



Changing the ROM Starting Address for RedBoot

(IXP425 Based Platforms)

Background

As on all ARM based processors, the RedBoot ROM image for IXP425 based platforms starts execution at address 0x00000000. The entry point for RedBoot is contained in the file

`<ecos_repository>/packages/hal/arm/arch/current/src/vectors.S`¹. The first instruction executed loads the program counter with the reset vector address stored at an offset of 0x20 bytes into the image. Execution then continues at the reset vector (designated by the label `reset_vector`: in the boot file). By default the reset vector is located at address 0x00000040.

When the IXP425 processor is brought out of reset, the Expansion Bus (where the ROM memory space is found) is temporarily located starting at address 0x00000000. This overlaps the SDRAM memory space, so boot code is responsible for remapping the Expansion Bus to address 0x50000000 by clearing the MEM_MAP bit in the Expansion Bus Configuration Register 0. Because of this, RedBoot ROM images are linked using a starting address 0x50000000.

RedBoot coordinates the remapping of the Expansion Bus through a few actions. The reset vector address stored at an offset of 0x20 bytes into the image is “fixed-up” so that the reset vector is located at address 0x00000040, instead of an address relative to 0x50000000 like the other exception vectors. RedBoot also ensures that no address dependent branch instructions are executed before the Expansion Bus is remapped to address 0x50000000. The code responsible for remapping the Expansion Bus uses a clever trick of executing out of the instruction cache when the actual remap occurs. Running from the instruction cache, prevents the processor from fetching instructions from physical memory during the switch. After the remapping is complete, RedBoot executes a hard-coded branch to an address relative to the new Expansion Bus base of 0x50000000. From this point forward, all instructions fetched from ROM are relative to address 0x50000000.

Modifying the Image Starting Address

Two platform specific files are used by the ECOS configuration tool to generate the linker directive file used during RedBoot ROM builds. Your platform sub tree should be found under the following directory:

```
<ecos_repository>/packages/hal/arm/xscale/<platform>/
```

The platform files used to create the linker directive file are found relative to your platform sub tree as:

```
./current/include/pkgconf/mlt_arm_xscale_<platform>_rom.ldi  
./current/include/pkgconf/mlt_arm_xscale_<platform>_rom.h
```

¹ When using official eCos/RedBoot releases, the sub-directory `current` is replaced by the eCos release number. e.g. `<ecos_repository>/packages/hal/arm/arch/v2_0/src/vectors.S`



For the Generic Residential Gateway platform from ADI Engineering (GRG), the original `mlt_arm_xscale_grg_rom.h` file contains²:

```
// eCos memory layout - Tue Jul 02 10:03:04 2002

// This is a generated file - do not edit

#ifndef __ASSEMBLER__
#include <cyg/infra/cyg_type.h>
#include <stddef.h>

#endif
#define CYGMEM_REGION_ram (0)
#define CYGMEM_REGION_ram_SIZE (0x10000000)
#define CYGMEM_REGION_ram_ATTR (CYGMEM_REGION_ATTR_R | CYGMEM_REGION_ATTR_W)
#define CYGMEM_REGION_rom (0x50000000)
#define CYGMEM_REGION_rom_SIZE (0x40000)
#define CYGMEM_REGION_rom_ATTR (CYGMEM_REGION_ATTR_R)
#ifndef __ASSEMBLER__
extern char CYG_LABEL_NAME (__heap1) [];
#endif
#define CYGMEM_SECTION_heap1 (CYG_LABEL_NAME (__heap1))
#define CYGMEM_SECTION_heap1_SIZE (0x10000000 - (size_t) CYG_LABEL_NAME
(__heap1))
```

Figure 1 Original `mlt_arm_xscale_grg_rom.h`

The first thing needed is to fix the ROM region size. The GRG platform has 16 megabytes of flash memory so the ROM region size is changed from 0x40000 to 0x01000000 (changes are shown in **red**).

```
// eCos memory layout - Tue Jul 02 10:03:04 2002

// This is a generated file - do not edit

#ifndef __ASSEMBLER__
#include <cyg/infra/cyg_type.h>
#include <stddef.h>

#endif
#define CYGMEM_REGION_ram (0)
#define CYGMEM_REGION_ram_SIZE (0x10000000)
#define CYGMEM_REGION_ram_ATTR (CYGMEM_REGION_ATTR_R | CYGMEM_REGION_ATTR_W)
#define CYGMEM_REGION_rom (0x50000000)
#define CYGMEM_REGION_rom_SIZE (0x01000000)
#define CYGMEM_REGION_rom_ATTR (CYGMEM_REGION_ATTR_R)
#ifndef __ASSEMBLER__
extern char CYG_LABEL_NAME (__heap1) [];
#endif
#define CYGMEM_SECTION_heap1 (CYG_LABEL_NAME (__heap1))
#define CYGMEM_SECTION_heap1_SIZE (0x10000000 - (size_t) CYG_LABEL_NAME (__heap1))
```

Figure 2 Modified `mlt_arm_xscale_grg_rom.h`

² Despite the comment shown in the third line of the file, it is safe to edit this file as long as edits are made to the file located in the `<ecos_repository>/packages` directory, and not in the auto-generated build directory.



The original `mlt_arm_xscale_grg_rom.ldi` file for the GRG platform is shown below.

```
// eCos memory layout - Tue Jul 02 10:03:04 2002

// This is a generated file - do not edit

#include <cyg/infra/cyg_type.inc>

MEMORY
{
    ram : ORIGIN = 0, LENGTH = 0x10000000
    rom : ORIGIN = 0x50000000, LENGTH = 0x40000
}

SECTIONS
{
    SECTIONS_BEGIN
    SECTION_rom_vectors (rom, 0x50000000, LMA_EQ_VMA)
    SECTION_text (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_fini (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_rodata (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_rodata1 (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_fixup (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_gcc_except_table (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_mmu_tables (rom, ALIGN (0x4000), LMA_EQ_VMA)
    SECTION_fixed_vectors (ram, 0x20, LMA_EQ_VMA)
    SECTION_data (ram, 0x8000, FOLLOWING (.mmu_tables))
    SECTION_bss (ram, ALIGN (0x4), LMA_EQ_VMA)
    CYG_LABEL_DEFN(__heap1) = ALIGN (0x8);
    SECTIONS_END
}
```

Figure 3 Original `mlt_arm_xscale_grg_rom.ldi`

For this file, the ROM section size also needs to be adjusted from `0x40000` to `0x01000000` to match the changes made to the other file. The new ROM starting address is also changed in this file. In this example, the starting address is changed from `0x50000000` to `0x50300000`.



```
// eCos memory layout - Tue Jul 02 10:03:04 2002

// This is a generated file - do not edit

#include <cyg/infra/cyg_type.inc>

MEMORY
{
    ram : ORIGIN = 0, LENGTH = 0x10000000
    rom : ORIGIN = 0x50000000, LENGTH = 0x01000000
}

SECTIONS
{
    SECTIONS_BEGIN
    SECTION_rom_vectors (rom, 0x50300000, LMA_EQ_VMA)
    SECTION_text (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_fini (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_rodata (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_rodata1 (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_fixup (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_gcc_except_table (rom, ALIGN (0x4), LMA_EQ_VMA)
    SECTION_mmu_tables (rom, ALIGN (0x4000), LMA_EQ_VMA)
    SECTION_fixed_vectors (ram, 0x20, LMA_EQ_VMA)
    SECTION_data (ram, 0x8000, FOLLOWING (.mmu_tables))
    SECTION_bss (ram, ALIGN (0x4), LMA_EQ_VMA)
    CYG_LABEL_DEFN(__heap1) = ALIGN (0x8);
    SECTIONS_END
}
```

Figure 4 Modified `mlt_arm_xscale_grg_rom.ldi`

At this point, if you create a new RedBoot build tree for your platform and rebuild RedBoot, a map file should show that RedBoot now begins at address 0x50300000. However, if this image is programmed to address 0x50300000 in the ROM, and execution is transferred to RedBoot using external boot or POST code, execution of RedBoot will fail. The reset vector must be corrected in order to ensure proper operation.

Fixing the Reset Vector

When starting execution of RedBoot at a new address, using external boot or POST code, there are two possible entry conditions. The ROM address space could be mapped to address 0x0000000, or it could be mapped to address 0x5000000.

After making the changes shown in the previous section, the link address of RedBoot was changed, but the reset vector is still “fixed up” and points to the hard-coded address 0x00000040. Because of the two possible entry conditions, this address may be located in SDRAM memory, or located within the external POST code.

In order to interoperate with both entry conditions into RedBoot, the execution must be transferred to the reset vector through a simple jump instead of using an indirect branch. RedBoot supports this operation by defining the macro `CYGSEM_HAL_ROM_RESET_USES_JUMP`. This macro can simply be added to the following file in your platform sub-tree:

```
./current/include/hal_platform_setup.h
```



The recommended place to add the new macro is immediately following the `CYGHWR_HAL_ARM_HAS_MMU` already in your file. For example, from the GRG platform file:

```
//...
#if defined(CYG_HAL_STARTUP_ROM)
#define PLATFORM_SETUP1 _platform_setup1
#define PLATFORM_EXTRAS <cyg/hal/hal_platform_extras.h>
#define CYGHWR_HAL_ARM_HAS_MMU
#define CYGSEM_HAL_ROM_RESET_USES_JUMP
//...
```

Figure 5 Modified `hal_platform_setup.h`

With this change, RedBoot executes a simple jump forward to an offset of 0x40 bytes from the image start and execution continues at the reset vector. This will work regardless of whether the ROM is currently mapped to address 0x00000000 or address 0x50000000.

At this point, if you create a new build tree and make RedBoot, the final binary image created can be programmed to address 0x50300000 and started using an external boot loader or POST code.

Patching vectors.S

As a final step, a change is required to the boot file

`<ecos_repository>/packages/hal/arm/arch/current/src/vectors.S` in order to fix the vector table stored in SDRAM after RedBoot is running.

RedBoot's support for the `CYGSEM_HAL_ROM_RESET_USES_JUMP` macro is incomplete. Regardless of the presence of this macro, RedBoot simply copies the vector as is from ROM to SDRAM. When the `CYGSEM_HAL_ROM_RESET_USES_JUMP` macro is defined, this breaks RedBoot's support for installing a custom vector service routine (VSR) for the reset vector. To fix the VSR support, the vector table in RAM needs to revert to using the branch indirect mechanism for calling vector service routines. The following lines shown in **red**, need to be added to `vectors.S`, near line 388.



```
#if defined(CYG_HAL_STARTUP_ROM) || defined(CYG_HAL_STARTUP_ROMRAM)
    // Set up reset vector
    mov    r0,#0
    ldr    r1,.__exception_handlers
#ifdef CYGSEM_HAL_ROM_RESET_USES_JUMP
    // When the ROM_RESET_USES_JUMP macro is enabled, the ROM
    // reset vector contains a simple branch instruction. When
    // we copy the vector table into RAM, we need to fix the
    // reset vector to use the branch indirect method so that
    // the application can modify the virtual vector if needed.
    // Make use of the fact that with the branch indirect, the
    // instruction used for every exception type is identical
    // so copy the ROM undefined instruction vector to the
    // RAM reset vector.
    ldr    r2,[r1,#0x04]    // Copy the undefined vector instruction
    str    r2,[r0,#0x00]
#else
    ldr    r2,[r1,#0x00]    // reset vector instruction
    str    r2,[r0,#0x00]
#endif
    ldr    r2,=warm_reset
    str    r2,[r0,#0x20]
    // Relocate [copy] data from ROM to RAM
    ldr    r3,.__rom_data_start
    ldr    r4,.__ram_data_start
    ldr    r5,.__ram_data_end
    cmp    r4,r5            // jump if no data to move
    beq    2f
    sub    r3,r3,#4        // loop adjustments
    sub    r4,r4,#4
1:    ldr    r0,[r3,#4]!    // copy info
    str    r0,[r4,#4]!
    cmp    r4,r5
    bne    1b
2:
#endif
```

Figure 6 Modified vectors.S

This final change can be verified by simply executing the “reset” command from the RedBoot prompt. With this change in place, RedBoot should execute a warm reset and restart itself, instead of performing a board reset and restarting the external boot or POST code.

Contact Information:

Kozio, Inc.
Longmont, Colorado
Phone: (303) 776-1356
sales@kozio.com
www.kozio.com

kDiagnostics™, kManufacturing™, kPOST™ and Flash-N-Run™ are registered trademarks of Kozio, all other trademarks are property of their respective owners.