NOHAU brand by ICE Technology

EMUL–ARM™

Flash Programming and Related Topics

© 2003 All rights reserved worldwide.

Contents

1 INTRODUCTION....4

Overview 4

Configuration 4

Limitations 5

Memory Mapping / Chip Selects 5

Reset Values in Seehau.... 5

Register Definitions in the Target Definition.... 5

Hardware/Software Breakpoints 6

2 SPECIFYING FLASH AND MEMORY REGIONS....7

Target Definition 7

Built in Flash Support (Flash List) 8

Unsupported Flash Devices 8

3 DEVELOPING NEW SEEHAU FLASH ALGORITHMS.... 9

Overview 9

Save Algorithms as Intel Hex 10

Testing the Algorithms 11

LoadGen Utility 11

4 REFERENCE....12

XML Files 12

General.... 12

MemChip XML Tags.... 13

MemRegion XML Tags.... 13

Register XML Tags.... 15

About This Guide

The 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 guide helps you to understand how to work with EMUL-ARM when the program memory resides in Flash, or partially in Flash.

!!! Please note that "Flash programming" is currently in a "beta state" !!!

!!! The specification for Flash Programming is to support up to 5 flash chips (same or different). However, it is currently only tested with 1 chip. !!!

!!! Currently, chip erase will probably cause a timeout error since it is not yet tested. !!!

1 INTRODUCTION

Overview

EMUL-ARM supports loading and executing programs (and data) into Flash. Conceptually, this is handled as any normal load (into RAM). However, there are some special considerations - fore-most – how define the target and using breakpoints. This and other related topics are defined in this document.

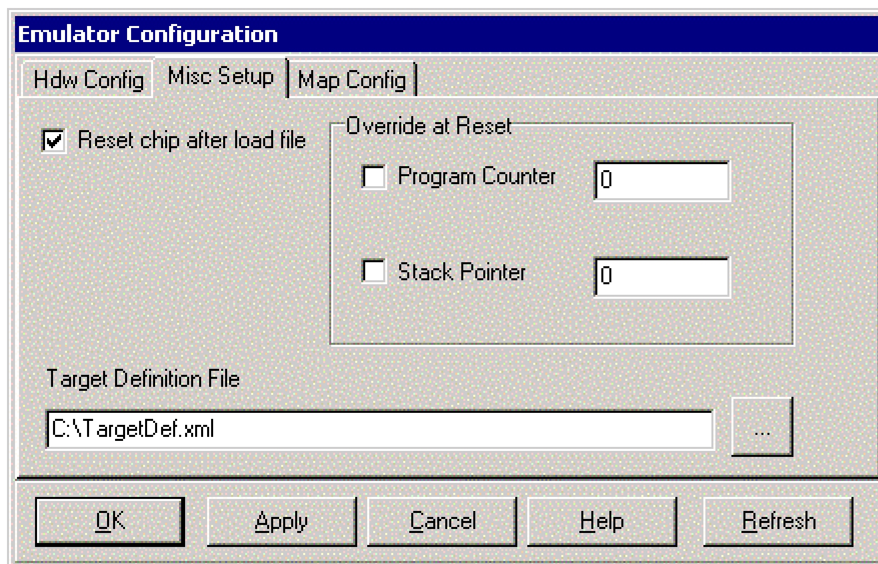
Note that it is currently not possible to write flash from a data window, and it is not possible to erase flash from the GUI. Flash programming is currently only handled when loading files.

Two erase mechanisms can be used – sector erase or chip erase. Built in support uses sector erase (unless chip erase is specified). When sector erase is used, only sectors that are being written to are erased. This means that it is possible to have two load files without erase/write conflicts as long as they do not share any sectors.

Configuration

The Target Definition file defines how flash programming is performed. Currently, there is no way to modify the target definition file through the Seehau user interface. The file format and details are described later in this document.

The Target Definition file name is configured through “Config | Emul” as shown in the picture.



EMUL-ARM uses two kinds of breakpoints – software and hardware breakpoints, see discussion below. Software breakpoints is the default, but when source and assembly stepping “in Flash”, hardware breakpoints must be used. To change to use hardware breakpoints, use menu:

- Run | Force Hardware Step.

Limitations

- Program memory in Flash is currently supported in Little Endian only.
- Chip lockout / security modes are not supported.
- Currently supported on MCUs with internal RAM (minimum 4KB).

Memory Mapping / Chip Selects

Writing to Flash (or RAM for that matter) often requires chip selects to be set up. This is the case for most MCUs when using external flash. With internal flash, it is common that the MCU is already setup to use the internal flash. There are two ways to deal with chip selects:

- “Reset values” for SFR registers defined from within Seehau – recommended while testing.
- Register definition in the target definition – recommended when imp

Reset Values in Seehau

The easiest way to enable reset values is to:

- Add SFRs that control the chip selects – right click and select: “Add Special Register (SFR)…”. Then add all the registers that controls chip selects, memory mapping etc to a new register window (by using Add To.../New Window), and close the dialog.
- Finally, for all relevant memory mapping registers, select the register, right click and select “Change Attributes…”. In the popup, check “Enable Reset Value” and set the desired value.

This will cause the “Reset Values” to be written to the SFRs every time reset is pressed.

Register Definitions in the Target Definition

The target definition file needs something like this:

```
<RegisterList>
   <Register Name="EBI_CSR0" Value="0x01002539" Addr="0xFFE00000" Size="32"/>
</RegisterList>
```

Hardware/Software Breakpoints

When executing in Flash, there are limitations on how many breakpoints that can be used. The ARM core only allows two “hardware” breakpoints (see description below), so that is all that is possible with EMUL-ARM. Note that one of the two breakpoints are used to implement source and assembly stepping, which leaves only one breakpoint during stepping. This is not a limitation in EMUL-ARM; it is a limitation of the ARM core itself.

To explain a little bit about breakpoints - there are two basic methods to implement a breakpoint:

1. Hardware – hardware resources are used to implement the breakpoint.

2. Software – a special instruction is written to the address where the break is desired. When a software breakpoint is executed, they cause the MCU to break.

This means that when using HW breaks, there can only be as many breakpoints as supported by the HW, in the case of ARM7 and ARM9 there are only two hardware breakpoints.

In order to use unlimited number of breakpoints, they have to be implemented using software breakpoints. EMUL-ARM does currently not support software breakpoints in Flash. (It is possible, but involves erasing a whole Flash block and writing data back every time a breakpoint is set and again when cleared.)

Note – in ARM7, software breakpoints are implemented using HW breakpoint resources. This means that one of the two HW breaks are used to implement unlimited amount of SW breaks in the same mode. If SW breaks are used in both ARM and Thumb modes, both HW breaks are used.

2 SPECIFYING FLASH AND MEMORY REGIONS

Target Definition

The target definition controls how flash programming is performed. Currently, the target definition is stored in an xml file, which needs to be modified manually. For many MCUs with internal flash, the target definition is pre-defined, but can be overridden (only necessary when using external flash).

A target definition file defaults to the extension “TDF” and an MCU file defaults to extension “MCU”.

Below is an example of what the target definition (xml) file looks like. There are three main parts of information:

- Memory Chips – a list of user defined flash chips – optional.
- Memory Regions – a list of memory regions with attributes. For flash regions, these need to be mapped to a chip. This allows multiple regions to use one chip.
- Registers – a list of register values to apply after each reset. This is intended to allow chip selects to be defined in the target definition.

```
<?xml version="1.0"?>
<Nohau Type="TargDef" Format="1">
  <Info>
     <RegisterList>
        <Register Name="EBI_CSR0" Value="0x01002539" Addr="0xFFE00000" Width="4"/>
     </RegisterList>
     <MemChipList>
        <MemChip Vendor="Avendor" Name="TestA" Size="0x200000" Type="FLASH"
                 File="FL_AT49.hex" ChipErase="def" Write="def"/>
        <MemChip Vendor="Bvendor" Name="TestB" Size="0x200000" Type="FLASH"
                 File="FL_AT49.hex" ChipErase="def" Write="def"/>
     </MemChipList>
     <MemRegionList>
        <MemRegion Type="RAM"   Access="RW" Chip="" Addr="0x8000"  Size="0x20000"/>
        <MemRegion Type="RAM"   Access="R"  Chip="" Addr="0x8000"  Size="0x20000"/>
        <MemRegion Type="RAM"   Access="W"  Chip="" Addr="0x8000"  Size="0x20000"/>
        <MemRegion Type="FLASH"             Chip="AT49BV1614" Addr="0x8000" />
        <MemRegion Type="FLASH"             Chip="AT49BV1614" Addr="0x20000"/>
     </MemRegionList>
  </Info>
</Nohau>
```

For full definition of the target definition format, see the Reference section. Note that a “MemChip” tag cannot refer to a “MemChip” in another file with the “Uses” attribute. However, the same effect can be achieved by using a “MemRegion” tag – just let the “MemRegion” use a corresponding chip.

Note – if a MemChip is defined in both “FlashList.xml” and in the target definition, the chip in the target definition will be ignored.

Built in Flash Support (Flash List)

The built in Flash Support defines algorithms for a number of Flash devices; this information is stored in a file named “FlashList.xml”. The file needs to be located in the “Logic” directory (i.e. C:\Nohau\SeehauARM\Logic for a default installation). The list of supported devices will grow over time.

The format of “FlashList.xml” is identical to that of the target definition, except it only contains the “Memory Chips” definitions. Below is an example of what it the “FlashList.xml” looks like:

Note the “Uses” element, which basically says that it is the same chip, but also allows overriding some properties, such as size. For full definition of the target definition format, see the Reference section.

```
<?xml version="1.0"?>
<Nohau Type="MemChipDef" Format="1">
  <Info>
     <MemChipList>
        <MemChip Vendor="Atmel" Name="AT49BV1614" Size="0x200000" Width="2"
                 Type="FLASH" File="FL_AT49.hex" ChipErase="def" Write="def"/>
        <MemChip Vendor="Atmel" Name="AT49BV1614T"  Uses="AT49BV1614"/>
        <MemChip Vendor="Atmel" Name="AT49BV1604"   Uses="AT49BV1614"/>
        <MemChip Vendor="Atmel" Name="AT49BV1604T"  Uses="AT49BV1614"/>
        <MemChip Vendor="Atmel" Name="AT49BV1604A"  Uses="AT49BV1614"/>
        <MemChip Vendor="Atmel" Name="AT49BV1604AT" Uses="AT49BV1614"/>
        <MemChip Vendor="Atmel" Name="AT49BV1614A"  Uses="AT49BV1614"/>
        <MemChip Vendor="Atmel" Name="AT49BV1614AT" Uses="AT49BV1614"/>
        <MemChip Vendor="Atmel" Name="AT49LV1614A"  Uses="AT49BV1614"/>
        <MemChip Vendor="Atmel" Name="AT49LV1614AT" Uses="AT49BV1614"/>
     </MemChipList>
  </Info>
</Nohau>
```

Unsupported Flash Devices

EMUL-ARM dos not support all Flash devices available, and it never will. However, it is possible for end users to develop support on their own. We encourage this, and will assist in getting your flash device to work if we in return can add it to our list of supported devices, in this case we may ask you to test a new “FlashList.xml” file.

3 DEVELOPING NEW SEEHAU FLASH ALGORITHMS

Overview

Before starting to develop algorithms for a new flash device, try to see if any of the already supported devices uses the same algorithm. Several of the options defined for a Memory Chip can be re-defined in a Memory Region – this includes Size and Sectors. So if the portioning of sectors and size is the only difference, just set their values for a memory region. It may not be necessary to develop new flash algorithms at all.

The Seehau Flash Algorithms consist of two (or three) functions. These should be standard "c" functions (ARM mode). These are called like any "c" function, which means that they need to use the link register when returning (R14). The input parameters to the functions are stored in R0..R3.

Note that the instructions in the Flash Algorithms needs to be "position independent" since they will not be loaded to the same address to which they where they originally were linked. It is a good idea to avoid function pointers and global variables.

The best way to develop a new flash driver is to start with the Flash examples located in the Seehau "Examples\EMUL-ARM\Flash" directory. There are two subdirectories:

- **Develop** – Use this source code as basis for developing driver for new chips.
- **AlgorithmsTest** – test the algorithms after they are saved to an Intel hex file.

So, first create a small application based on the "Flash\Develop" example. It should be able to erase and write a buffer to the flash. Following interface is used for the functions called by Seehau – see the files "NFlash.h" and "NFlash_AT49.c" for additional information:

```
void WriteBuf   (ADDRESS base, ADDRESS write, BYTE* pFrom, BYTE* pLast);
void EraseSector(ADDRESS base, ADDRESS sector);
void EraseChip  (ADDRESS base);
```

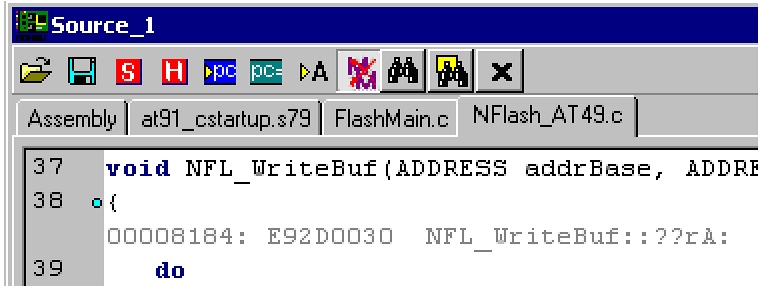
Note that this example is developed with the IAR Embedded Workbench and that the project files should be possible to use directly with IAR EWB version 3.3 (and higher).

IMPORTANT! The algorithms cannot access global data – only data defined in the function is possible. Also, avoid the keyword "static" which will make it more difficult to analyze problems.

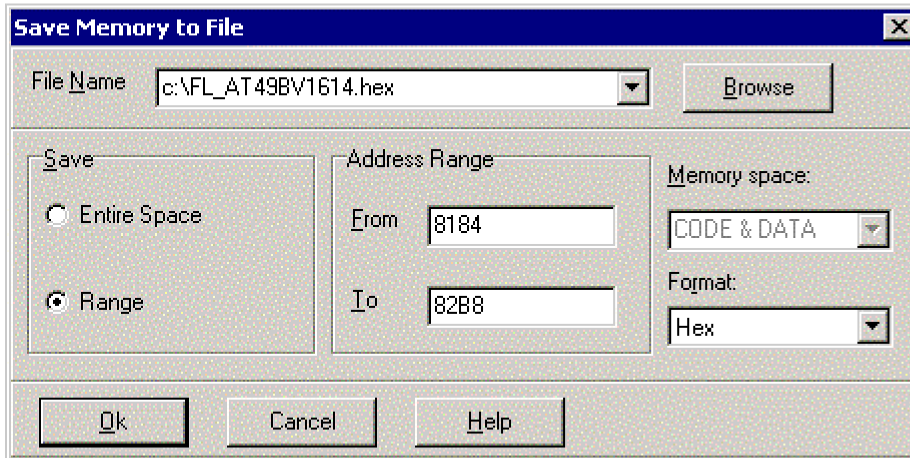
Save Algorithms as Intel Hex

Seehau loads flash algorithms from a file on Intel Hex format as needed. There are no flash algorithms that are hard coded into Seehau.

When the functions are implemented and tested, go into mixed mode (press the M/M button as shown below) and find the respective algorithm/function start addresses and the memory range they occupy. In the picture below, WriteBuf() starts at 0x8184. We are going to save the whole memory range to a file, so it makes sense to put the two functions together in the source.



The instructions can be saved using menu "Source | File | Save Assembly as Hex…" (available when a source window is active). Following popup is shown – enter text shown. Note that the only supported format is Intel Hex.



This will give an (partial) entry like following in the Flash List / Target Definition:

```
<MemChip ... File="FL_AT4949BV1614.hex" EraseChipAlgo="0x8220"
Write="0x8184" ... />
```

Testing the Algorithms

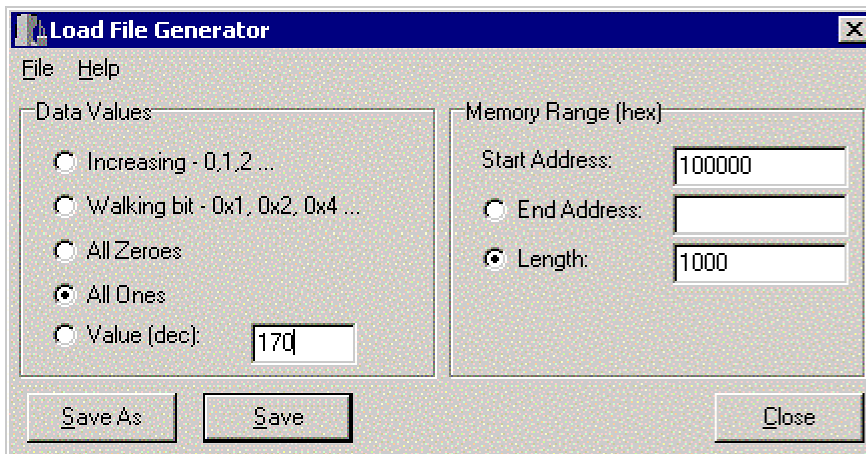
So we have developed algorithms and saved them to file. Now we want to test that what was saved to the Intel Hex file actually works as intended. For this purpose, use the AlgorithmsTest example, which will call the algorithms using function pointers. You will have to modify the addresses they are the pointers point to – the write address should be 0x8184 for the example above. The values of the function pointers are located in the file “NFlash_Pfn.c”.

When executing the AlgorithmTest, study the results of each call by looking into a data window.

When this works, you may also want to make sure that the algorithms are can be moved in memory and that they are still working. You can do this by right clicking in the data window and selecting “Block | Block Move”. Remember to change the function pointers accordingly.

LoadGen Utility

The LoadGen.exe utility (Load File Generator) can be found in the Seehau install directory. This utility allows creating Load Files on Intel Hex format for testing Flash drivers (and more). A number of data patterns for specified address ranges can be generated, as shown in the figure.



4 REFERENCE

XML Files

General

The XML files read by EMUL-ARM must be “well formed” – i.e. follow the XML syntax. If not a warning or error message will be issued, and flash programming will not be available. Also, XML comments (<!-- comment -->) are not supported.

The root tag must be named “Nohau” and have a “Type” and a “Format” attribute. Next level must be the “Info” tag as shown below. Naturally, the top level <?xml> is required.

```
<?xml version="1.0"?>
<Nohau Type="MemChipDef" Format="1">
  <Info>
```

The document “Type” can be one of:

- **TargDef** – used for Target Definitions.
- **MemChipDef** – used for FlashList.xml.

MemChip XML Tags

- **Vendor** – the name of the vendor. Currently used for reference only, but will be used for GUI based configuration in the future.
- **Name** – chip name, i.e. manufacturer part number. This is used to match with a "Memory Region", which specifies it as "Chip".
- **Size** – chip size in number of bytes.
- **Width** – width of one write operation, i.e. 1, 2 or 4 bytes.
- **Type** – memory type. Possible values are: "FLASH" and "RAM".
- **File** – File with algorithms for erasing and writing (and possibly more in the future).
- **EraseValue (optional)** – bit values after erase is performed. Possible values 0 and 1 (default).
- **EraseChipAlgo** – Address of chip erase algorithm within "File".
- **EraseSectorAlgo** – Address of sector erase algorithm within "File".
- **WriteAlgo** – Address of write algorithm within "File".
- **Access (optional)** – Memory access type. Possible values are "R"=Read Only, "W"=Write Only, "RW"=Read and Write.
- **Sectors (optional)** – Start address of all sectors in the chip in a comma separated list, or list of count and size – i.e. "2*0x1000,4*0x80000".
- **Uses (optional)** – Specifies that a chip use a definition for another chip (many chips use same algorithms.) Note that the MemChip referred to with "Uses" must be located in the same file.

One if "EraseChipAlgo" and "EraseSectorAlgo" must be defined. If both are defined, sector erase will be used.

MemRegion XML Tags

- **Type** – same as above.
- **Access** – same as above.
- **Chip** – chip name – "Name" in MemChip definition.
- **Addr** – base (start) address for the memory chip.
- **Size** – same as above – can be used to override a definition in the chip.
- **Sectors** – same as above – can be used to override a definition in the chip.

Register XML Tags

- **Name** – register name – should match that of the SFR definition in Seehau.
- **Value** – reset value.
- **Addr** – register address.
- **Width** – width of one write operation, i.e. 1, 2 or 4 bytes.