



**STR71x MEMORY MAPPING
AND DEVELOPMENT TOOLSET**

INTRODUCTION

A major consideration in the design of STR71 applications is the layout of the memory map.

This note describes how to define the memory map of your STR71x application. The first part of this document shows how to use scatter loading files to place and organize data and code in memory. The second part focuses on software memory remapping.

1 USING SCATTER FILES FOR IMAGE ORGANIZATION IN MEMORY

This paragraph describes how to place and organize the data and code in memory. It focuses on the use of the scatter-loading file for the RealView toolset.

An image is made up of regions and output sections. Every region in the image can have a different load and execution address. The scatter-loading mechanism enables you to specify the memory map of an image to armlink. For more information on the scatter file, load and execution regions, please refer to 'ARM Linker and Utilities Guide'.

Following reset, the processor starts to fetch instructions from the reset vector at the address 0x00000000. This instruction should transfer control to the initialization code. The entry point must lie within an execution region that must be a root region. For this reason, the vector and initialization routines must always be present at the address 0x00000000.

Different types of memory are present in our system so different mappings can be used depending on the boot mechanism. For further information on boot modes, refer to the STR71 datasheet.

The following table explains the use of the different scatter loading files.

Table 1. Scatter file use

Scatter File	Boot mode
71x_armr	Bootting from RAM
71x_arne	Bootting from External Memory
71x_armf	Bootting from Embedded Flash

1.1 BOOTING FROM RAM

On power-up the RAM is located at address 0x00000000. The first section is a root region made up of the vector and initialization routines.

The contents and the organization of the scatter loading description file are shown below.

```
RAM_MODE 0x20000000 0x10000 ; Load region starts at the real RAM address
{
  RAM 0x20000000 ; Startup exec region address is the same as the
  {
    ; load address (root region)
    71x_vect.o (Vect, +First); - Section containing the entry point
    71x_init.o (Init) ; - Section containing the init routines
    * (+RO) ; - The code and the read only data are placed
    ; after the init routine
    * (+RW) ; - The read/write data (variables)
    * (+ZI) ; - The zero initialized region is placed after
  } ; the read/write data
}
```

1.2 BOOTING FROM EXTERNAL MEMORY

On power-up the external memory bank 0 is aliased at address 0x00000000. The contents and organization of the scatter loading description file are shown below.

```
EXTMEM 0x60000000 0x400000; Load region starts at the real EMI address
{
  EXTMEM 0x60000000          ; Startup exec region address is the same as the
  {                          ; load address (root region)
    71x_vect.o (Vect, +First); - Section containing the entry point
    71x_init.o (Init)        ; - Section containing the init routines
    * (+RO)                  ; - The code and the read only data are placed
  }                          ; after the init routine
  RAM 0x20000000             ; A second exec region placed in the internal RAM that
  {                          ; contains the Read/Write data and the zero
    * (+RW)                  ; initialized region.
    * (+ZI)
  }
}
```

or

```
EXTMEM 0x60000000 0x400000 ; Load region starts at the real EMI address
{
  EXTMEM 0x60000000          ; Startup exec region address is the same as the
  {                          ; load address (root region)
    71x_vect.o (Vect, +First); - Section containing the entry point
    71x_init.o (Init)        ; - Section containing the init routines
    * (+RO)                  ; - The code and the read only data are placed
  }                          ; after the init routine
  RAM 0x62000000             ; A second exec region placed in the external
  {                          ; SRAM that contains the Read/Write data and
    * (+RW)                  ; the zero initialized region.
    * (+ZI)
  }
}
```

1.3 BOOTING FROM EMBEDDED FLASH

On power-up the flash is aliased at address 0x00000000. The contents and organization of the scatter loading description file are shown below.

```
FLASH 0x40000000 0x40000    ; Load region starts at the real FLASH address
{
  FLASH 0x40000000          ; Startup exec region address is the same as the
  {                          ; load address (root region)
    71x_vect.o (Vect, +First); - Section containing the entry point
    71x_init.o (Init)        ; - Section containing the init routines
    * (+RO)                  ; - The code and the read only data are placed
  }                          ; after the init routine
  RAM 0x20000000             ; A second exec region placed in the internal RAM that
  {                          ; contains the Read/Write data and the zero
    * (+RW)                  ; initialized region.
    * (+ZI)
  }
}
```

or

```
FLASH 0x40000000 0x40000    ; Load region starts at the real FLASH address
{
  EXTMEM 0x40000000          ; Startup exec region address is the same as the
  {                          ; load address (root region)
    71x_vect.o (Vect, +First); - Section containing the entry point
  }
```

```
71x_init.o (Init)      ; - Section containing the init routines
* (+RO)                ; - The code and the read only data are placed
}                      ; after the init routine
RAM 0x62000000         ; A second exec region placed in the external
{                      ; SRAM that contains the Read/Write data and
* (+RW)                ; the zero initialized region.
* (+ZI)
}
}
```

2 CODE INITIALIZATION

Before entering the C code, the RW region must be copied from the load region to the execution region and the ZI region must be created.

The initialization code is integrated in the startup routine "71x_init.s".

The following software is a typical initialization code that corresponds to the last examples. It uses the symbols generated by the linker to find information about each region.

```

IMPORT IImage$$RAM$$BaseI      ; Base of RAM to initialise
IMPORT ILoad$$RAM$$BaseI      ; End of ROM code (=start of ROM data)
IMPORT IImage$$RAM$$ZI$$BaseI  ; Base of area to zero initialise
IMPORT IImage$$RAM$$ZI$$LengthI ; Length of area to zero initialise

LDR    r0, =ILoad$$RAM$$BaseI   ; Get pointer to ROM data
LDR    r1, =IImage$$RAM$$BaseI  ; and RAM copy
LDR    r3, =IImage$$RAM$$ZI$$BaseI ; Zero init base => top of initialised data
CMP    r0, r1                   ; Check that they are different
BEQ    %F1

0  CMP    r1, r3                 ; Copy init data
   LDRCC  r2, [r0], #4
   STRCC  r2, [r1], #4
   BCC    %B0

1  LDR    r1, =IImage$$RAM$$ZI$$LengthI ; Length of zero init segment
   ADD    r1, r1, r3                 ; Top of zero init segment
   MOV    r2, #0

2  CMP    r3, r1                 ; Zero init
   STRCC  r2, [r3], #4
   BCC    %B2

```

If the execution region is mapped to the external memory, before the code initialization, the EMI must be enabled and configured. The following assembly code enables and configures the EIM bank 1 with 7 wait states and 16 bit wide.

```

LDR    r0, =GPIO2_Base_addr      ; Configure P2.0 -> 3 in AF_PP mode
LDR    r1, =0x0000000F
STR    r1, [r0, #PC0_off_addr]
STR    r1, [r0, #PC1_off_addr]
STR    r1, [r0, #PC2_off_addr]

LDR    r0, =EMI_Base_addr
LDR    r1, =0x0000803D
STR    r1, [r0, #BCON1_off_addr] ; Enable bank 1 16-bit 7 wait state

```

3 SOFTWARE REMAPPING

On power-up, an aliased copy of either ROM or RAM is located at address 0x0. This depends on the boot mode and the user application. But in some applications it may be necessary to change the memory map after reset.

This is because, having ROM starting at address 0x0 slows down the handling of processor exceptions through the exception table. Also, the vector table cannot be overwritten if it is on ROM. On the other hand, if RAM is mapped to address 0x0 after reset, there will be no valid instruction in the reset vector, unless you use debug mode.

You can get around this issue by resorting to memory remapping. After remapping, the vector table must be copied to memory address 0x0.

Remapping is done by changing the Boot[1:0] bits in the boot configuration register (BOOT-CONF). This code is executed by the ARM7TDMI in SVC mode.

The following example shows the code for remapping the RAM to 0x0 and copying the vector table to memory address 0x0.

```
IF :DEF: remap_flash
    MOV r0, #FLASH_mask
    LDR pc, =next
ENDIF

IF :DEF: remap_ram
    ; Copying the vector table into RAM
    LDR r0, =vector_begin    ; r0 is start address from which to copy
    LDR r2, =0xffffffff
    LDR r3, =vector_end
    AND r3, r2, r3          ; r3 is number of byte to be copy
    LDR r1, =RAM_base       ; r1 is start address where to copy
copy
    LDR r2, [r0], #4        ; Read a word from the source
    STR r2, [r1], #4        ; Copy the word to destination
    SUBS r3, r3, #4         ; Decrement number of word to copy
    BNE copy               ; Branch to copy if r5 not equal to 0
    MOV r0, #RAM_mask
    LDR pc, =next
ENDIF

IF :DEF: remap_ext
    MOV r0, #EXTMEM_mask
    LDR pc, =next
ENDIF

next
    LDR r1, =CPM_base_addr    ; r0= CPM base address
    LDRH r2, [r1, #BOOTCONF_off_addr] ; Read BOOTCONF Register
    BIC r2, r2, #0x03         ; Reset the two lsb bits of BOOTCONF Reg.
    ORR r2, r2, r0           ; Change the two LSB bits of BOOTCONF Reg.
    STRH r2, [r1, #BOOTCONF_off_addr] ; Write BOOTCONF Register
```

“THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.”

Information furnished is believed to be accurate and reliable. However, STMicroelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of STMicroelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of STMicroelectronics.

The ST logo is a registered trademark of STMicroelectronics.

All other names are the property of their respective owners

© 2004 STMicroelectronics - All rights reserved

STMicroelectronics GROUP OF COMPANIES

Australia – Belgium - Brazil - Canada - China – Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States

www.st.com