Freescale Semiconductor
Application Note

Document Number: AN4188

RS08 Upper Memory Access

1. Introduction

The purpose of this document is to provide the RS08 programmer with the information necessary for performing correct access to data placed in upper memory, that is, beyond the first, directly addressable, 256 bytes.

Contents

2. RS08 Memory Map

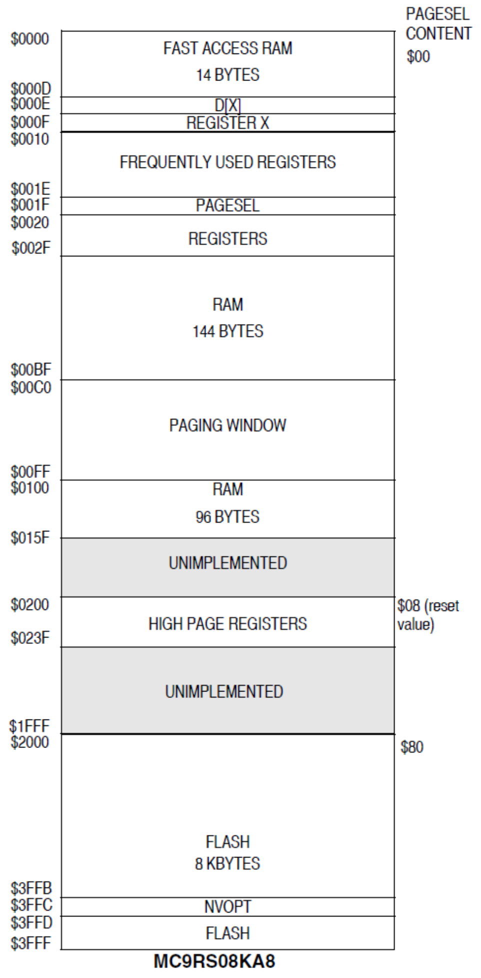
Figure 1 displays the memory map for MC9RS08KA8, a typical device of the RS08 family.

The memory map of the MCU is divided into the following groups:

- Fast access RAM using tiny and short instructions ($0000 – $000D)
- Indirect data access D[X] ($000E)
- Index register X for D[X] ($000F)
- Frequently used peripheral registers ($0010 – $001E, $0020 – $002F)
- PAGESEL register ($001F)
- RAM ($0030 – $00BF, $0100 – $015F)
- Paging window ($00C0 – $00FF)

- Other peripheral registers ($0200 – $023F)
- Non-volatile memory ($2000 – $3FFF)

Figure 1. MC9RS08KA8 Memory Map



3. RS08 Paging Scheme

The RS08 core does not inherently support memory access for data access beyond the first 256 bytes. Therefore, a paging scheme has been implemented, that segments the full 16-Kbyte address map of the RS08 core into 256 pages of 64 bytes each. The $0000–$00FF address range is mapped into the first four 64-byte pages.

In order to access data in a certain page, the paging window, which is located at $00C0–$00FF, must be positioned at the desired address range within the 16-Kbyte address space. This can be achieved by writing the page number into the page selection register (PAGESEL), which is located at $001F. As soon as the PAGESEL register has been updated to the appropriate value, subsequent accesses to the paging window will address the area determined by the register content.

Table 1 provides the range of pages that can be accessed through the paging window ($00C0 – $00FF) on

MC9RS08KA8.

Table 1. MC9RS08KA8 Paging Window

Page	Memory Address
$00	$0000–$003F
$01	$0040–$007F
$02	$0080–$00BF
$03	$00C0–$00FF
$04	$0100–$013F
. . .	. . .
$FE	$3F80–$3FBF
$FF	$3FC0–$3FFF

4. Paged Access to Data

In order to exploit the RS08 paging scheme, the compiler supports paged addressing: data is accessed via 16-bits addresses, with the page number in the high byte, and the paging window offset in the low byte.

4.1. Direct Access

If a variable is being accessed directly, rather than through a pointer, the compiler will generate a paged access if either one of the following conditions is fulfilled:

- Object has been explicitly placed in a paged segment (using a DATA_SEG / CONST_SEG pragma with the __PAGED_SEG modifier)
- Compiler has been set up for the BANKED memory model (all data accessed as paged by default).

Listing 1 provides an example.

Listing 1: Paged access to variable my_data, of type int (BANKED memory model)

```
Source code:
#pragma push
#pragma DATA_SEG __PAGED_SEG MY_RAM
int  my_data;
#pragma pop
void Test() {
    my_data = 0x1234;
}
Generated code:
  10:    my_data = 0x1234;
0000 3e001f          MOV        #%HIGH_6_13(my_data),PAGESEL
0003 3e1200          MOV        #18,%MAP_ADDR_6(my_data)
0006 3e3401          MOV        #52,%MAP_ADDR_6(my_data:1)
```

4.2. Pointer Access

If access to the variable is performed via pointers, the compiler will generate a paged access if either one of the following conditions applies:

- the pointer has been explicitly qualified as paged (using the __paged pointer qualifier)
- the compiler has been set up for the BANKED memory model (all data accessed as paged by default).

Listing 2 provides an example.

Listing 2: Paged pointer access to variable my_data, of type int (BANKED memory model)

```
Source code:
#pragma push
#pragma DATA_SEG __PAGED_SEG MY_RAM
int  my_data = 0x1234;
#pragma pop
int * __paged p = &my_data;
void Test() {
  volatile int v = *p;
}
Generated code:
   11:    volatile int v = *p;
0000 3e011f          MOV        #%HIGH_6_13(p:1),PAGESEL
0003 4e010f          LDX        %MAP_ADDR_6(p:1)
0006 4e001f          MOV        %MAP_ADDR_6(p),PAGESEL
0009 4e0e00          MOV        D[X],__OVL_Test_v
000c 2f              INCX
000d 4e0e01          MOV        D[X],__OVL_Test_v:1
```

5. Far Access to Data

If a data object is large enough to cross page boundaries, paged addressing no longer works, because with paged access, the page selection register is only written once, for the first byte of the object. This is why the RS08 compiler also supports far addressing: the format of the address is the same as for paged addressing, but the page register is updated before each byte access.

Far addressing is more expensive than paged addressing, and you should refrain from using it unless absolutely required.

5.1. Direct Access

If a variable is being accessed directly, rather than using a pointer, the compiler will generate a far access if the object has been explicitly placed in a far segment (using a DATA_SEG / CONST_SEG pragma with the __FAR_SEG modifier). The page selection register will be updated before each byte access.

Listing 3 provides an example.

Listing 3: Far access to variable my_data, of type int (BANKED memory model)

```
Source code:
#pragma push
#pragma DATA_SEG _FAR_SEG MY_RAM
int  my_data;
#pragma pop
void Test() {
  my_data = 0x1234;
}
Generated code:
10:     my_data = 0x1234;
0000 3e001f           MOV        #%HIGH_6_13(my_data),PAGESEL
0003 3e1200           MOV        #18,%MAP_ADDR_6(my_data)
0006 3e011f           MOV        #%HIGH_6_13(my_data:1),PAGESEL
0009 3e3401           MOV        #52,%MAP_ADDR_6(my_data:1)
```

5.2. Pointer Access

If a variable is being accessed via pointers, the compiler will generate a far access if the pointer has been explicitly qualified as far (using the __far pointer qualifier). The page selection register will be updated for each byte to be accessed at the pointed-to memory location.

Listing 4 provides an example.

Listing 4: Far pointer access to variable my_data, of type int (BANKED memory model)

```
Source code:
#pragma push
#pragma DATA_SEG __FAR_SEG MY_RAM
int my_data = 0x1234;
#pragma pop
int * __far p = &my_data;
void Test() {
  volatile int v = *p;
}
Generated code:
11:     volatile int v = *p;
0000 3e011f           MOV        #%HIGH_6_13(p:1),PAGESEL
0003 4e010f           LDX        %MAP_ADDR_6(p:1)
0006 4e001f           MOV        %MAP_ADDR_6(p),PAGESEL
0009 a602             LDA        #2
000b 3e0000           MOV        #__OVL_Test_v,_Y
000e bc0000           JMP        %FIX16(_FAR_COPY)
```

5.3. Startup Code and __far Pointers

The startup code makes use of pointers when performing the zero-out and copy-down operations. If the application has data above the 0xFF boundary, then the startup code has to use __far pointers. To enforce this, the user has to provide option -D__STARTUP_USE_FAR_POINTERS in the command line when compiling the startup code.

6. Cross-Page Data

Paged addressing does not work if the object crosses page boundaries, because the page selection register would have to be updated before each byte access, which is not the case with paged addressing (the register is written only once, for the first byte of the object).

If the object spans multiple pages, you must enforce far addressing for that object, which means:

- Place the object in a __far segment (using a DATA_SEG / CONST_SEG pragma with the __FAR_SEG modifier)
- Ensure all pointer accesses are performed with __far pointers (pointers declared with the __far pointer qualifier).

You can tell which object, if any, crosses page boundaries, because the linker will report warning L1023 (“Object <object> spans multiple pages”) for each such occurrence.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

Document Number: AN4188

21 September 2011