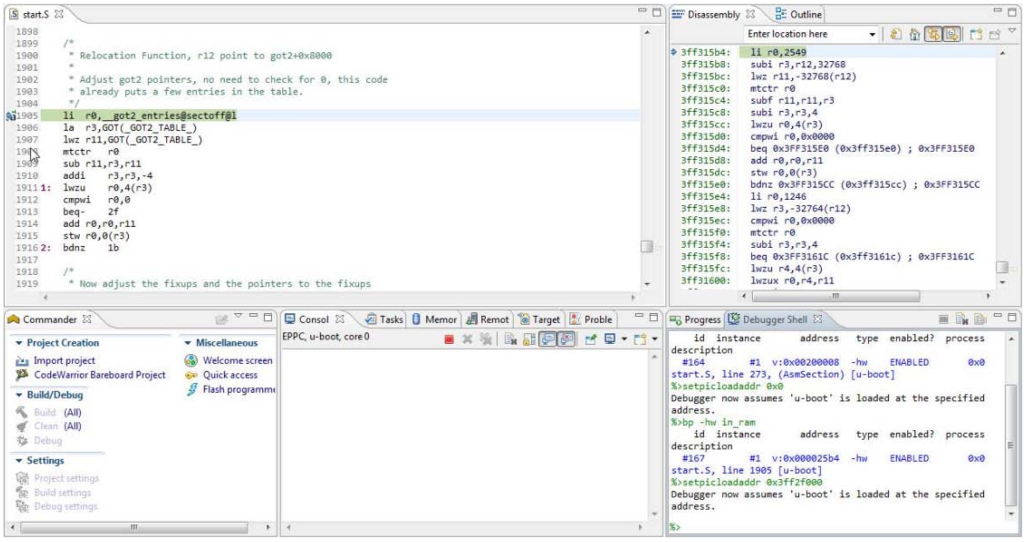
```
     * Make sure IVOR15 has valid opcode to support debugger
     */
    mtspr   IVOR15,r0

    /*
     * Re-point the IVPR at RAM
     */
    mtspr   IVPR,r10

    mtlr    r0
    blr                 /* NEVER RETURNS! */
    .globl  in_ram
in_ram:

    /*
     * Relocation Function, r12 point to got2+0x8000
     *
     * Adjust got2 pointers, no need to check for 0, this code
     * already puts a few entries in the table.
     */
    li  r0,__got2_entries@sectoff@l
    la  r3,GOT(_GOT2_TABLE_)
```

7. Now Step Into `blr`, CodeWarrior will show the code in assembly. (See Stage 2 – Debug in DDR's higher address, for more information.)

6.3.3. Stage 2 – Debug in DDR's higher address

1. Set PIC load address as `0x3ff2f000` using Debugger Shell command `setpicloadaddr 0x3ff2f000`. (See How to calculate PIC load address, for more information.)

Figure 48. Debugger shell view



```
1898
1899     /*
1900      * Relocation Function, r12 point to got2+0x8000
1901      *
1902      * Adjust got2 pointers, no need to check for 0, this code
1903      * already puts a few entries in the table.
1904      */
1905     li  r0,__got2_entries@sectoff@l
1906     la  r3,GOT(_GOT2_TABLE_)
1907     lwz r11,GOT(_GOT2_TABLE_)
1908     mtctr   r0
1909     sub r11,r3,r11
1910     addi    r3,r3,-4
1911 1:  lwzu    r0,4(r3)
1912     cmpwi   r0,0
1913     beq-    2f
1914     add r0,r0,r11
1915     stw r0,0(r3)
1916 2:  bdnz    1b
1917
1918     /*
1919      * Now adjust the fixups and the pointers to the fixups
```

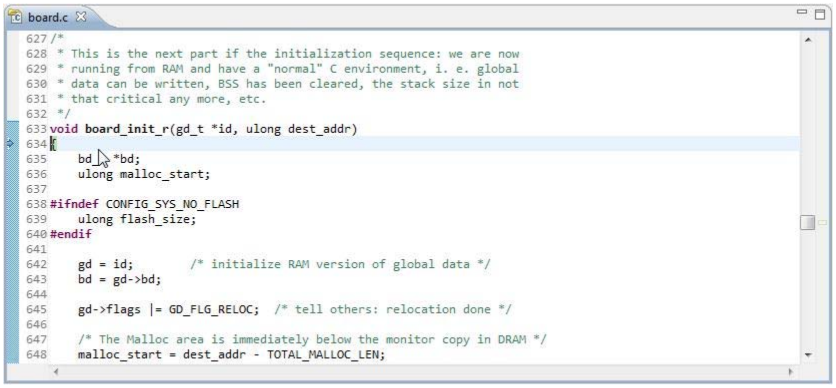
```
3ff315b4:   li r0,2549
3ff315b8:   subi r3,r12,32768
3ff315bc:   lwz r11,-32768(r12)
3ff315c0:   mtctr r0
3ff315c4:   subf r11,r11,r3
3ff315c8:   subi r3,r3,4
3ff315cc:   lwzu r0,4(r3)
3ff315d0:   cmpwi r0,0x0000
3ff315d4:   beq 0x3FF315E0 (0x3ff315e0) ; 0x3FF315E0
3ff315d8:   add r0,r0,r11
3ff315dc:   stw r0,0(r3)
3ff315e0:   bdnz 0x3FF315CC (0x3ff315cc) ; 0x3FF315CC
3ff315e4:   li r0,1246
3ff315e8:   lwz r3,-32764(r12)
3ff315ec:   cmpwi r0,0x0000
3ff315f0:   mtctr r0
3ff315f4:   subi r3,r3,4
3ff315f8:   beq 0x3FF3161C (0x3ff3161c) ; 0x3FF3161C
3ff315fc:   lwzu r4,4(r3)
3ff31600:   lwzux r0,r4,r11
```

```
   id  instance      address   type  enabled?  process
description
  #164        #1  v:0x00200008  -hw    ENABLED       0x0
start.S, line 273, (AsmSection) [u-boot]
%>setpicloadaddr 0x0
Debugger now assumes 'u-boot' is loaded at the specified
address.
%>bp -hw in_ram
   id  instance      address   type  enabled?  process
description
  #167        #1  v:0x000025b4  -hw    ENABLED       0x0
start.S, line 1905 [u-boot]
%>setpicloadaddr 0x3ff2f000
Debugger now assumes 'u-boot' is loaded at the specified
address.

%>
```

2. You can debug until U-Boot is running.

a) Run to Line: `board_init_r` and Step Into.

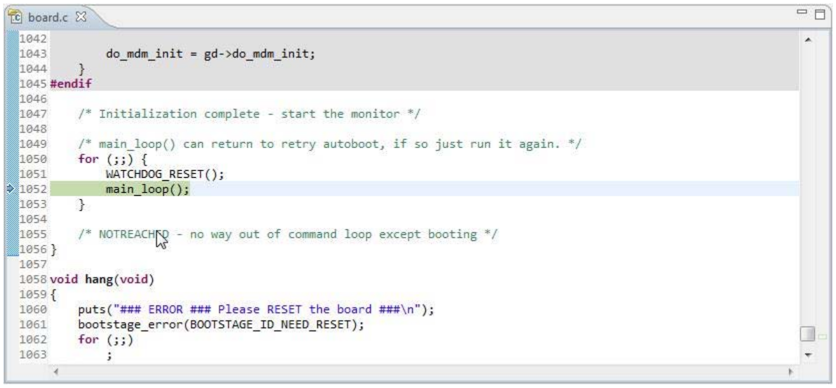
Figure 49. File editor



```
/*
 * This is the next part if the initialization sequence: we are now
 * running from RAM and have a "normal" C environment, i. e. global
 * data can be written, BSS has been cleared, the stack size in not
 * that critical any more, etc.
 */
void board_init_r(gd_t *id, ulong dest_addr)
{
    bd_t *bd;
    ulong malloc_start;

#ifndef CONFIG_SYS_NO_FLASH
    ulong flash_size;
#endif

    gd = id;        /* initialize RAM version of global data */
    bd = gd->bd;

    gd->flags |= GD_FLG_RELOC;  /* tell others: relocation done */

    /* The Malloc area is immediately below the monitor copy in DRAM */
    malloc_start = dest_addr - TOTAL_MALLOC_LEN;
```

b) Run to Line: `main_loop()`

Figure 50. File editor



```
        do_mdm_init = gd->do_mdm_init;
    }
#endif

    /* Initialization complete - start the monitor */

    /* main_loop() can return to retry autoboot, if so just run it again. */
    for (;;) {
        WATCHDOG_RESET();
        main_loop();
    }

    /* NOTREACHED - no way out of command loop except booting */
}

void hang(void)
{
    puts("### ERROR ### Please RESET the board ###\n");
    bootstage_error(BOOTSTAGE_ID_NEED_RESET);
    for (;;)
        ;
```

7. Debugging U-Boot from SPI/SD/MMC for e500v2

Booting from SPI and SD\MMC are similar, the only difference between these is how the final image is build. This chapter provides necessary steps for SPI U-Boot debugging.

7.1. Debugging environment

Given below is the setup used for U-Boot debugging on e500v2 core:

1. P1010RDB board.
2. Compiled U-Boot for the SPI FLASH target.
3. Flash U-Boot on the target board. (See SDK documentation, for more information on how to program the U-Boot to SPI flash.)
4. Switches set for SPI boot. (See SDK documentation for more information on how to set switches.)
5. Latest release of CodeWarrior IDE.
6. P1010RDB_uboot_32.tcl initialization file.
7. USB TAP or other probe.

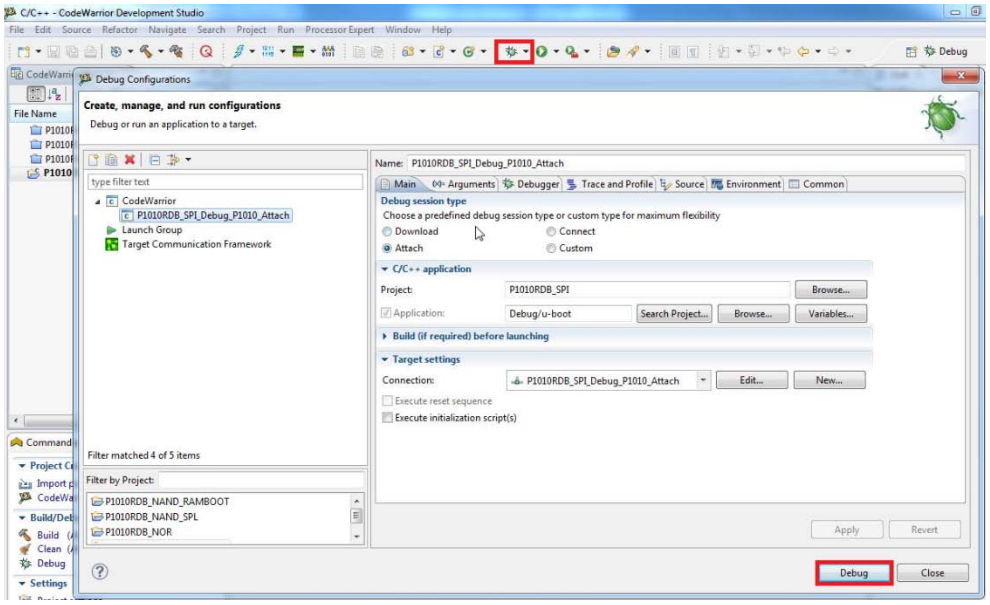
7.2. U-Boot SPI debugging

Import the U-Boot elf file, generated during U-Boot compilation as a CodeWarrior project. (See Configuring a CodeWarrior project, for more information.)

7.2.1. Stage 0 – Connect CodeWarrior to board

1. Before debugging, run the board in debug mode.

Figure 51. Debug configurations dialog



Create, manage, and run configurations

Debug or run an application to a target.

- CodeWarrior
  - P1010RDB_SPI_Debug_P1010_Attach
- Launch Group
- Target Communication Framework

Filter matched 4 of 5 items

Name: P1010RDB_SPI_Debug_P1010_Attach

Debug session type

Choose a predefined debug session type or custom type for maximum flexibility

Download ( ) Connect ( )
Attach (●) Custom ( )

C/C++ application

Project: P1010RDB_SPI

Application: Debug/u-boot

Build (if required) before launching

Target settings

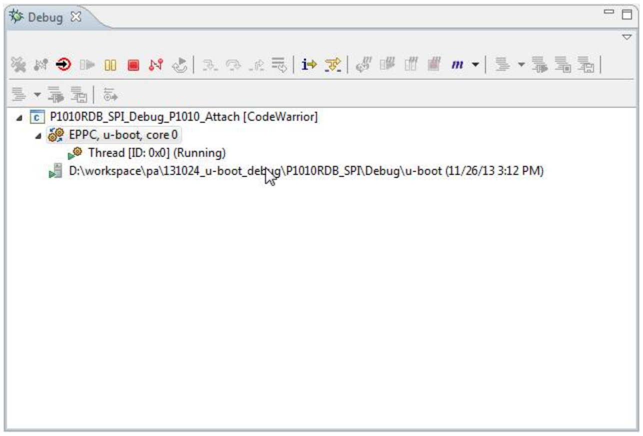
Connection: P1010RDB_SPI_Debug_P1010_Attach

☐ Execute reset sequence

☐ Execute initialization script(s)

2. Start the debugging session using the setup provided in Figure 51. This initializes and configures the TAP, then attaches to the board.

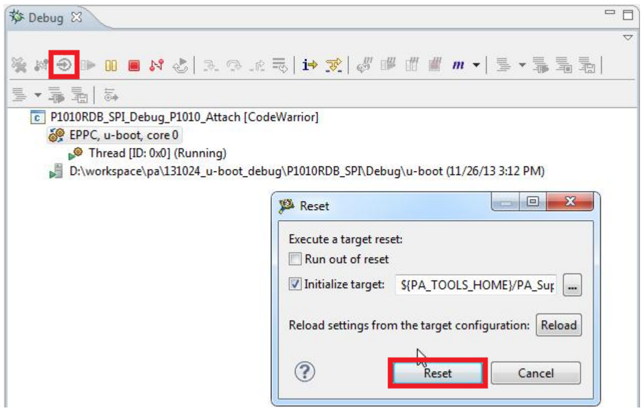
Figure 52. Debug view



```
P1010RDB_SPI_Debug_P1010_Attach [CodeWarrior]
  EPPC, u-boot, core 0
    Thread [ID: 0x0] (Running)
  D:\workspace\pa\131024_u-boot_debug\P1010RDB_SPI\Debug\u-boot (11/26/13 3:12 PM)
```

3. Reinitialize the target from CodeWarrior.

Figure 53. Reset dialog



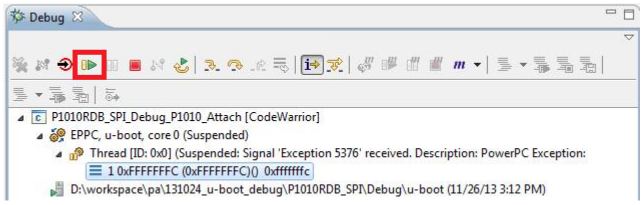
P1010RDB_SPI_Debug_P1010_Attach [CodeWarrior]
EPPC, u-boot, core 0
Thread [ID: 0x0] (Running)
D:\workspace\pa\131024_u-boot_debug\P1010RDB_SPI\Debug\u-boot (11/26/13 3:12 PM)

Execute a target reset:
Run out of reset
[✓] Initialize target: ${PA_TOOLS_HOME}/PA_Sup
Reload settings from the target configuration:

7.2.2. Stage 1 – Debug SPI until U-Boot relocated to DDR’s higher address

1. Set hardware break point at address 0x1107f008 in Debugger Shell using *bp* `–hw 0x1107f008` command.
2. Resume core operation.

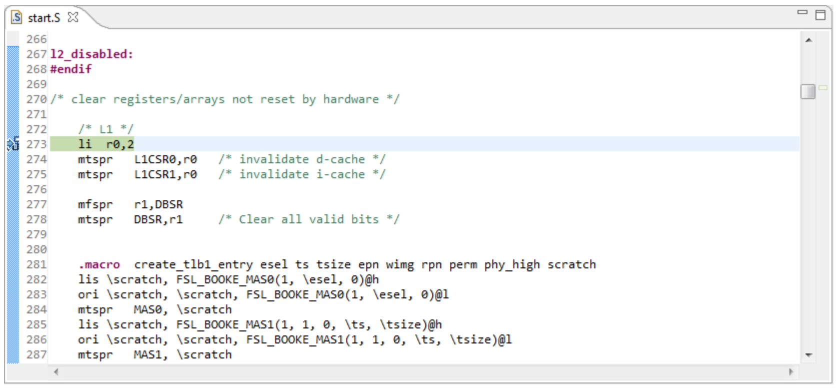
Figure 54. Debug view



P1010RDB_SPI_Debug_P1010_Attach [CodeWarrior]
EPPC, u-boot, core 0 (Suspended)
Thread [ID: 0x0] (Suspended: Signal 'Exception 5376' received. Description: PowerPC Exception:
1 0xFFFFFFFC (0xFFFFFFFC)() 0xfffffffc
D:\workspace\pa\131024_u-boot_debug\P1010RDB_SPI\Debug\u-boot (11/26/13 3:12 PM)

3. Once the break point is hit, specify the source code location. After the path is specified, it shows the source code in CodeWarrior.

Figure 55. File editor

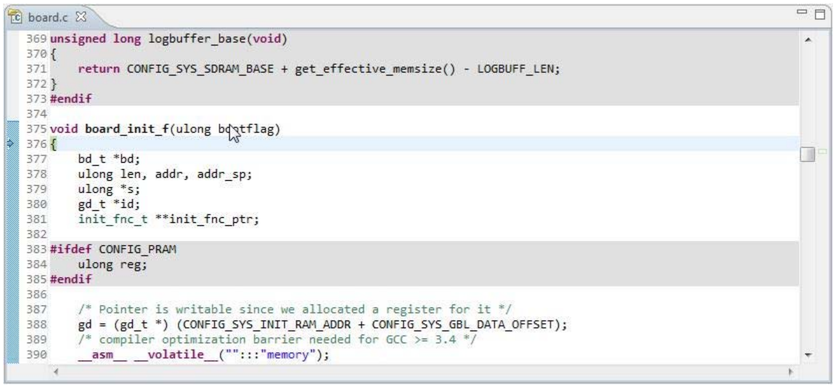


```

l2_disabled:
#endif

/* clear registers/arrays not reset by hardware */

    /* L1 */
    li  r0,2
    mtspr   L1CSR0,r0   /* invalidate d-cache */
    mtspr   L1CSR1,r0   /* invalidate i-cache */

    mfspr   r1,DBSR
    mtspr   DBSR,r1     /* Clear all valid bits */


    .macro  create_tlb1_entry esel ts tsize epn wimg rpn perm phy_high scratch
    lis \scratch, FSL_BOOKE_MAS0(1, \esel, 0)@h
    ori \scratch, \scratch, FSL_BOOKE_MAS0(1, \esel, 0)@l
    mtspr   MAS0, \scratch
    lis \scratch, FSL_BOOKE_MAS1(1, 1, 0, \ts, \tsize)@h
    ori \scratch, \scratch, FSL_BOOKE_MAS1(1, 1, 0, \ts, \tsize)@l
    mtspr   MAS1, \scratch
```

4. Now debugging (step, run, or breakpoint) can be done until U-Boot code is relocated to the higher address of DDR.

a) Run to Line: `board_init_f` and do Step Into.

Figure 56. File editor

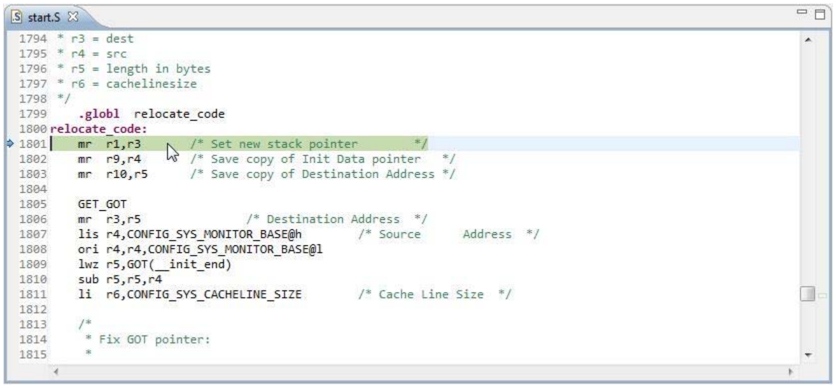


board.c

```
unsigned long logbuffer_base(void)
{
    return CONFIG_SYS_SDRAM_BASE + get_effective_memsize() - LOGBUFF_LEN;
}
#endif

void board_init_f(ulong bootflag)
{
    bd_t *bd;
    ulong len, addr, addr_sp;
    ulong *s;
    gd_t *id;
    init_fnc_t **init_fnc_ptr;

#ifdef CONFIG_PRAM
    ulong reg;
#endif

    /* Pointer is writable since we allocated a register for it */
    gd = (gd_t *) (CONFIG_SYS_INIT_RAM_ADDR + CONFIG_SYS_GBL_DATA_OFFSET);
    /* compiler optimization barrier needed for GCC >= 3.4 */
    __asm__ __volatile__("":::"memory");
```

b) Run to Line: `relocate_code` and do Step Into.

Figure 57. File editor

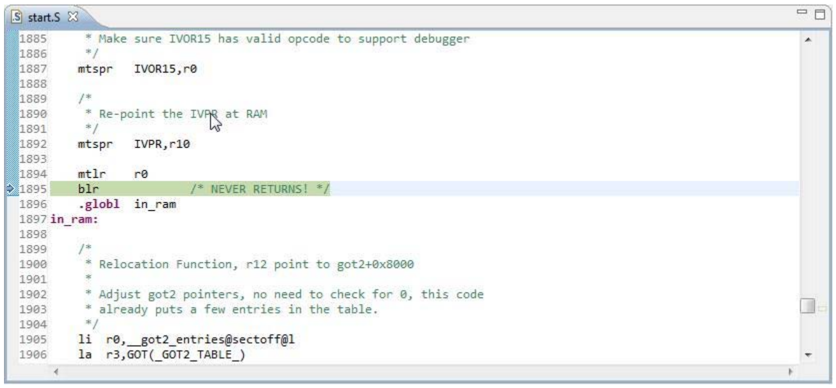


start.S

```
 * r3 = dest
 * r4 = src
 * r5 = length in bytes
 * r6 = cachelinesize
 */
    .globl  relocate_code
relocate_code:
    mr   r1,r3          /* Set new stack pointer         */
    mr   r9,r4          /* Save copy of Init Data pointer   */
    mr   r10,r5         /* Save copy of Destination Address */

    GET_GOT
    mr   r3,r5                  /* Destination Address  */
    lis r4,CONFIG_SYS_MONITOR_BASE@h        /* Source      Address  */
    ori r4,r4,CONFIG_SYS_MONITOR_BASE@l
    lwz r5,GOT(__init_end)
    sub r5,r5,r4
    li  r6,CONFIG_SYS_CACHELINE_SIZE        /* Cache Line Size  */

    /*
     * Fix GOT pointer:
     *
```

c) In file *start.S*, last instruction before moving to the higher address of DDR is `relocate_code`.

Figure 58. File editor



start.S

```
     * Make sure IVOR15 has valid opcode to support debugger
     */
    mtspr   IVOR15,r0

    /*
     * Re-point the IVPR at RAM
     */
    mtspr   IVPR,r10

    mtlr    r0
    blr                 /* NEVER RETURNS! */
    .globl  in_ram
in_ram:

    /*
     * Relocation Function, r12 point to got2+0x8000
     *
     * Adjust got2 pointers, no need to check for 0, this code
     * already puts a few entries in the table.
     */
    li  r0,__got2_entries@sectoff@l
    la  r3,GOT(_GOT2_TABLE_)
```

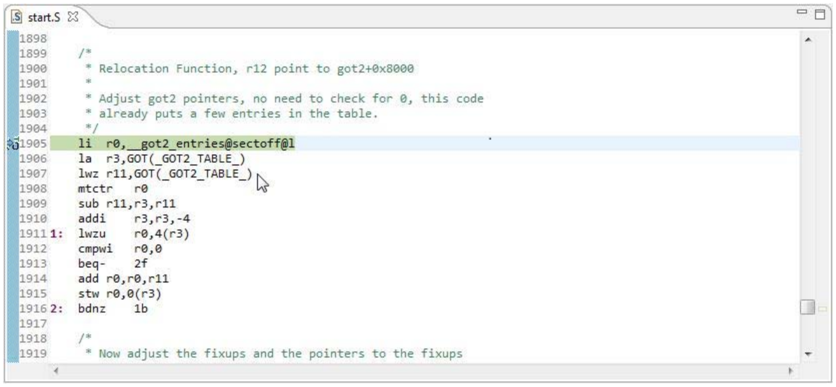
d) Step Into `blr`, it shows the code in assembly. (See Stage 2– Debug SPI in DDR's higher address, for more information.)

NOTE To find the correct address for hardware break point, that is, `0x1107008`, disassembly on u-boot.elf is done and the `_start_e500` address is searched for. Add 8 to this address to have the correct address for hardware break point.

7.2.3.Stage 2– Debug SPI in DDR's higher address

1. Set PIC load address as `0x3ff30000` using Debugger Shell command `setpicloadaddr 0x3ff30000`. (See How to calculate PIC load address, for more information.)

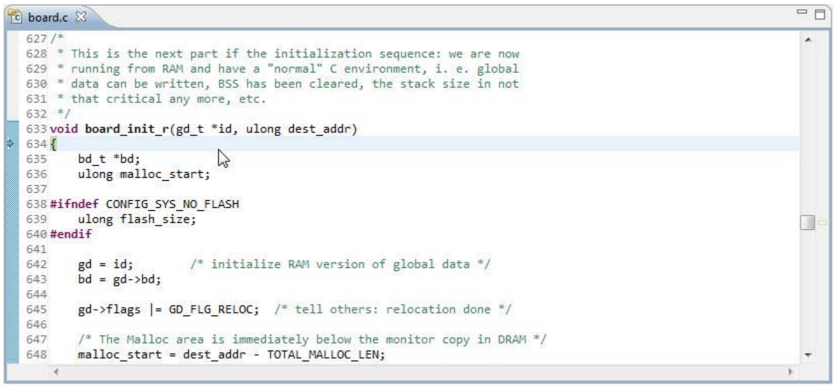
Figure 59. File editor



```
	/*
	 * Relocation Function, r12 point to got2+0x8000
	 *
	 * Adjust got2 pointers, no need to check for 0, this code
	 * already puts a few entries in the table.
	 */
	li	r0,__got2_entries@sectoff@l
	la	r3,GOT(_GOT2_TABLE_)
	lwz	r11,GOT(_GOT2_TABLE_)
	mtctr	r0
	sub	r11,r3,r11
	addi	r3,r3,-4
1:	lwzu	r0,4(r3)
	cmpwi	r0,0
	beq-	2f
	add	r0,r0,r11
	stw	r0,0(r3)
2:	bdnz	1b

	/*
	 * Now adjust the fixups and the pointers to the fixups
```

2. Run to Line: `board_init_r` and do Step Into.

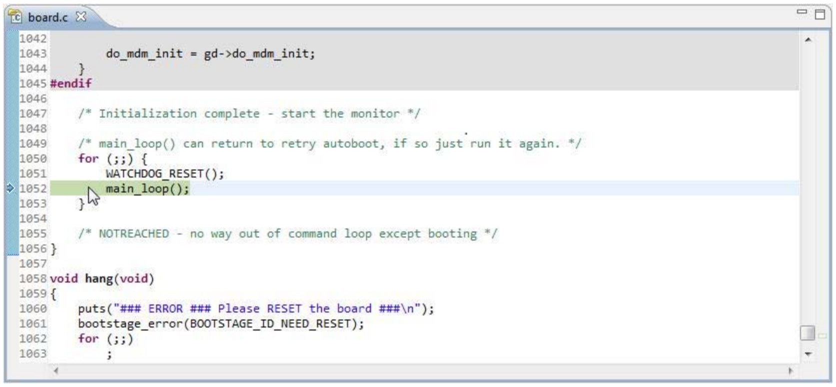
Figure 60. File editor



```
/*
 * This is the next part if the initialization sequence: we are now
 * running from RAM and have a "normal" C environment, i. e. global
 * data can be written, BSS has been cleared, the stack size in not
 * that critical any more, etc.
 */
void board_init_r(gd_t *id, ulong dest_addr)
{
	bd_t *bd;
	ulong malloc_start;

#ifndef CONFIG_SYS_NO_FLASH
	ulong flash_size;
#endif

	gd = id;		/* initialize RAM version of global data */
	bd = gd->bd;

	gd->flags |= GD_FLG_RELOC;	/* tell others: relocation done */

	/* The Malloc area is immediately below the monitor copy in DRAM */
	malloc_start = dest_addr - TOTAL_MALLOC_LEN;
```

3. Run to Line: `main_loop()`.

Figure 61. File editor



```
        do_mdm_init = gd->do_mdm_init;
    }
#endif

    /* Initialization complete - start the monitor */

    /* main_loop() can return to retry autoboot, if so just run it again. */
    for (;;) {
        WATCHDOG_RESET();
        main_loop();
    }

    /* NOTREACHED - no way out of command loop except booting */
}

void hang(void)
{
    puts("### ERROR ### Please RESET the board ###\n");
    bootstage_error(BOOTSTAGE_ID_NEED_RESET);
    for (;;)
        ;
```

8. Debugging U-Boot from NOR for e500mc

8.1. Debug environment

Setup used for U-Boot debugging on e500mc core:

1. P3041DS Hydra board.
2. Compiled U-Boot for the NOR FLASH target.
3. Flash U-Boot on the target board. (See SDK documentation, for more information on how to program the U-Boot to NOR flash.)
4. Switches set for NOR boot (See SDK documentation, for more information on how to set switches.)
5. Latest release of CodeWarrior IDE.
6. P3041DS_uboot_36.tcl initialization file.
7. USB TAP or other probe.

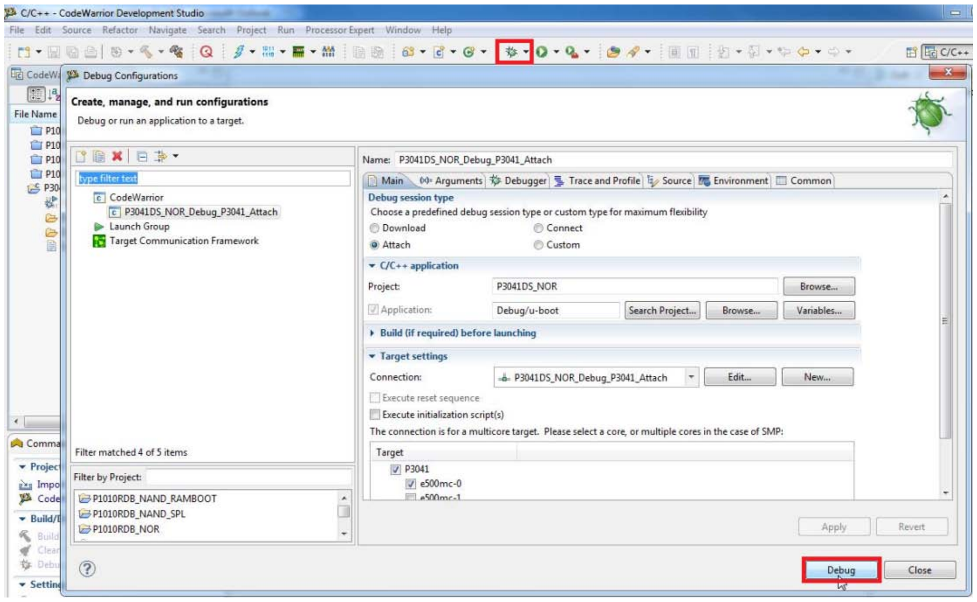
8.2. U-Boot NOR debugging

U-Boot elf file generated during U-Boot compilation should be imported as CodeWarrior project. (See Configuring a CodeWarrior project, for more information.)

8.2.1. Stage 0 – Connect CodeWarrior to board

1. Before debugging, run the board in debug mode.

Figure 62. Debug configurations dialog



Debug Configurations

Create, manage, and run configurations

Debug or run an application to a target.

type filter text

CodeWarrior
P3041DS_NOR_Debug_P3041_Attach
Launch Group
Target Communication Framework

Filter matched 4 of 5 items

Filter by Project:
P1010RDB_NAND_RAMBOOT
P1010RDB_NAND_SPL
P1010RDB_NOR

Name: P3041DS_NOR_Debug_P3041_Attach

Main | Arguments | Debugger | Trace and Profile | Source | Environment | Common

Debug session type
Choose a predefined debug session type or custom type for maximum flexibility
Download | Connect
Attach (selected) | Custom

C/C++ application
Project: P3041DS_NOR
Application: Debug/u-boot

Build (if required) before launching

Target settings
Connection: P3041DS_NOR_Debug_P3041_Attach
Execute reset sequence
Execute initialization script(s)
The connection is for a multicore target. Please select a core, or multiple cores in the case of SMP:

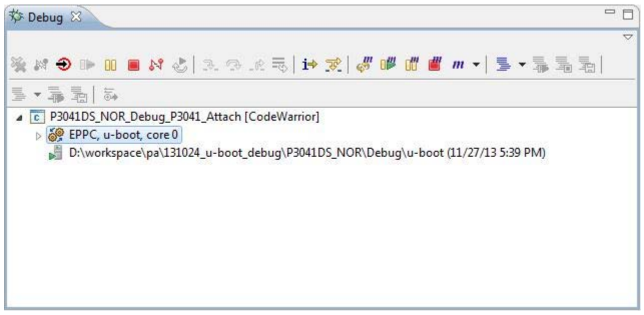
Target
[✓] P3041
[✓] e500mc-0
e500mc-1

Apply | Revert

Debug | Close

2. Start the debugging session using the setup provided in Figure 62. This initializes and configures the TAP, then attaches to the board.

Figure 63. Debug view

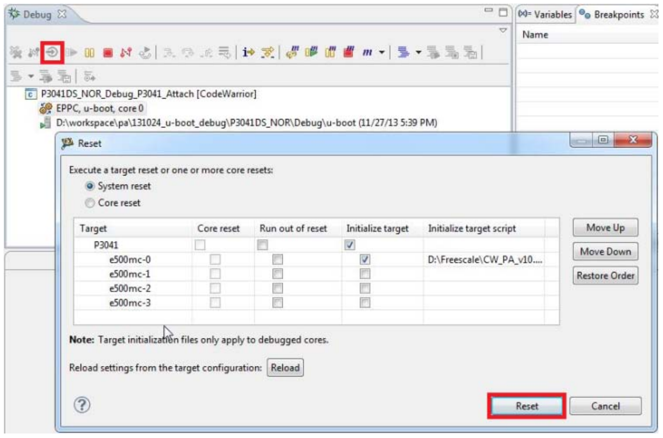


Debug

P3041DS_NOR_Debug_P3041_Attach [CodeWarrior]
EPPC, u-boot, core 0
D:\workspace\pa\131024_u-boot_debug\P3041DS_NOR\Debug\u-boot (11/27/13 5:39 PM)

3. Reinitialize the target from CodeWarrior.

Figure 64. Reset dialog



```
P3041DS_NOR_Debug_P3041_Attach [CodeWarrior]
  EPPC, u-boot, core 0
  D:\workspace\pa\131024_u-boot_debug\P3041DS_NOR\Debug\u-boot (11/27/13 5:39 PM)
```

Reset

Execute a target reset or one or more core resets:
[✓ System reset] Core reset

Target	Core reset	Run out of reset	Initialize target	Initialize target script
P3041	☐	☐	✓	
e500mc-0	☐	☐	✓	D:\Freescale\CW_PA_v10....
e500mc-1	☐	☐	☐	
e500mc-2	☐	☐	☐	
e500mc-3	☐	☐	☐	

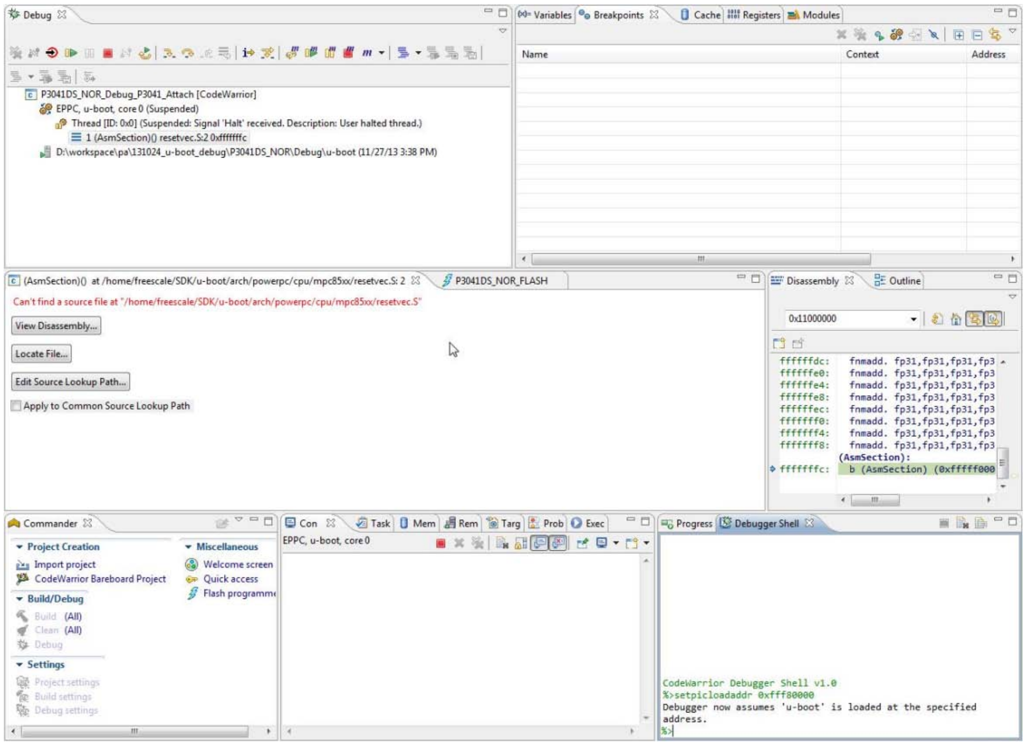
Note: Target initialization files only apply to debugged cores.

Reload settings from the target configuration: Reload

8.2.2. Stage 1 – Debug NOR for AS0

1. Set PIC load address as `0xfff80000` using Debugger Shell command `setpicloadaddr 0xfff80000`.

Figure 65. Debugger shell view



```
P3041DS_NOR_Debug_P3041_Attach [CodeWarrior]
  EPPC, u-boot, core 0 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      1 (AsmSection)() resetvec.S:2 0xfffffffc
  D:\workspace\pa\131024_u-boot_debug\P3041DS_NOR\Debug\u-boot (11/27/13 3:38 PM)
```

(AsmSection)() at /home/freescale/SDK/u-boot/arch/powerpc/cpu/mpc85xx/resetvec.S: 2

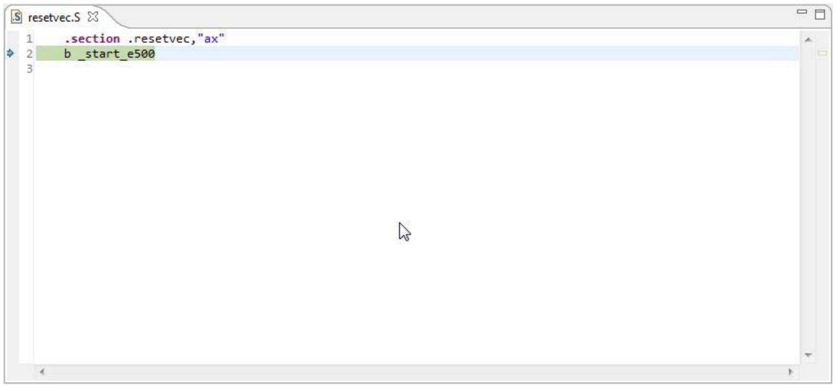
Can't find a source file at "/home/freescale/SDK/u-boot/arch/powerpc/cpu/mpc85xx/resetvec.S"

```
fffffffdc:   fnmadd. fp31,fp31,fp31,fp3
ffffffe0:    fnmadd. fp31,fp31,fp31,fp3
ffffffe4:    fnmadd. fp31,fp31,fp31,fp3
ffffffe8:    fnmadd. fp31,fp31,fp31,fp3
ffffffec:    fnmadd. fp31,fp31,fp31,fp3
fffffff0:    fnmadd. fp31,fp31,fp31,fp3
fffffff4:    fnmadd. fp31,fp31,fp31,fp3
fffffff8:    fnmadd. fp31,fp31,fp31,fp3
           (AsmSection):
fffffffc:    b (AsmSection) (0xfffff000
```

```
CodeWarrior Debugger Shell v1.0
%>setpicloadaddr 0xfff80000
Debugger now assumes 'u-boot' is loaded at the specified
address.
%>
```

2. After the path is specified, source code is available in CodeWarrior.

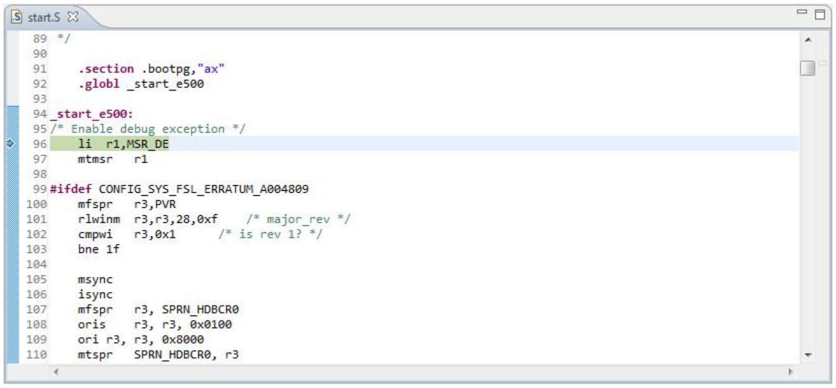
Figure 66. File editor



```
    .section .resetvec,"ax"
    b _start_e500
```

3. Now debugging (step, run, or breakpoint) can be done before switching to AS1.

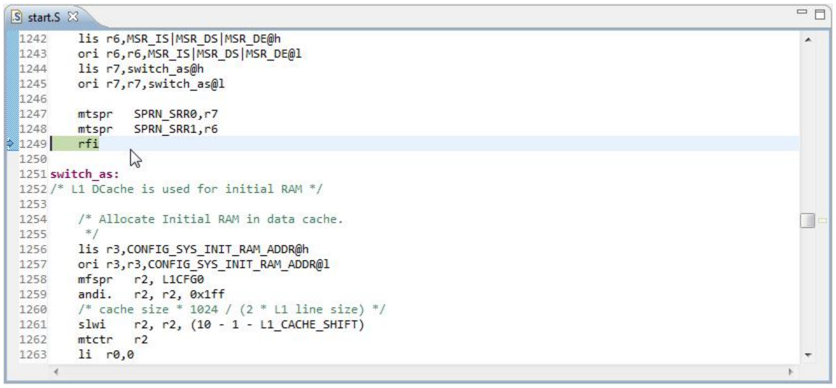
Figure 67. File editor



```
 */

    .section .bootpg,"ax"
    .globl _start_e500

_start_e500:
/* Enable debug exception */
    li  r1,MSR_DE
    mtmsr   r1

#ifdef CONFIG_SYS_FSL_ERRATUM_A004809
    mfspr   r3,PVR
    rlwinm  r3,r3,28,0xf    /* major_rev */
    cmpwi   r3,0x1      /* is rev 1? */
    bne 1f

    msync
    isync
    mfspr   r3, SPRN_HDBCR0
    oris    r3, r3, 0x0100
    ori r3, r3, 0x8000
    mtspr   SPRN_HDBCR0, r3
```

4. In start.S, last instruction before moving to AS1 is `rfi` before `switch_as`. (See Stage 2 – Debug NOR for AS1, for more information.)

Figure 68. File editor



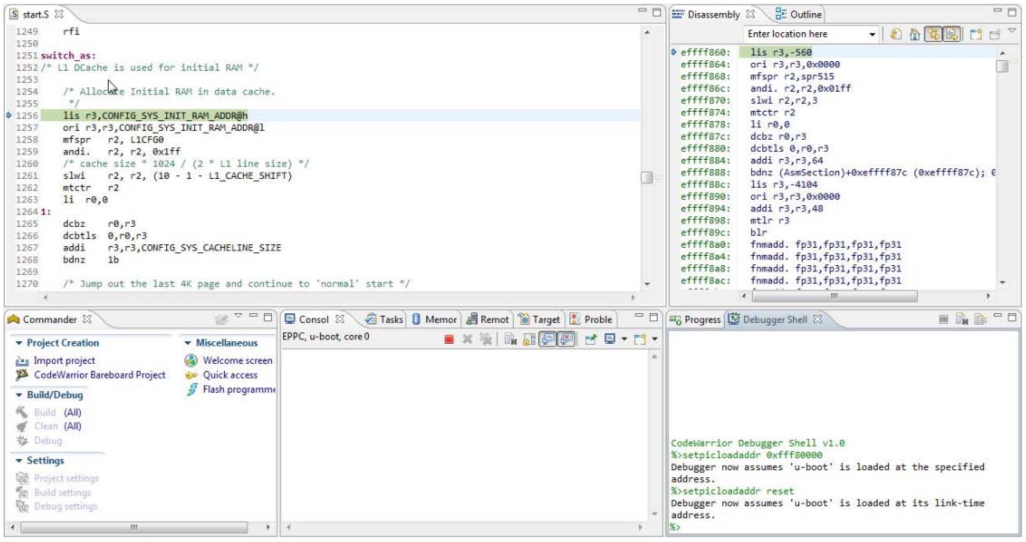
```
    lis r6,MSR_IS|MSR_DS|MSR_DE@h
    ori r6,r6,MSR_IS|MSR_DS|MSR_DE@l
    lis r7,switch_as@h
    ori r7,r7,switch_as@l

    mtspr   SPRN_SRR0,r7
    mtspr   SPRN_SRR1,r6
    rfi

switch_as:
/* L1 DCache is used for initial RAM */

    /* Allocate Initial RAM in data cache.
     */
    lis r3,CONFIG_SYS_INIT_RAM_ADDR@h
    ori r3,r3,CONFIG_SYS_INIT_RAM_ADDR@l
    mfspr   r2, L1CFG0
    andi.   r2, r2, 0x1ff
    /* cache size * 1024 / (2 * L1 line size) */
    slwi    r2, r2, (10 - 1 - L1_CACHE_SHIFT)
    mtctr   r2
    li  r0,0
```

8.2.3. Stage 2 – Debug NOR for AS1

1. Step Into this instruction.

2. Reset PIC load address using Debugger Shell command `setpicloadaddr reset`.

Figure 69. Debugger shell view



```
    rfi

switch_as:
/* L1 DCache is used for initial RAM */

    /* Allocate Initial RAM in data cache.
     */
    lis r3,CONFIG_SYS_INIT_RAM_ADDR@h
    ori r3,r3,CONFIG_SYS_INIT_RAM_ADDR@l
    mfspr   r2, L1CFG0
    andi.   r2, r2, 0x1ff
    /* cache size * 1024 / (2 * L1 line size) */
    slwi    r2, r2, (10 - 1 - L1_CACHE_SHIFT)
    mtctr   r2
    li  r0,0
1:
    dcbz    r0,r3
    dcbtls  0,r0,r3
    addi    r3,r3,CONFIG_SYS_CACHELINE_SIZE
    bdnz    1b

    /* Jump out the last 4K page and continue to 'normal' start */
```

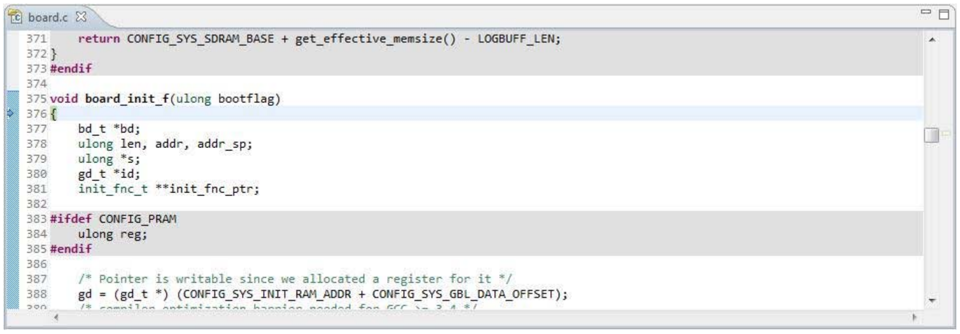
```
effff860:   lis r3,-560
effff864:   ori r3,r3,0x0000
effff868:   mfspr r2,spr515
effff86c:   andi. r2,r2,0x01ff
effff870:   slwi r2,r2,3
effff874:   mtctr r2
effff878:   li r0,0
effff87c:   dcbz r0,r3
effff880:   dcbtls 0,r0,r3
effff884:   addi r3,r3,64
effff888:   bdnz (AsmSection)+0xeffff87c (0xeffff87c); 0
effff88c:   lis r3,-4104
effff890:   ori r3,r3,0x0000
effff894:   addi r3,r3,48
effff898:   mtlr r3
effff89c:   blr
effff8a0:   fnmadd. fp31,fp31,fp31,fp31
effff8a4:   fnmadd. fp31,fp31,fp31,fp31
effff8a8:   fnmadd. fp31,fp31,fp31,fp31
effff8ac:   fnmadd. fp31,fp31,fp31,fp31
```

```
CodeWarrior Debugger Shell v1.0
%>setpicloadaddr 0xfff80000
Debugger now assumes 'u-boot' is loaded at the specified
address.
%>setpicloadaddr reset
Debugger now assumes 'u-boot' is loaded at its link-time
address.
%>
```

3. Now debugging (step, run, or breakpoint) can be done until code is relocated in DDR.

   a) Run to Line: `board_init_f` and Step Into.

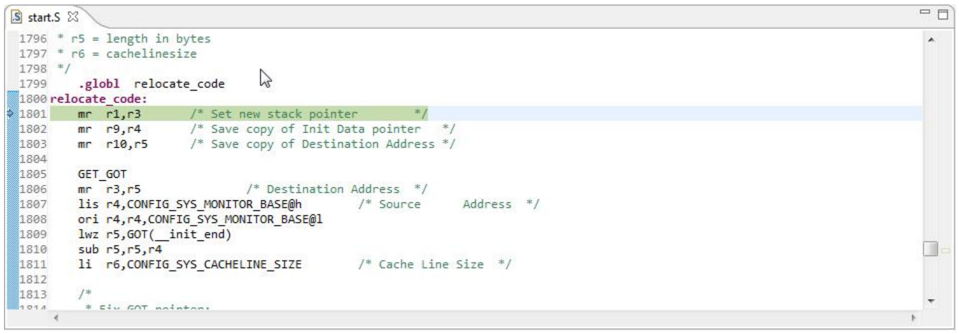
Figure 70. File editor



```
    return CONFIG_SYS_SDRAM_BASE + get_effective_memsize() - LOGBUFF_LEN;
}
#endif

void board_init_f(ulong bootflag)
{
    bd_t *bd;
    ulong len, addr, addr_sp;
    ulong *s;
    gd_t *id;
    init_fnc_t **init_fnc_ptr;

#ifdef CONFIG_PRAM
    ulong reg;
#endif

    /* Pointer is writable since we allocated a register for it */
    gd = (gd_t *) (CONFIG_SYS_INIT_RAM_ADDR + CONFIG_SYS_GBL_DATA_OFFSET);
```

   b) Run to Line: `relocate_code` and Step Into.

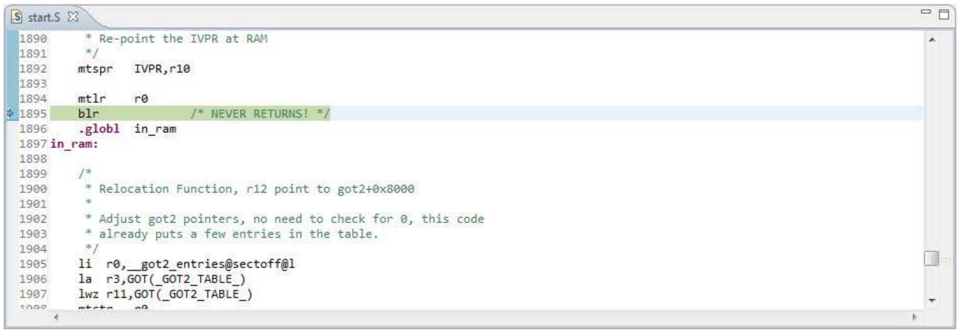
Figure 71. File editor



```
 * r5 = length in bytes
 * r6 = cachelinesize
 */
    .globl  relocate_code
relocate_code:
    mr  r1,r3       /* Set new stack pointer        */
    mr  r9,r4       /* Save copy of Init Data pointer   */
    mr  r10,r5      /* Save copy of Destination Address */

    GET_GOT
    mr  r3,r5               /* Destination Address  */
    lis r4,CONFIG_SYS_MONITOR_BASE@h        /* Source       Address  */
    ori r4,r4,CONFIG_SYS_MONITOR_BASE@l
    lwz r5,GOT(__init_end)
    sub r5,r5,r4
    li  r6,CONFIG_SYS_CACHELINE_SIZE        /* Cache Line Size  */

    /*
```

c) In start.S, last instruction before relocate to DDR is `relocate_code`.

Figure 72. File editor



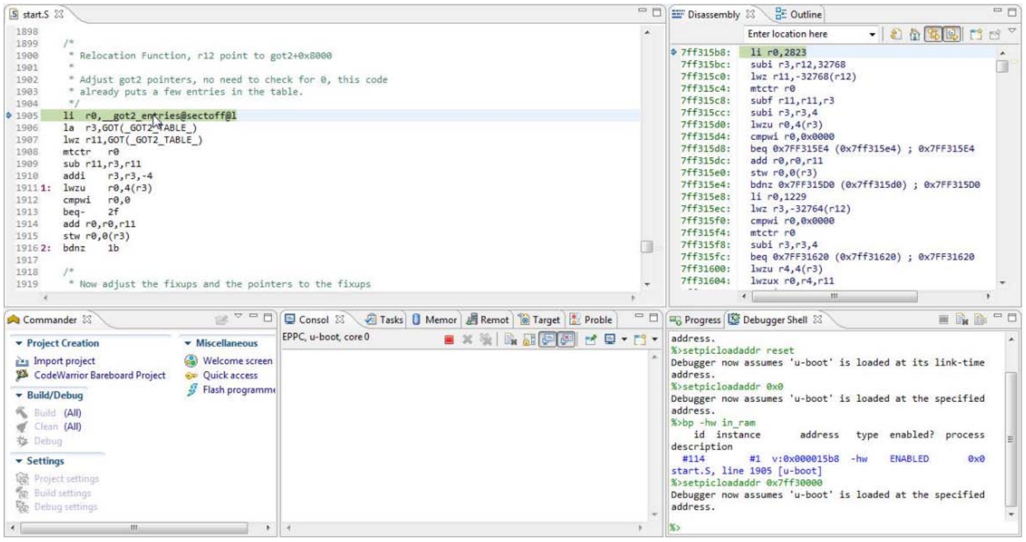
```
     * Re-point the IVPR at RAM
     */
    mtspr   IVPR,r10

    mtlr    r0
    blr                 /* NEVER RETURNS! */
    .globl  in_ram
in_ram:

    /*
     * Relocation Function, r12 point to got2+0x8000
     *
     * Adjust got2 pointers, no need to check for 0, this code
     * already puts a few entries in the table.
     */
    li  r0,__got2_entries@sectoff@l
    la  r3,GOT(_GOT2_TABLE_)
    lwz r11,GOT(_GOT2_TABLE_)
```

d) Step Into `blr`, it shows assembly code. (See Stage 3 – Debug in DDR's higher address, for more information.)

8.2.4. Stage 3 – Debug in DDR's higher address

1. Set PIC load address as `0x7ff30000` using Debugger Shell command `setpicloadaddr 0x7ff30000`. (See How to calculate PIC load address, for more information.)

Figure 73. Debugger shell view



```
    /*
     * Relocation Function, r12 point to got2+0x8000
     *
     * Adjust got2 pointers, no need to check for 0, this code
     * already puts a few entries in the table.
     */
    li  r0,__got2_entries@sectoff@l
    la  r3,GOT(_GOT2_TABLE_)
    lwz r11,GOT(_GOT2_TABLE_)
    mtctr   r0
    sub r11,r3,r11
    addi    r3,r3,-4
1:  lwzu    r0,4(r3)
    cmpwi   r0,0
    beq-    2f
    add r0,r0,r11
    stw r0,0(r3)
2:  bdnz    1b

    /*
     * Now adjust the fixups and the pointers to the fixups
```

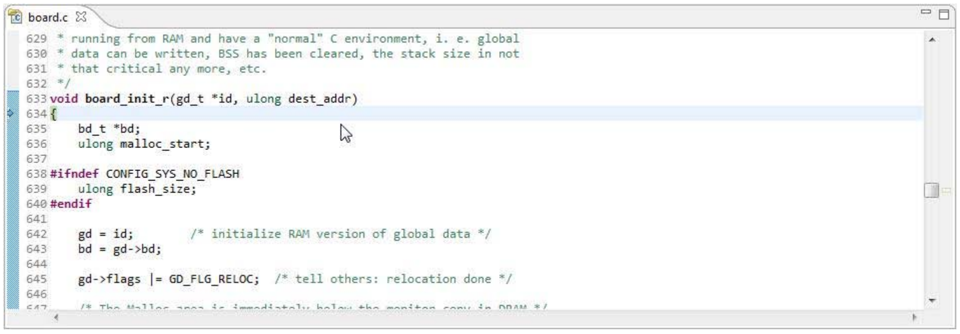
```
7ff315b8:   li r0,2823
7ff315bc:   subi r3,r12,32768
7ff315c0:   lwz r11,-32768(r12)
7ff315c4:   mtctr r0
7ff315c8:   subf r11,r11,r3
7ff315cc:   subi r3,r3,4
7ff315d0:   lwzu r0,4(r3)
7ff315d4:   cmpwi r0,0x0000
7ff315d8:   beq 0x7FF315E4 (0x7ff315e4) ; 0x7FF315E4
7ff315dc:   add r0,r0,r11
7ff315e0:   stw r0,0(r3)
7ff315e4:   bdnz 0x7FF315D0 (0x7ff315d0) ; 0x7FF315D0
7ff315e8:   li r0,1229
7ff315ec:   lwz r3,-32764(r12)
7ff315f0:   cmpwi r0,0x0000
7ff315f4:   mtctr r0
7ff315f8:   subi r3,r3,4
7ff315fc:   beq 0x7FF31620 (0x7ff31620) ; 0x7FF31620
7ff31600:   lwzu r4,4(r3)
7ff31604:   lwzux r0,r4,r11
```

```
address.
%>setpicloadaddr reset
Debugger now assumes 'u-boot' is loaded at its link-time
address.
%>setpicloadaddr 0x0
Debugger now assumes 'u-boot' is loaded at the specified
address.
%>bp -hw in_ram
    id  instance       address   type  enabled?  process
description
  #114        #1  v:0x000015b8  -hw    ENABLED       0x0
start.S, line 1905 [u-boot]
%>setpicloadaddr 0x7ff30000
Debugger now assumes 'u-boot' is loaded at the specified
address.

%>
```

2. We can debug until U-Boot is running.

a) Run to Line: `board_init_r` and do Step into.

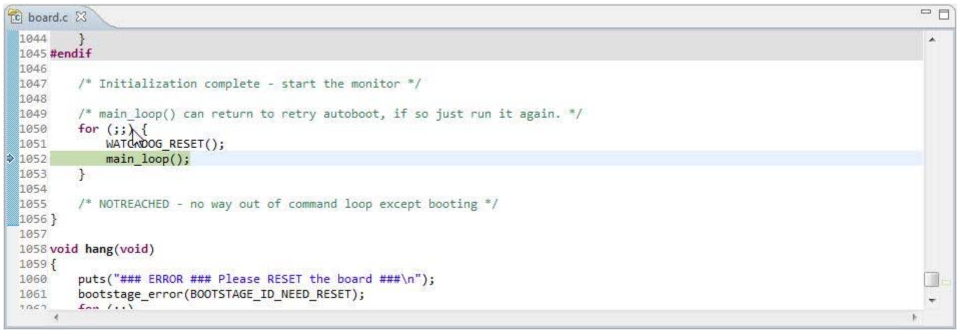
Figure 74. File editor


board.c

```
 * running from RAM and have a "normal" C environment, i. e. global
 * data can be written, BSS has been cleared, the stack size in not
 * that critical any more, etc.
 */
void board_init_r(gd_t *id, ulong dest_addr)
{
    bd_t *bd;
    ulong malloc_start;

#ifndef CONFIG_SYS_NO_FLASH
    ulong flash_size;
#endif

    gd = id;        /* initialize RAM version of global data */
    bd = gd->bd;

    gd->flags |= GD_FLG_RELOC;  /* tell others: relocation done */
```

b) Run to Line: `main_loop()`.

Figure 75. File editor


board.c

```
    }
#endif

    /* Initialization complete - start the monitor */

    /* main_loop() can return to retry autoboot, if so just run it again. */
    for (;;) {
        WATCHDOG_RESET();
        main_loop();
    }

    /* NOTREACHED - no way out of command loop except booting */
}

void hang(void)
{
    puts("### ERROR ### Please RESET the board ###\n");
    bootstage_error(BOOTSTAGE_ID_NEED_RESET);
```

9. Debugging U-Boot from NAND for e500mc

9.1. Debug environment

Setup used for U-Boot NAND debugging on e500mc core:

1. P3041DS Hydra board.
2. Compiled U-Boot for the NAND FLASH target. (See PBL configuration tool, for more information.)
3. Flash U-Boot on the target board. (See SDK documentation, for more information on how to program the U-Boot to NAND flash.)
4. Switches set for NAND boot (See SDK documentation, for more information on how to set switches.)
5. Latest release of CodeWarrior IDE.
6. P3041_uboot_36.tcl initialization file.
7. USB TAP or other probe.

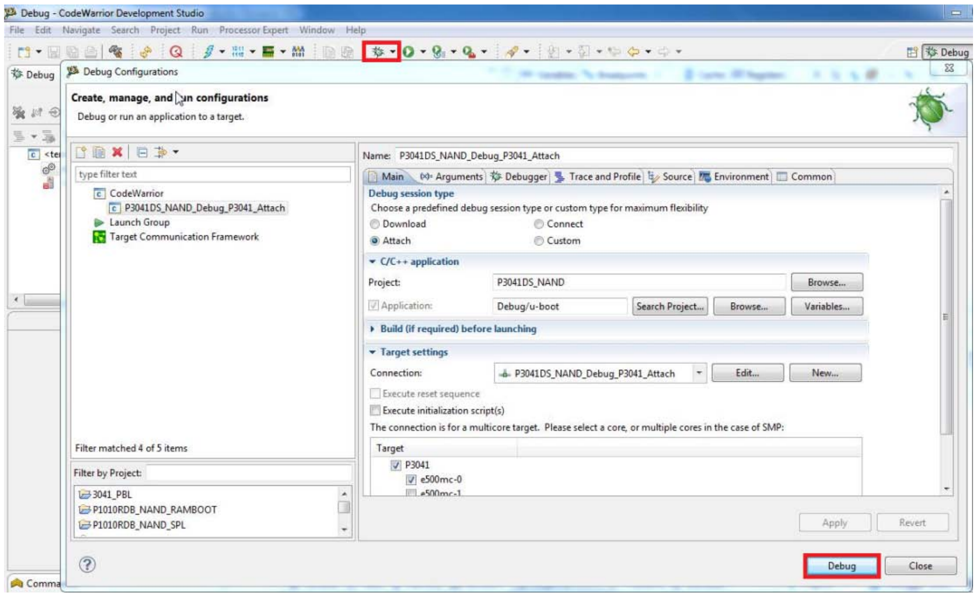
9.2. U-Boot NAND debugging

Import the U-Boot elf file generated during U-Boot compilation as a CodeWarrior project. (See Configuring a CodeWarrior project, for more information.)

9.2.1. Stage 0 – Connect CodeWarrior to board

1. Before debugging, run the project in debug mode.

Figure 76. Debug configurations dialog


Debug - CodeWarrior Development Studio
File Edit Navigate Search Project Run Processor Expert Window Help
Debug
Debug Configurations
Create, manage, and run configurations
Debug or run an application to a target.
type filter text
CodeWarrior
P3041DS_NAND_Debug_P3041_Attach
Launch Group
Target Communication Framework
Filter matched 4 of 5 items
Filter by Project:
3041_PBL
P1010RDB_NAND_RAMBOOT
P1010RDB_NAND_SPL
Main
Arguments
Debugger
Trace and Profile
Source
Environment
Common
Browse...
Search Project...
Variables...
Edit...
New...
Apply
Revert
Debug
Close

Name: P3041DS_NAND_Debug_P3041_Attach

Debug session type
Choose a predefined debug session type or custom type for maximum flexibility
Download
Connect
[✓ Attach]
Custom

C/C++ application
Project: P3041DS_NAND
Application: Debug/u-boot

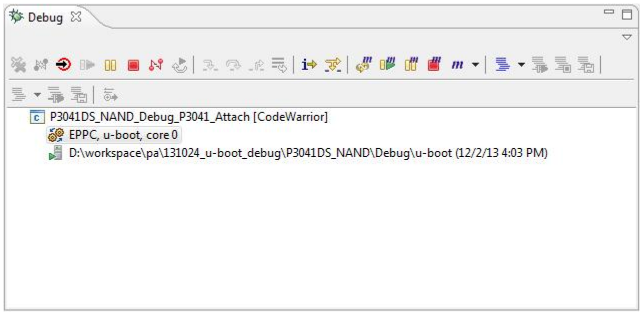
Build (if required) before launching

Target settings
Connection: P3041DS_NAND_Debug_P3041_Attach
Execute reset sequence
Execute initialization script(s)
The connection is for a multicore target. Please select a core, or multiple cores in the case of SMP:

Target
✓ P3041
✓ e500mc-0
e500mc-1

2. Start the debugging session using the setup provided in Figure 76. This initializes and configures the TAP, then attaches to the board.

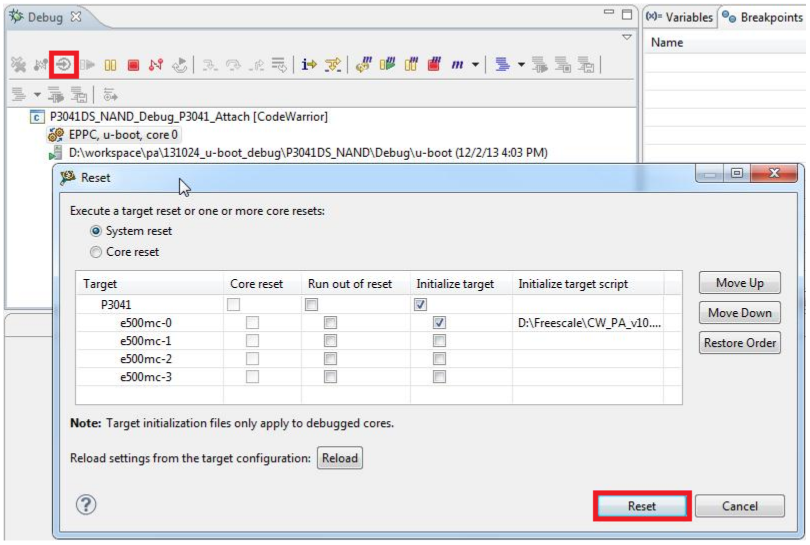
Figure 77. Debug view



P3041DS_NAND_Debug_P3041_Attach [CodeWarrior]
EPPC, u-boot, core 0
D:\workspace\pa\131024_u-boot_debug\P3041DS_NAND\Debug\u-boot (12/2/13 4:03 PM)

3. Reinitialize the target from CodeWarrior, using U-Boot initialization file.

Figure 78. Reset dialog



P3041DS_NAND_Debug_P3041_Attach [CodeWarrior]
EPPC, u-boot, core 0
D:\workspace\pa\131024_u-boot_debug\P3041DS_NAND\Debug\u-boot (12/2/13 4:03 PM)

Reset

Execute a target reset or one or more core resets:
[✓ System reset]
Core reset

Target	Core reset	Run out of reset	Initialize target	Initialize target script
P3041	☐	☐	✓	
e500mc-0	☐	☐	✓	D:\Freescale\CW_PA_v10....
e500mc-1	☐	☐	☐	
e500mc-2	☐	☐	☐	
e500mc-3	☐	☐	☐	

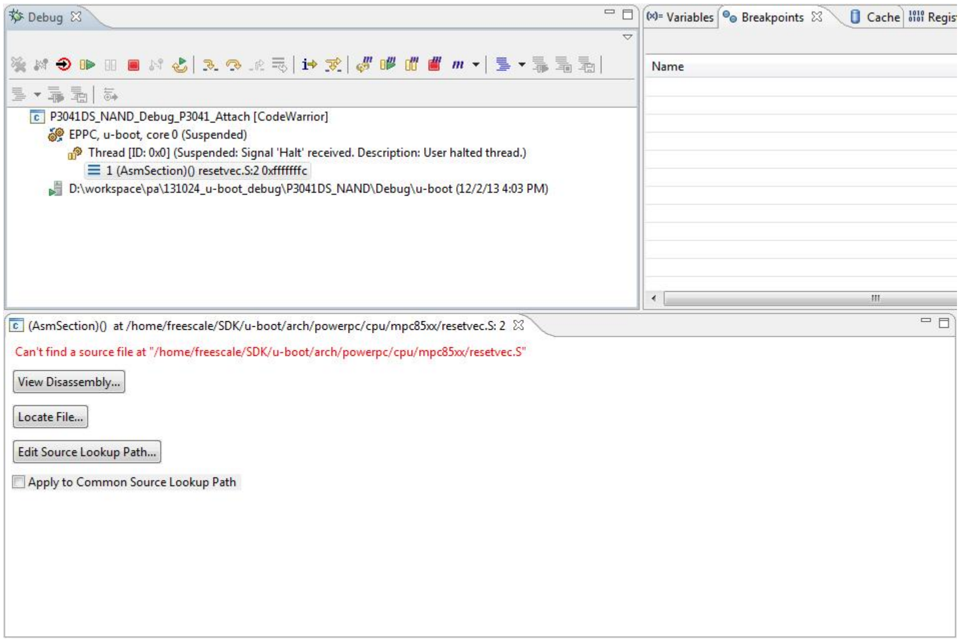
Note: Target initialization files only apply to debugged cores.

Reload settings from the target configuration: Reload

9.2.2. Stage 1 – Debug NAND SRAM for AS0

1. Reset PIC load address using Debugger Shell command `setpicloadaddr reset`.
2. Source code location is asked by CodeWarrior.

Figure 79. Debug view



P3041DS_NAND_Debug_P3041_Attach [CodeWarrior]
EPPC, u-boot, core 0 (Suspended)
Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
1 (AsmSection)() resetvec.S:2 0xfffffffc
D:\workspace\pa\131024_u-boot_debug\P3041DS_NAND\Debug\u-boot (12/2/13 4:03 PM)

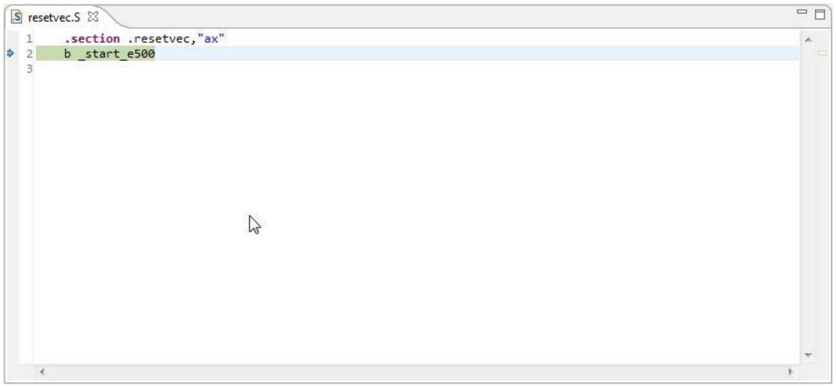
(AsmSection)() at /home/freescale/SDK/u-boot/arch/powerpc/cpu/mpc85xx/resetvec.S: 2

Can't find a source file at "/home/freescale/SDK/u-boot/arch/powerpc/cpu/mpc85xx/resetvec.S"

☐ Apply to Common Source Lookup Path

3. After the path is specified, sources are available in CodeWarrior.

Figure 80. File editor

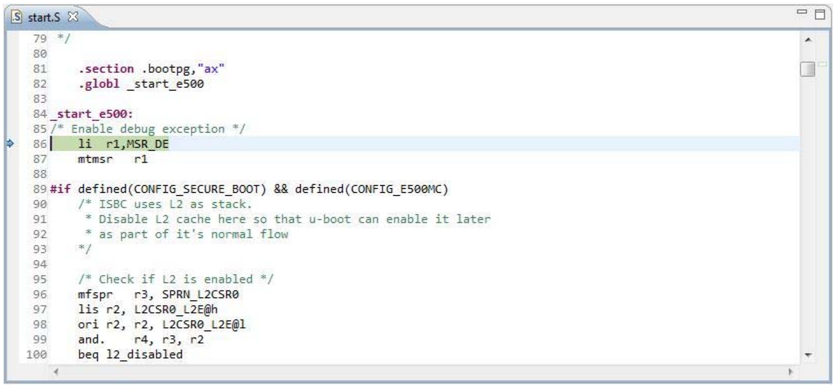


resetvec.S

```
    .section .resetvec,"ax"
    b _start_e500
```

4. Now debugging (step, run, or breakpoint) can be done before switching to AS1.

Figure 81. File editor

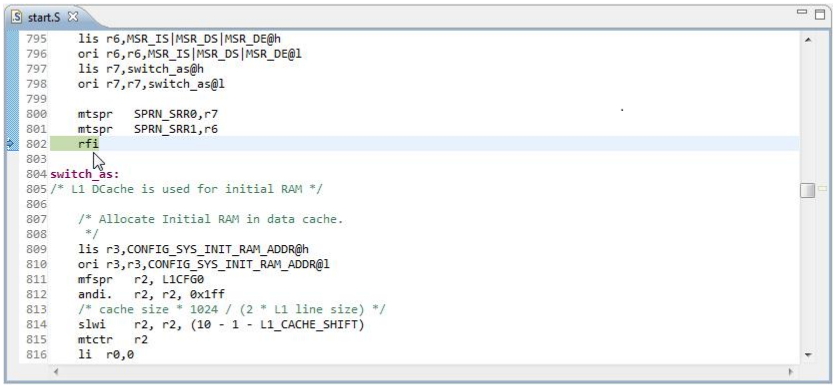


start.S

```
 */

    .section .bootpg,"ax"
    .globl _start_e500

_start_e500:
/* Enable debug exception */
    li  r1,MSR_DE
    mtmsr   r1

#if defined(CONFIG_SECURE_BOOT) && defined(CONFIG_E500MC)
    /* ISBC uses L2 as stack.
     * Disable L2 cache here so that u-boot can enable it later
     * as part of it's normal flow
    */

    /* Check if L2 is enabled */
    mfspr   r3, SPRN_L2CSR0
    lis r2, L2CSR0_L2E@h
    ori r2, r2, L2CSR0_L2E@l
    and.    r4, r3, r2
    beq l2_disabled
```

5. In file start.S, last instruction before moving to AS1 is `rfi` before `switch_as`. (See Stage 2 – Debug NAND for AS1, for more information.)

Figure 82. File editor



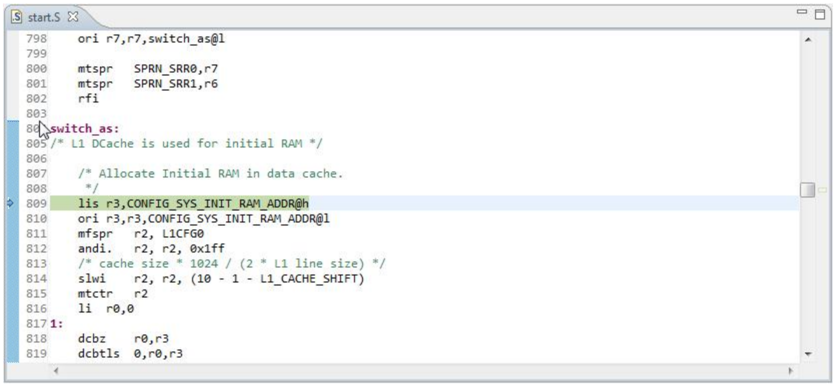
start.S

```
    lis r6,MSR_IS|MSR_DS|MSR_DE@h
    ori r6,r6,MSR_IS|MSR_DS|MSR_DE@l
    lis r7,switch_as@h
    ori r7,r7,switch_as@l

    mtspr   SPRN_SRR0,r7
    mtspr   SPRN_SRR1,r6
    rfi

switch_as:
/* L1 DCache is used for initial RAM */

    /* Allocate Initial RAM in data cache.
     */
    lis r3,CONFIG_SYS_INIT_RAM_ADDR@h
    ori r3,r3,CONFIG_SYS_INIT_RAM_ADDR@l
    mfspr   r2, L1CFG0
    andi.   r2, r2, 0x1ff
    /* cache size * 1024 / (2 * L1 line size) */
    slwi    r2, r2, (10 - 1 - L1_CACHE_SHIFT)
    mtctr   r2
    li  r0,0
```

9.2.3. Stage 2 – Debug NAND for AS1

1. Step Into this instruction.

Figure 83. File editor



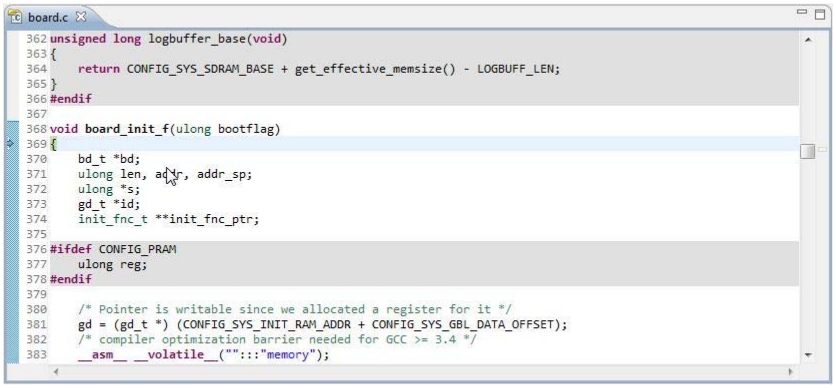
start.S

```
    ori r7,r7,switch_as@l

    mtspr   SPRN_SRR0,r7
    mtspr   SPRN_SRR1,r6
    rfi

switch_as:
/* L1 DCache is used for initial RAM */

    /* Allocate Initial RAM in data cache.
     */
    lis r3,CONFIG_SYS_INIT_RAM_ADDR@h
    ori r3,r3,CONFIG_SYS_INIT_RAM_ADDR@l
    mfspr   r2, L1CFG0
    andi.   r2, r2, 0x1ff
    /* cache size * 1024 / (2 * L1 line size) */
    slwi    r2, r2, (10 - 1 - L1_CACHE_SHIFT)
    mtctr   r2
    li  r0,0
1:
    dcbz    r0,r3
    dcbtls  0,r0,r3
```

2. Now debugging is be possible, before the code is relocated in DDR.

a) Run to Line: `board_init_f` and Step into: `board_init_f`.

Figure 84. File editor

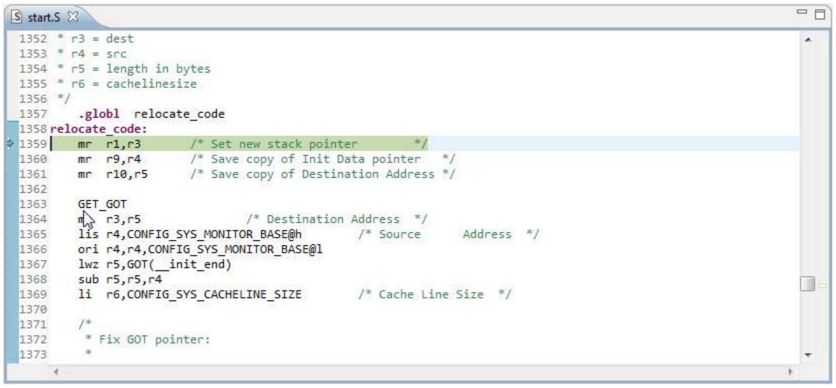


board.c

```
unsigned long logbuffer_base(void)
{
    return CONFIG_SYS_SDRAM_BASE + get_effective_memsize() - LOGBUFF_LEN;
}
#endif

void board_init_f(ulong bootflag)
{
    bd_t *bd;
    ulong len, addr, addr_sp;
    ulong *s;
    gd_t *id;
    init_fnc_t **init_fnc_ptr;

#ifdef CONFIG_PRAM
    ulong reg;
#endif

    /* Pointer is writable since we allocated a register for it */
    gd = (gd_t *) (CONFIG_SYS_INIT_RAM_ADDR + CONFIG_SYS_GBL_DATA_OFFSET);
    /* compiler optimization barrier needed for GCC >= 3.4 */
    __asm__ __volatile__("":::"memory");
```

b) Run to Line: `relocate_code` and do Step Into.

Figure 85. File editor

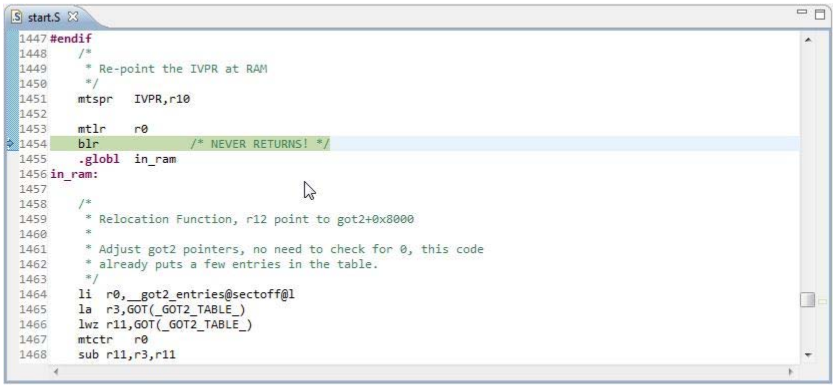


start.S

```
 * r3 = dest
 * r4 = src
 * r5 = length in bytes
 * r6 = cachelinesize
 */
    .globl  relocate_code
relocate_code:
    mr  r1,r3       /* Set new stack pointer       */
    mr  r9,r4       /* Save copy of Init Data pointer   */
    mr  r10,r5      /* Save copy of Destination Address */

    GET_GOT
    mr  r3,r5               /* Destination Address  */
    lis r4,CONFIG_SYS_MONITOR_BASE@h        /* Source      Address  */
    ori r4,r4,CONFIG_SYS_MONITOR_BASE@l
    lwz r5,GOT(__init_end)
    sub r5,r5,r4
    li  r6,CONFIG_SYS_CACHELINE_SIZE        /* Cache Line Size  */

    /*
     * Fix GOT pointer:
     *
```

c) In file start.S, last code before relocate to DDR is `relocate_code`.

Figure 86. File editor



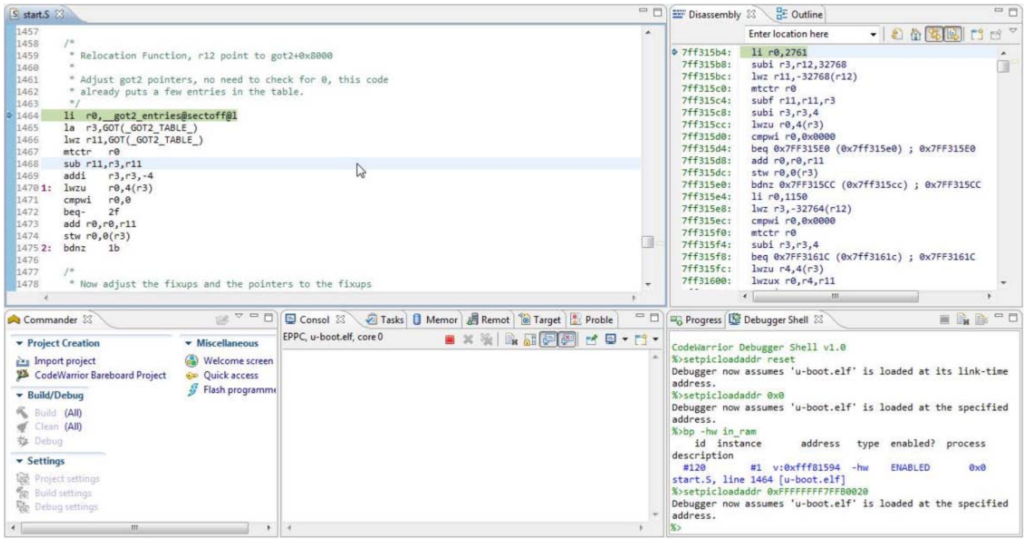
```
#endif
    /*
     * Re-point the IVPR at RAM
     */
    mtspr   IVPR,r10

    mtlr    r0
    blr                 /* NEVER RETURNS! */
    .globl  in_ram
in_ram:

    /*
     * Relocation Function, r12 point to got2+0x8000
     *
     * Adjust got2 pointers, no need to check for 0, this code
     * already puts a few entries in the table.
     */
    li  r0,__got2_entries@sectoff@l
    la  r3,GOT(_GOT2_TABLE_)
    lwz r11,GOT(_GOT2_TABLE_)
    mtctr   r0
    sub r11,r3,r11
```

3. Step Into: `blr`, it shows code in assembly. (See Stage 3 – Debug in RAM, for more information.)

9.2.4. Stage 3 – Debug in RAM

1. Set PIC load address as `0xFFFFFFFF7FFB0020`, using Debugger Shell command `setpicloadaddr 0xFFFFFFFF7FFB0020`. (See How to calculate PIC load address, for more information.)

Figure 87. Debugger shell view



```
    /*
     * Relocation Function, r12 point to got2+0x8000
     *
     * Adjust got2 pointers, no need to check for 0, this code
     * already puts a few entries in the table.
     */
    li  r0,__got2_entries@sectoff@l
    la  r3,GOT(_GOT2_TABLE_)
    lwz r11,GOT(_GOT2_TABLE_)
    mtctr   r0
    sub r11,r3,r11
    addi    r3,r3,-4
1:  lwzu    r0,4(r3)
    cmpwi   r0,0
    beq-    2f
    add r0,r0,r11
    stw r0,0(r3)
2:  bdnz    1b

    /*
     * Now adjust the fixups and the pointers to the fixups
```

```
7ff315b4:   li r0,2761
7ff315b8:   subi r3,r12,32768
7ff315bc:   lwz r11,-32768(r12)
7ff315c0:   mtctr r0
7ff315c4:   subf r11,r11,r3
7ff315c8:   subi r3,r3,4
7ff315cc:   lwzu r0,4(r3)
7ff315d0:   cmpwi r0,0x0000
7ff315d4:   beq 0x7FF315E0 (0x7ff315e0) ; 0x7FF315E0
7ff315d8:   add r0,r0,r11
7ff315dc:   stw r0,0(r3)
7ff315e0:   bdnz 0x7FF315CC (0x7ff315cc) ; 0x7FF315CC
7ff315e4:   li r0,1150
7ff315e8:   lwz r3,-32764(r12)
7ff315ec:   cmpwi r0,0x0000
7ff315f0:   mtctr r0
7ff315f4:   subi r3,r3,4
7ff315f8:   beq 0x7FF3161C (0x7ff3161c) ; 0x7FF3161C
7ff315fc:   lwzu r4,4(r3)
7ff31600:   lwzux r0,r4,r11
```

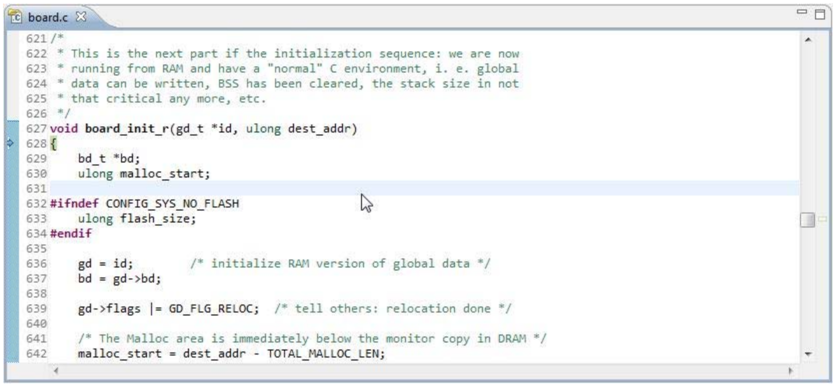
EPPC, u-boot.elf, core 0

```
CodeWarrior Debugger Shell v1.0
%>setpicloadaddr reset
Debugger now assumes 'u-boot.elf' is loaded at its link-time
address.
%>setpicloadaddr 0x0
Debugger now assumes 'u-boot.elf' is loaded at the specified
address.
%>bp -hw in_ram
    id  instance       address   type  enabled?  process
description
  #120        #1  v:0xfff81594  -hw    ENABLED       0x0
start.S, line 1464 [u-boot.elf]
%>setpicloadaddr 0xFFFFFFFF7FFB0020
Debugger now assumes 'u-boot.elf' is loaded at the specified
address.
%>
```

2. Debug until U-Boot code is copied from NAND to RAM and control is transferred to it.

   a) Run to Line: `board_init_r` and do Step Into.

Figure 88. File editor

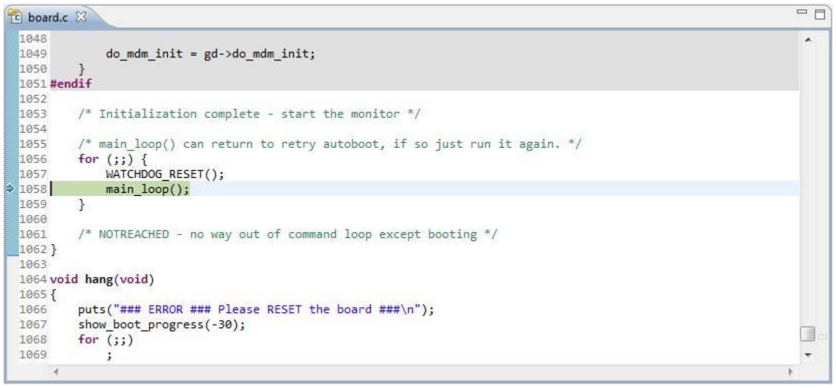


board.c

```
/*
 * This is the next part if the initialization sequence: we are now
 * running from RAM and have a "normal" C environment, i. e. global
 * data can be written, BSS has been cleared, the stack size in not
 * that critical any more, etc.
 */
void board_init_r(gd_t *id, ulong dest_addr)
{
    bd_t *bd;
    ulong malloc_start;

#ifndef CONFIG_SYS_NO_FLASH
    ulong flash_size;
#endif

    gd = id;        /* initialize RAM version of global data */
    bd = gd->bd;

    gd->flags |= GD_FLG_RELOC;  /* tell others: relocation done */

    /* The Malloc area is immediately below the monitor copy in DRAM */
    malloc_start = dest_addr - TOTAL_MALLOC_LEN;
```

b) Step Into: `main_loop()` function.

Figure 89. File editor



board.c

```

        do_mdm_init = gd->do_mdm_init;
    }
#endif

    /* Initialization complete - start the monitor */

    /* main_loop() can return to retry autoboot, if so just run it again. */
    for (;;) {
        WATCHDOG_RESET();
        main_loop();
    }

    /* NOTREACHED - no way out of command loop except booting */
}

void hang(void)
{
    puts("### ERROR ### Please RESET the board ###\n");
    show_boot_progress(-30);
    for (;;)
        ;
```

10. Debugging U-Boot from SPI/SD/MMC for e500mc

Booting from SPI and SD\MMC are similar, the only difference between these is, how the final image is build. This chapter provides steps for SPI U-Boot debugging.

10.1. Debugging environment

Given below is the setup used for U-Boot debugging on e500mc core:

1. Compiled U-Boot for SPI FLASH target.
2. Flash U-Boot on the target board. (For more information, see Chapter 7.6.1.1 Using the Boot Format Tool, of Targeting PA Processor.pdf)
3. Switches set for SPI boot. (See SDK documentation, for more information on how to set switches.)
4. Latest release of CodeWarrior IDE.
5. P3041_uboot_36.tcl initialization file.
6. USB TAP or other probe.

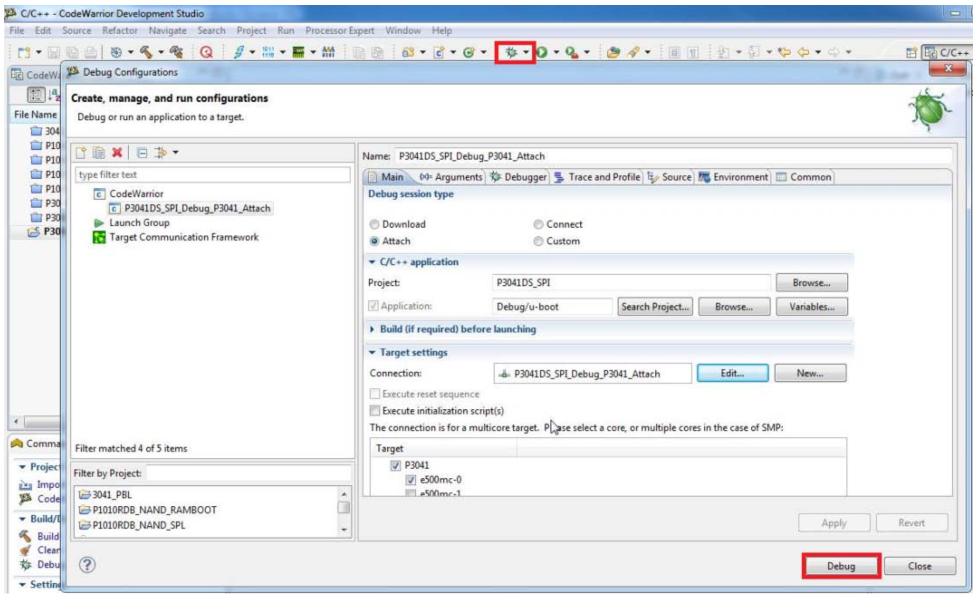
10.2. U-Boot SPI debugging

Import the U-Boot elf file generated during U-Boot compilation as CodeWarrior project. (See Configuring a CodeWarrior project, for more information.)

10.2.1. Stage 0 – Connect CodeWarrior to board

1. Before debugging, run the board in debug mode.

Figure 90. Debug configurations dialog



Debug Configurations

Create, manage, and run configurations

Debug or run an application to a target.

Name: P3041DS_SPI_Debug_P3041_Attach

Debug session type: Attach

C/C++ application

Project: P3041DS_SPI

Application: Debug/u-boot

Target settings

Connection: P3041DS_SPI_Debug_P3041_Attach

Execute reset sequence

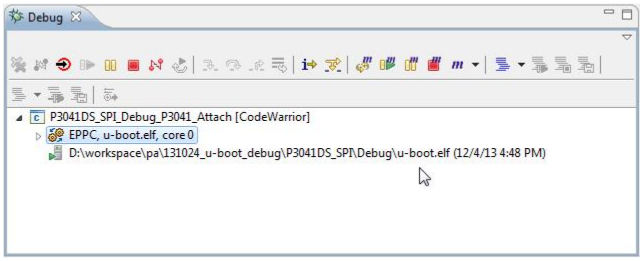
Execute initialization script(s)

The connection is for a multicore target. Please select a core, or multiple cores in the case of SMP:

Target: P3041 [✓], e500mc-0 [✓]

2. Start the debugging session using the setup provided in Figure 87. This initializes and configures the TAP, then attaches to the board.

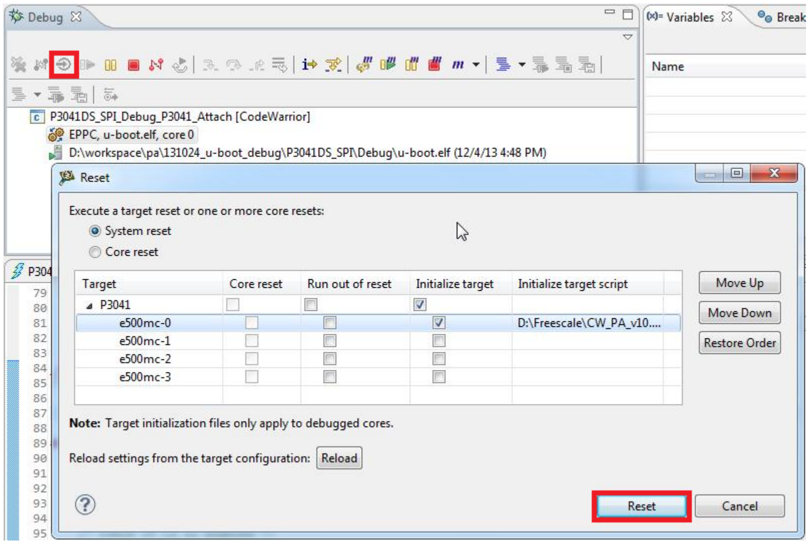
Figure 91. Debug view



```
P3041DS_SPI_Debug_P3041_Attach [CodeWarrior]
  EPPC, u-boot.elf, core 0
  D:\workspace\pa\131024_u-boot_debug\P3041DS_SPI\Debug\u-boot.elf (12/4/13 4:48 PM)
```

3. Reinitialize the target from CodeWarrior.

Figure 92. Reset dialog



P3041DS_SPI_Debug_P3041_Attach [CodeWarrior]
EPPC, u-boot.elf, core 0
D:\workspace\pa\131024_u-boot_debug\P3041DS_SPI\Debug\u-boot.elf (12/4/13 4:48 PM)

Reset

Execute a target reset or one or more core resets:
[✓ System reset]
Core reset

Target	Core reset	Run out of reset	Initialize target	Initialize target script
P3041	☐	☐	✓	
e500mc-0	☐	☐	✓	D:\Freescale\CW_PA_v10....
e500mc-1	☐	☐	☐	
e500mc-2	☐	☐	☐	
e500mc-3	☐	☐	☐	

Note: Target initialization files only apply to debugged cores.

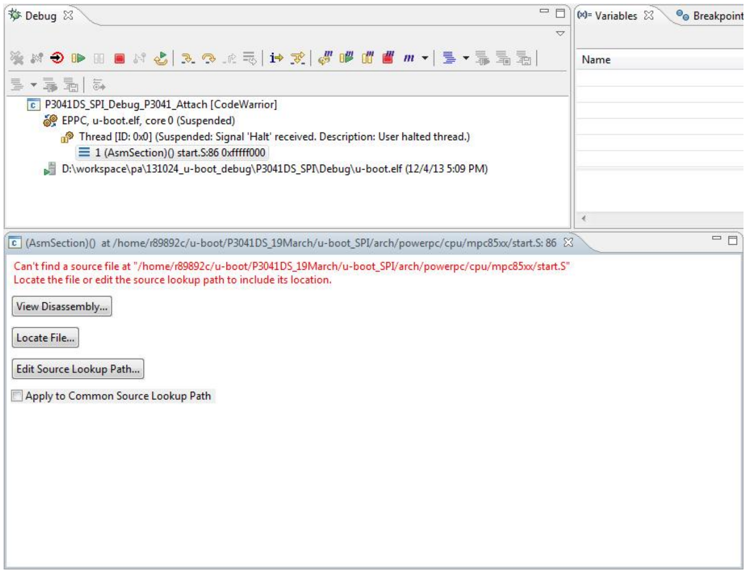
Reload settings from the target configuration: Reload

NOTE If `Reset Failed` error appears go to **Debug Configurations**, edit **Target settings connection**, and then go to **Advanced** tab, select **Reset delay (ms)** and set the value to `2000`.

10.2.2. Stage 1 – Debug SPI until U-Boot relocated to DDR's higher address

1. Reset PIC load address, using Debugger Shell command `setpicloadaddr reset`.
2. Source code location is asked by CodeWarrior.

Figure 93. Debug view



P3041DS_SPI_Debug_P3041_Attach [CodeWarrior]
EPPC, u-boot.elf, core 0 (Suspended)
Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
1 (AsmSection)() start.S:86 0xfffff000
D:\workspace\pa\131024_u-boot_debug\P3041DS_SPI\Debug\u-boot.elf (12/4/13 5:09 PM)

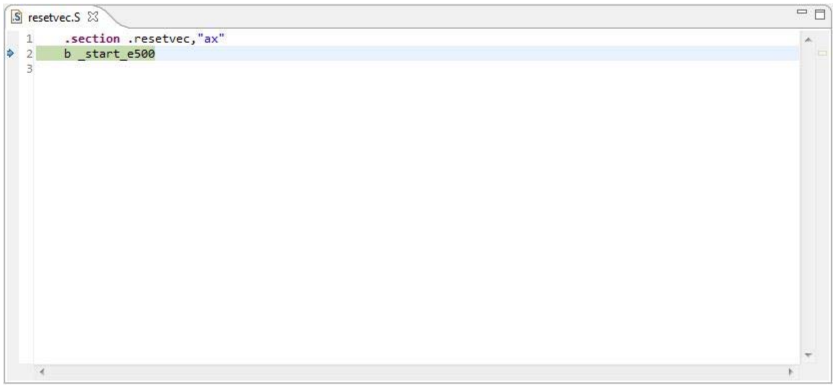
(AsmSection)() at /home/r89892c/u-boot/P3041DS_19March/u-boot_SPI/arch/powerpc/cpu/mpc85xx/start.S: 86

Can't find a source file at "/home/r89892c/u-boot/P3041DS_19March/u-boot_SPI/arch/powerpc/cpu/mpc85xx/start.S"
Locate the file or edit the source lookup path to include its location.

☐ Apply to Common Source Lookup Path

3. After the path is specified, source will be in CodeWarrior.

Figure 94. File editor



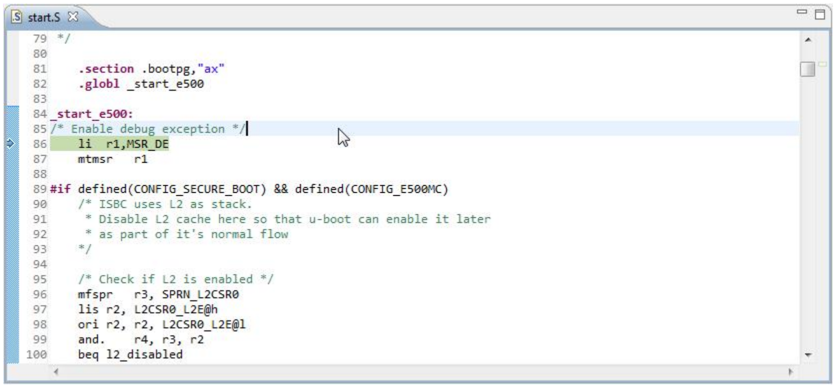
resetvec.S

```
	.section .resetvec,"ax"
	b _start_e500
```

4. Now debugging (step, run, or breakpoint) can be done until U-Boot code will be relocated to the higher address of DDR.

   a) Step Into:

Figure 95. File editor

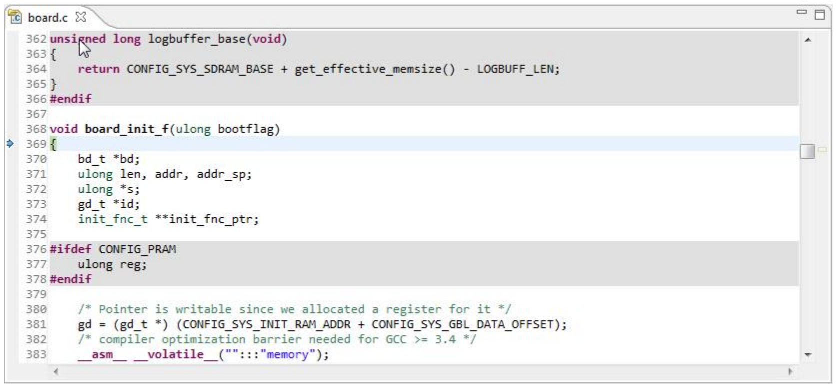


start.S

```
 */

	.section .bootpg,"ax"
	.globl _start_e500

_start_e500:
/* Enable debug exception */
	li	r1,MSR_DE
	mtmsr	r1

#if defined(CONFIG_SECURE_BOOT) && defined(CONFIG_E500MC)
	/* ISBC uses L2 as stack.
	 * Disable L2 cache here so that u-boot can enable it later
	 * as part of it's normal flow
	*/

	/* Check if L2 is enabled */
	mfspr	r3, SPRN_L2CSR0
	lis r2, L2CSR0_L2E@h
	ori r2, r2, L2CSR0_L2E@l
	and.	r4, r3, r2
	beq l2_disabled
```

   b) Run to Line: `board_ini_f` and do Step Into.

Figure 96. File editor

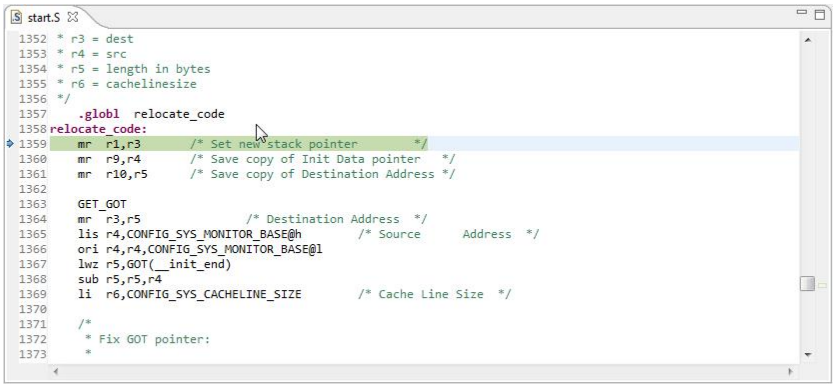


board.c

```
unsigned long logbuffer_base(void)
{
	return CONFIG_SYS_SDRAM_BASE + get_effective_memsize() - LOGBUFF_LEN;
}
#endif

void board_init_f(ulong bootflag)
{
	bd_t *bd;
	ulong len, addr, addr_sp;
	ulong *s;
	gd_t *id;
	init_fnc_t **init_fnc_ptr;

#ifdef CONFIG_PRAM
	ulong reg;
#endif

	/* Pointer is writable since we allocated a register for it */
	gd = (gd_t *) (CONFIG_SYS_INIT_RAM_ADDR + CONFIG_SYS_GBL_DATA_OFFSET);
	/* compiler optimization barrier needed for GCC >= 3.4 */
	__asm__ __volatile__("":::"memory");
```

c) Run to Line: `relocate_code` and do Step Into.

Figure 97. File editor

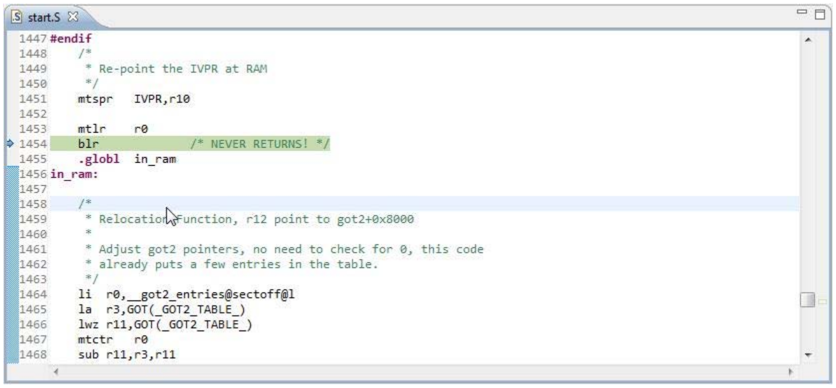


start.S

```
 * r3 = dest
 * r4 = src
 * r5 = length in bytes
 * r6 = cachelinesize
 */
    .globl  relocate_code
relocate_code:
    mr   r1,r3          /* Set new stack pointer         */
    mr   r9,r4          /* Save copy of Init Data pointer   */
    mr   r10,r5         /* Save copy of Destination Address */

    GET_GOT
    mr   r3,r5                      /* Destination Address  */
    lis r4,CONFIG_SYS_MONITOR_BASE@h        /* Source      Address  */
    ori r4,r4,CONFIG_SYS_MONITOR_BASE@l
    lwz r5,GOT(__init_end)
    sub r5,r5,r4
    li  r6,CONFIG_SYS_CACHELINE_SIZE        /* Cache Line Size  */

    /*
     * Fix GOT pointer:
     *
```

d) In file start.S, last instruction before moving to DDR's higher address is `relocate_code`.

Figure 98. File editor



start.S

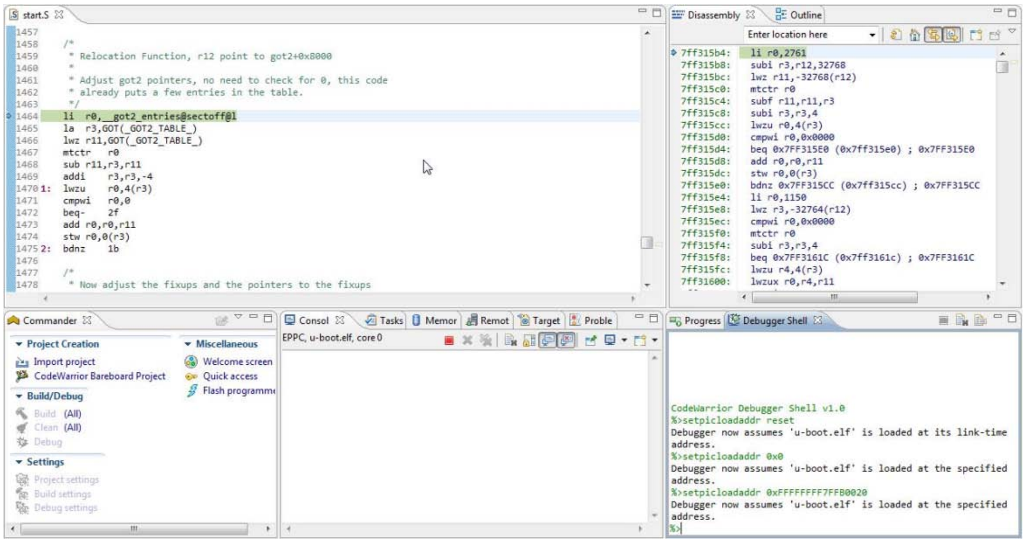
```
#endif
    /*
     * Re-point the IVPR at RAM
     */
    mtspr   IVPR,r10

    mtlr    r0
    blr                 /* NEVER RETURNS! */
    .globl  in_ram
in_ram:

    /*
     * Relocation Function, r12 point to got2+0x8000
     *
     * Adjust got2 pointers, no need to check for 0, this code
     * already puts a few entries in the table.
     */
    li  r0,__got2_entries@sectoff@l
    la  r3,GOT(_GOT2_TABLE_)
    lwz r11,GOT(_GOT2_TABLE_)
    mtctr   r0
    sub r11,r3,r11
```

e) Step Into: `blr`, it shows code in assembly. (See Stage 2 – Debug SPI in DDR's higher address, for more information.)

10.2.3. Stage 2 – Debug SPI in DDR's higher address

1. Set PIC load address as `0xfffffffff7ffb0020` using Debugger Shell command `setpicloadaddr 0xfffffffff7ffb0020`. (See How to calculate PIC load address, for more information.)

Figure 99. Debugger shell view



```
    /*
     * Relocation Function, r12 point to got2+0x8000
     *
     * Adjust got2 pointers, no need to check for 0, this code
     * already puts a few entries in the table.
     */
    li  r0,__got2_entries@sectoff@l
    la  r3,GOT(_GOT2_TABLE_)
    lwz r11,GOT(_GOT2_TABLE_)
    mtctr   r0
    sub r11,r3,r11
    addi    r3,r3,-4
1:  lwzu    r0,4(r3)
    cmpwi   r0,0
    beq-    2f
    add r0,r0,r11
    stw r0,0(r3)
2:  bdnz    1b

    /*
     * Now adjust the fixups and the pointers to the fixups
```

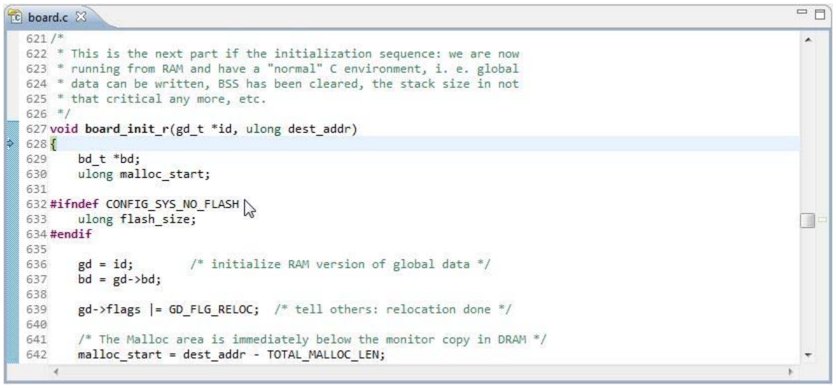
```
7ff315b4:   li r0,2761
7ff315b8:   subi r3,r12,32768
7ff315bc:   lwz r11,-32768(r12)
7ff315c0:   mtctr r0
7ff315c4:   subf r11,r11,r3
7ff315c8:   subi r3,r3,4
7ff315cc:   lwzu r0,4(r3)
7ff315d0:   cmpwi r0,0x0000
7ff315d4:   beq 0x7FF315E0 (0x7ff315e0) ; 0x7FF315E0
7ff315d8:   add r0,r0,r11
7ff315dc:   stw r0,0(r3)
7ff315e0:   bdnz 0x7FF315CC (0x7ff315cc) ; 0x7FF315CC
7ff315e4:   li r0,1150
7ff315e8:   lwz r3,-32764(r12)
7ff315ec:   cmpwi r0,0x0000
7ff315f0:   mtctr r0
7ff315f4:   subi r3,r3,4
7ff315f8:   beq 0x7FF3161C (0x7ff3161c) ; 0x7FF3161C
7ff315fc:   lwzu r4,4(r3)
7ff31600:   lwzux r0,r4,r11
```

```
EPPC, u-boot.elf, core 0
```

```
CodeWarrior Debugger Shell v1.0
%>setpicloadaddr reset
Debugger now assumes 'u-boot.elf' is loaded at its link-time
address.
%>setpicloadaddr 0x0
Debugger now assumes 'u-boot.elf' is loaded at the specified
address.
%>setpicloadaddr 0xFFFFFFFF7FFB0020
Debugger now assumes 'u-boot.elf' is loaded at the specified
address.
%>
```

2. Run to Line: `board_init_r` and do Step Into.

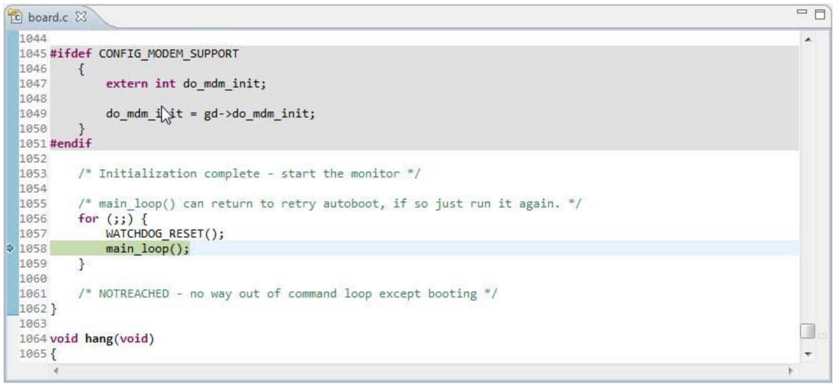
Figure 100. File editor



```
/*
 * This is the next part if the initialization sequence: we are now
 * running from RAM and have a "normal" C environment, i. e. global
 * data can be written, BSS has been cleared, the stack size in not
 * that critical any more, etc.
 */
void board_init_r(gd_t *id, ulong dest_addr)
{
    bd_t *bd;
    ulong malloc_start;

#ifndef CONFIG_SYS_NO_FLASH
    ulong flash_size;
#endif

    gd = id;        /* initialize RAM version of global data */
    bd = gd->bd;

    gd->flags |= GD_FLG_RELOC;  /* tell others: relocation done */

    /* The Malloc area is immediately below the monitor copy in DRAM */
    malloc_start = dest_addr - TOTAL_MALLOC_LEN;
```

3. Run to Line: `main_loop()`.

Figure 101. File editor



```

#ifdef CONFIG_MODEM_SUPPORT
    {
        extern int do_mdm_init;

        do_mdm_init = gd->do_mdm_init;
    }
#endif

    /* Initialization complete - start the monitor */

    /* main_loop() can return to retry autoboot, if so just run it again. */
    for (;;) {
        WATCHDOG_RESET();
        main_loop();
    }

    /* NOTREACHED - no way out of command loop except booting */
}

void hang(void)
{
```

11. How to calculate PIC load address

To set the PIC load address, apply this formula:

```
PIC address = Runtime symbol address (RAM symbol address in our case)
- Compile time symbol address
```

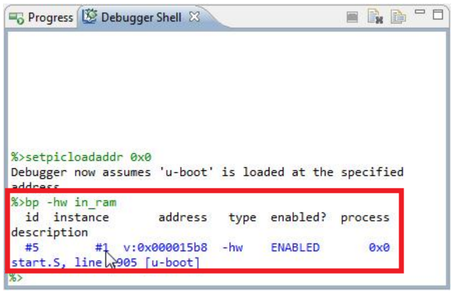
After Step Into: *blr* , in Debugger Shell perform these operations:

1. `%>setpicloadaddr 0x0`: It tells the debugger that the main executables are loaded at `0x0`.

NOTE	This is not the same as `setpicloadaddr reset` command, which tells the debugger that the main executables are loaded at the address set in the ELF.

2. `%>bp –hw in_ram`: It shows the compile time symbol address.

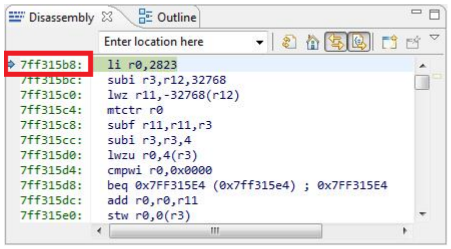
Figure 102. Debugger shell view



```
%>setpicloadaddr 0x0
Debugger now assumes 'u-boot' is loaded at the specified
address
%>bp -hw in_ram
  id  instance       address   type  enabled?  process
description
  #5        #1  v:0x000015b8  -hw    ENABLED       0x0
start.S, line 905 [u-boot]
%>
```

3. Calculate the difference between PC address (single step after blr instruction) and compile time symbol address.

Figure 103. Disassembly view



```
7ff315b8:   li r0,2823
7ff315bc:   subi r3,r12,32768
7ff315c0:   lwz r11,-32768(r12)
7ff315c4:   mtctr r0
7ff315c8:   subf r11,r11,r3
7ff315cc:   subi r3,r3,4
7ff315d0:   lwzu r0,4(r3)
7ff315d4:   cmpwi r0,0x0000
7ff315d8:   beq 0x7FF315E4 (0x7ff315e4) ; 0x7FF315E4
7ff315dc:   add r0,r0,r11
7ff315e0:   stw r0,0(r3)
```

```
PIC address = 0X7FF315B8 (PC address) – 0x000015B8 (in_ram break
point address) = 0x7FF30000.
```

12. Troubleshooting Tips

This section explains:

12.1. Selecting the correct breakpoint type

To avoid issues with incorrect interpretation of memory access during the various U-Boot stages, ensure you use hardware breakpoints only when you have successfully reached the first breakpoint in RAM. The debugger tries to do modify the target memory map and breakpoints, but you can avoid risks by sticking to hardware breakpoints during initial bring-up.

12.2. Risky memory maps

Some SoCs do not provide access to invalid memory ranges and get locked due to unfinished transactions. In such cases, the debug session needs to be restarted. When performing early U-Boot, consider the following points:

- Do not open the **Memory** or **Memory Browser** views for ranges that are not actually readable yet and do not leave them open if you know that the next reset will render them inaccessible.
- For U-Boot debug, your debugger `init` script should be nearly empty, but it should contain at least a `reg sp=1` line. This prohibits the debugger in the very early stages from trying to show a stack back trace that causes invalid accesses, if there is no stack yet.

12.3. Setting multiple hardware breakpoints

The number of active hardware breakpoints is limited, but you can use the **Breakpoints** view to disable those that are not relevant right now and then add more. This way you can create a library of breakpoints that persists across project debug cycles. Whenever you need a specific one, you can enable it and disable others to stay within the limits of the available hardware breakpoints. Also remember that the debugger requires a free hardware breakpoint to do specific operations like `step over`. To avoid error messages, monitor how many hardware breakpoints you have enabled at a specific point of time.

12.4. Skipping U-Boot stages effectively

Remember that `setpicloadaddr` automatically relocates all active source related breakpoints to the space where a PIC executable is loaded. This means that you can pick a specific breakpoint from your library of source related hardware breakpoints and use `setpicloadaddr` to instantiate it for an appropriate stage of U-Boot debug. For example, if you have determined that U-Boot will relocate to `0x7ff30000` in RAM, run the following sequence:

1. reset hard
2. `%>bp -hw in_ram`: Assuming this breakpoint is not yet enabled in your **Breakpoints** view.
3. `%>setpicloadaddr 0x7ff30000`: It instantiates the hardware breakpoint at the right address. Check the **Breakpoints** view.
4. `%>go`: It runs through all the various memory map changes and stops on `in_ram` breakpoint in RAM.

Similarly, you can go straight to `board_init_f` breakpoint:

1. reset hard
2. `%>bp -hw board_init_f`: Assuming that this breakpoint is not yet enabled in your **Breakpoints** view.
3. `%>setpicloadaddr reset`: For a NOR flash setup, `board_init_f` runs in the address range to which U-Boot was linked to. So, reset is ok.
4. `%>go`*:* It runs through all the various memory map changes and stops on `board_init_f` breakpoint in NOR.

12.5. Setting correct absolute addresses

Absolute hex addresses shown in this application note for the `setpiclaodaddr` command or breakpoint operations are common for Freescale provided setups. For example, a 512KB U-Boot starts in NOR flash at `0xfff80000` and is linked to `0xeff80000`. Relocation to RAM is based on RAM sizes. All these perceived absolute values can change depending on the U-Boot size and configuration. So, if your U-Boot configuration differs from the one shown, adjust the addresses used appropriately. Go manually from one debugging stage to another debugging stage during debug, and you will see to what extent addresses may be different for your setup. Then you will know all the required values for subsequent runs.

12.6. Secure Boot and U-Boot debug

When using Secure Boot, remember that ESBC starts at a different virtual address as configured using CSF after ISBC has verified it, and not from `0xfffffffc`. If you try to debug U-Boot without considering this, the debugger shows you the ESBC code starting at `0xfffffffc` when it is internally executing an invisible ROM ISBC at those addresses. This shows a discrepancy in the assembly code and execution behavior. If by using CSF you get, for example, `0xcffffffc` as ESBC entry vector, then set an initial hardware breakpoint on the ESBC entry and adjust the source mapping with `setpiclaodaddr` appropriately. Then run from the original reset vector to your breakpoint and skip the invisible ISBC from ROM completely. This procedure can also be entered into lines of a debugger initialization files so that ISBC is automatically skipped when you start debugging ESBC.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

Document Number: AN4876

28 April 2014