nohau
CORPORATION

EMUL-ARM

Board Support: Atmel EB40

February 25, 2004

Contents

About This Guide

EMUL–ARM is a PC-based hardware debugger for the ARM™ Core (currently ARM7 and ARM9 cores). Seehau is the name of the user interface of EMUL-ARM – Seehau and EMUL-ARM is often used interchangeably.

This User's Guide helps you to understand how to use the Atmel EB40 evaluation board with:

- EMUL-ARM
- HI-TECH ARM-C and HI-TIDE
- IAR EWARM C/C++

Note that there might exist two versions of this document (and other BSP/Target Board Related documents):

- If it is located in the "SeehauARM\Documents" install directory then it relates to currently installed Seehau.
- If it is located in the "C:\Nohau\BSP_ARM\Doc" directory then it relates to the currently installed BSP.

Please note that it is required to install BSPs to "C:\Nohau\BSP_ARM" because compiler project files are often dependent on file location.

EB40 is equipped with AT91R40807, but EB40 is also the evaluation board for following devices:

- AT91FR4081
- AT91M40800
- AT91M40807
- AT91F40816

1 INTRODUCTION

Important Notes

The target board comes with a manual – please read it. This document is intended to complement the board documentation – not replace it.

Atmel has some FAQs on their web site with information – currently at:

- http://www.atmel.com/dyn/products/app_notes.asp?family_id=605&part_id=1980

It is assumed and required that the BSP is installed to c:\Nohau\BSP_ARM.

Memory Configuration – EB40 (AT91R40807)

Internal memory:

- SRAM: 8 KB. Address: 30_0000 - 30_1FFF **Before remap**
- SRAM: 8 KB. Address: 0 - 1FFF **After remap, EBI_RCR register**
- SRAM: 128 KB. Address: 10_0000 – 11_FFFF

External memory:

- Flash: 2 MB. Address: 0 – 1F_FFFF **Before remap**
- Flash: 2 MB. Address: 100_0000 – 11F_FFFF **After remap, chip selects**

Chip selects to remap external Flash:

- EBI_CSR1 = 01002535
- EBI_CSR2 = 02002121

Remap the internal RAM

The ARM vectors (Reset, Interrupt etc) are mapped from address 0x0 to address 0x20. In order to allow these vectors to be redefined dynamically by the software, the AT91X40 Series Microcontrollers use a remap command that enables switching between the boot memory and the internal primary SRAM bank addresses. The remap command is accessible through the EBI User Interface, by writing one in RCB of EBI_RCR (Remap Control Register). Performing a remap command is mandatory if access to the other external devices (connected to chip selects 1 to 7) is required. The remap operation can only be changed back by an internal reset or an NRST assertion.

Seehau supports this remapping (enabled by default). Change by menu:

- Config | Emulator – then use the Misc Tab.

Flash Programming with EMUL-ARM

Seehau has built in support for flash programming for Atmel EB40. However it is disabled by default. When flash programming is enabled, normal "load" causes flash to erased and written as needed. To enable, use menu:

- Config | Emulator – then use the Misc Tab, and uncheck "Flash Written as RAM".

Seehau uses a "target definition file" to define which the flash memory is, and where it is located. It is automatically setup if you configured for Atmel EB40 during the initial configuration. It can be configured later by selecting a file on the Misc tab:

- Atmel_EB40.tdf – assumes flash at address 0x1000000.
- Atmel_EB40_0.tdf – assumes flash at address 0.

Flash programming is disabled by default, enable by using menu "Config | Emulator" – Misc Tab:

- Un-check "Flash Written as RAM".

Also, "Monitor Load" is required, enable by using menu "Config | Emulator" – Misc Tab:

- Check "Monitor Load".

Memory Configuration – AT91M40807

Internal memory:

- SRAM: 8 KB. Address: 30_0000 - 30_1FFF **Before remap**
- SRAM: 8 KB. Address: 0 - 1FFF **After remap**

Memory Configuration – AT91FR4081

Internal memory:

- SRAM: 8 KB. Address: 30_0000 - 30_1FFF **Before remap**
- SRAM: 8 KB. Address: 0 – 1FFF **After remap, EBI_RCR register**
- Flash: 1 MB Address: 0 – F_FFFF **Before remap**
- Flash: 1 MB Address: 100_0000 – 10F_FFFF **After remap, chip selects**

Memory Configuration – AT91F40816

Internal memory:

- SRAM: 8 KB. Address: 30_0000 - 30_1FFF **Before remap**
- SRAM: 8 KB. Address: 0 – 1FFF **After remap, EBI_RCR register**
- Flash: 2 MB Address: 0 – 1F_FFFF **Before remap**
- Flash: 2 MB Address: 100_0000 – 11F_FFFF **After remap, chip selects**

Memory Configuration – AT91M40800

Internal memory:

- SRAM: 8 KB. Address: 30_0000 - 30_1FFF **Before remap**
- SRAM: 8 KB. Address: 0 – 1FFF **After remap, EBI_RCR register**

SF_MMR / RAMWU on AT91FR4081 and AT91R40807

To allow writes to the internal RAM on the AT91FR4081 and AT91R40807, you need to set the RAMWU bit in the SF_MMR register.

This is important, because otherwise an exception will occur when you try to write to the internal 128 KB SRAM.

AIC Protect Mode

The AIC (Advanced Interrupt Controller) has a Protect Mode – from the manual:

- The Protect Mode permits reading of the Interrupt Vector Register without performing the associated automatic operations. This is necessary when working with a debug system.

Please note that the debugger does nothing to enable Protect Mode automatically. See the MCU Manual.

Board Configuration

- U10 – in MAST position.
- Upper Mem.

Other Documents

Please also see following documents:

- **ARM_BSP.pdf** – overview of general source code that is supplied with EMUL-ARM for Atmel EB40 – see ARM_BSP.pdf. This document discusses Timer, different Target Console applications and uC/OS-II.
- **ARM_Compilers.pdf** – how to set up different compilers to be used with EMUL-ARM.
- **Nohau_Monitor .pdf –** using the Nohau Monitor to get printf() from target application in Seehau etc.

Target Board CD

The Atmel EB40 board comes with a CD with documentation and schematics.

2 DEVELOPING SOFTWARE

Include Files

The compiler will generally have #include files for MCU registers. If yours doesn't have it, you can use following file (where the initial N is used to denote Nohau and avoiding name conflicts):

- c:\Nohau\BSP_ARM\Src\Inc\ **NAT91X40.h.h**

All examples in Nohau BSP ARM are based on a couple of .h files. These are:

- **NConfig.h** (located in the application directory)
- c:\Nohau\BSP_ARM\Src\Inc**NDefs.h**
- c:\Nohau\BSP_ARM\Src\Inc**NArm.h** (there is an NArm.c file aswell)

The Nohau Monitor uses an additional .h file in the application directory, and so does uC/OS-II:

- **NMon_Cfg.h** – Nohau Monitor.

Startup Code

Startup code for EB40 is located in:

- c:\Nohau\BSP_ARM\Boards\Atmel\EB40\Startup:

IAR

For IAR C/C++, there are two files with startup code in directory

- **cstartup_IAR.s –** assumes program memory (flash or ram) at address zero.
- **cstartup_Remap_IAR.s –** assumes that the application shall execute at address 0x1000000, but start at 0x0 after reset, for which reason it performs the remap.

HI-TECH

For HI-TECH ARM-C, there are three files with startup code in directory

- **stacks_HTC.c** – sets up stacks for selected modes.
- **vectors_HTC.as** – sets up the interrupt vectors.
- **powerup_Remap_HTC.as** – startup code that sets up memory, copies vectors and performs remap.

Memory Map / Linking

HI-TECH ARM-C does not use link control files. Memory ranges are defined within the HI-TIDE project.

For the IAR compiler, link control files are provided in directory:

- c:\Nohau\BSP_ARM\Boards\Atmel\EB40\Link\

Following files are available:

- **RAM_IAR.xcl** – code and data in RAM at address 0x0 + 10_0000;
- **FLASH_IAR.xcl** – code in Flash at address 0x0, RAM at address 0x30000.
- **FLASH_REMAP_IAR.xcl** – code in Flash at address 0x1000000, RAM at address 0x0.

3 EXAMPLE APPLICATIONS

General Examples

These are applications that are shared between most supported boards. See ARM_BSP.pdf for additional information.

The application categories are available:

- **TargetConsole** - Shows the capabilities of the EMUL-ARM debugger-target communication.
- **Interrupt** – Shows how to do basic interrupts with and without the debugger target-communication.
- **uCOSII** – Shows the uC/OS-II RTOS. The EB40 does not have the “Full” examples, because the RAM_IAR.xcl file does not define enough RAM for this.

Notice the Interrupt/TimerFlash example, which shows how to create a program that “remaps“ the hardware.