Freescale Semiconductor
Application Note

Document Number: AN3741
Rev. 0, 08/2008

Implement an In-Application-Programmable Data-Storage on the MC9RS08KA8

System Configuration and Sensor Calibration

by: Jerry Shi
Field Application Engineer
Beijing

1 Introduction

The MC9RS08KA8 (RS08 series) microcontroller is Freescale's general purpose 8-bit device for the ultra low-end market. RS08KA devices are specifically crafted to be more efficient and cost effective for applications that require a small memory size. It inherits most of the S08 core architecture and peripherals but with limited features and performance due to the nature of the low-cost simplified core design. One significant difference is its on-chip flash memory. It has no page-erase scheme or associated build-in charge-pump circuit to generate high the voltage needed for in-application flash re-programming. This makes the RS08 largely different from the conventional flash re-programming approach applied to the S08 series. It remains possible to reprogram part of its on-chip flash contents if an external 12 V is present. This approach is useful if there are small amounts of data that need to be set once or a limited multiple-times by using a rolling-storage mechanism.

Contents

freescale™ semiconductor

2 Flash Programming on RS08

This chapter describes registers and specifications relative to the RS08 flash module and how to program the flash memory procedure.

2.1 Flash Programming Procedure

The RS08 flash memory is programmed on a row basis. A row consists of 64 consecutive bytes starting from addresses $3X00, $3X40, $3X80, or $3XC0. A specific procedure must be strictly followed to program a row of flash memory. Figure 1 is a flow chart for programming flash memory. When high voltage is applied to the flash array, the flash memory is mapped longer to the memory. Fetching from flash may cause fatal error. Commands must be executed in the RAM to program flash memory codes.

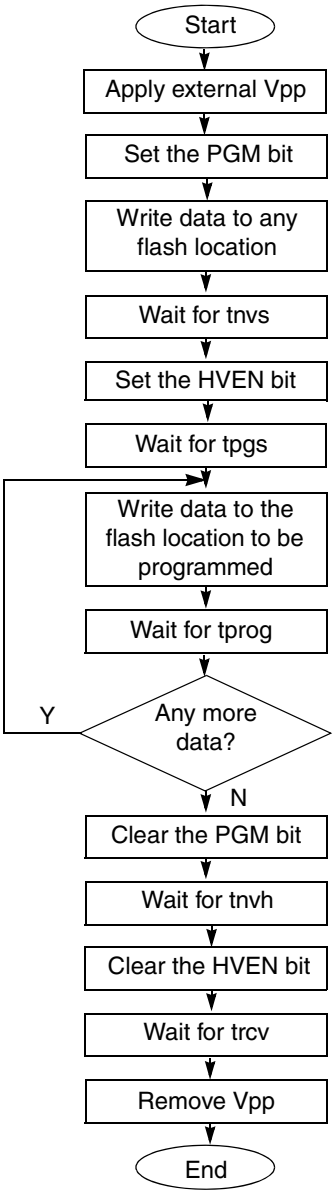

Figure 1. Flash Programming Flow Chat

2.2 Flash Specifications

Flash module specifications must be considered carefully for flash program and erase operation, such as program voltage and timing for the program.

Table 1 lists MC9RS08KA8 specifications that are related to the flash module and operations. For more information refer to the data sheet of a specific RS08 MCU.

Table 1. Flash Specifications

Characteristic	Symbol	Min	Typical	Max	Unit
Supply voltage for program/erase	VDD	2.7	–	5.5	V
Program/erase voltage	VPP	11.8	12	12.2	V
VPP current	IVPP_prog	–	–	200	µA
	IVPP_erase	–	–	100	µA
Supply voltage for read operation	VRead	1.8	–	5.5	V
Byte program time	tprog	20	–	40	µs
Mass erase time	tme	500	–	–	ms
Cumulative program HV time	thv	-	–	8	ms
Total cumulative HV time	thv_total	-	–	2	hours
HVEN to program setup time	tpgs	10	–	–	µs
PGM/MASS to HVEN setup time	tnvs	5	–	–	µs
HVEN hold time for PGM	tnvh	5	–	–	µs
HVEN hold time for MASS	tnvh1	100	–	–	µs
Vpp to PGM/MASS setup	tvps	20	–	–	ns
HVEN to Vpp hold time	tvph	20	–	–	ns
Vpp rise time	tvrs	200	–	–	ns
Recovery time	trcv	1	–	–	µs
Program/erase endurance		1000			cycles
Data retention	tD_ret	15			years

2.3 External Program Voltage Control

The external program voltage must be controlled to apply or to remove from the target microcontroller by either the user program or manually. It is recommended to control it by the user program with an external switch to avoid damage to the circuit due to operating exceptions.

A recommended power supply with program voltage is shown in Figure 2. To switch program voltage an on/off switch is controlled by an I/O pin of the MCU. This on/off switch can be implemented by a transistor. Additionally, a 100 Ù resistor and a 1 nF capacitor are added to filter the program voltage. This filter prevents the program voltage from rising too fast. If the voltage rises too fast this may result in the current overflowing into the pad and cause permanent damage to the internal circuit.

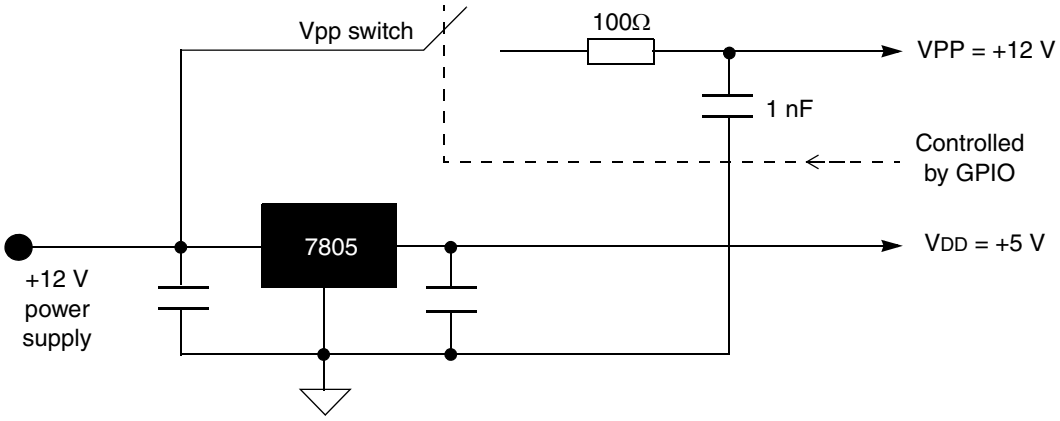

Figure 2. Power Supply with Program Voltage

In Figure 2 the Vpp switch can be an analog switch, a MOSFET switch circuit, or any other electrical switch. Figure 3 shows a switch circuit made with 2 transistors. If the transistor VCE drop is considered, the supply voltage is slightly higher than 12 V.

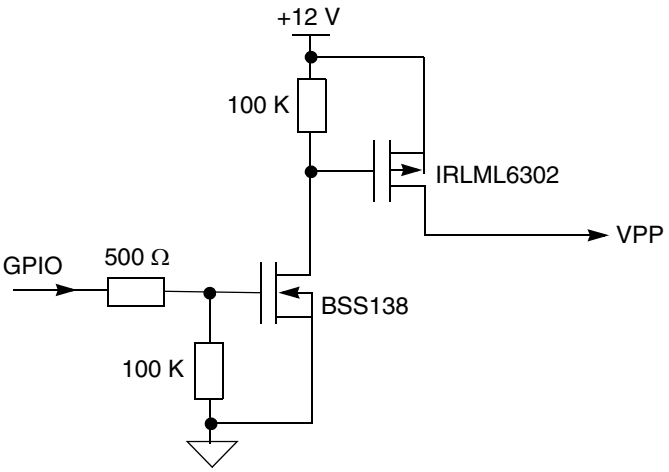

Figure 3. Power Supply with Program Voltage

3 Implement Programmable Data Storage on the MC9RS08KA8

Program and mass erase are two kinds of operations supported by the RS08 flash module. Program operation makes it possible to implement non-volatile data storage on the RS08. Every time the RS08 is mass erased and programmed all unused flash units keep the value of 0xFF. Those values can be programmed by the user program applied with the external programming voltage. Because those units can not be erased by row, they can not be reprogrammed again.

3.1 Reserve Flash Locations for Data Storage

When a project is created, the linker maps all the code and data according to the specified parameter (PRM) file. All the codes are mapped into the ROM segment defined in the default PRM file of a KA8 project:

```
ROM = READ_ONLY      0x2000 TO 0x3FF7;
```

Address range 0x2000 to 0x3FF7 is for the flash module on the KA8 indicated by the key word READ_ONLY. Because the data storage is mapped into the memory their locations must be reserved, therefore no codes can be mapped to the same area. This can be easily done by manually modifying the PRM file.

The RS08 flash memory is programmed on a 64 bytes a row basis, therefore reserved locations for data storage must be integral times of 64 bytes. For example, if the length of configuration/calibration data is equal to or less than 64 bytes, the first row of flash memory can be reserved. The ROM segment in the PRM file must change to:

```
ROM = READ_ONLY      0x2040 TO 0x3FF7;
```

Locations from 0x2000 to 0x203F are reserved for data storage. There is no code mapped to this area.

3.2 Memory Access from the Page Window

Extended memory can only be accessed by the paged or indexed mode because the RS08 CPU core has limited addressing modes. The 14 bit address to access all of the 16 K memory is a combination of the 8 bit PAGESEL register and a 6 bit offset address. Refer to Figure 4. The page window is mapped to the direct address area. It is a 64-byte block with the address ranging from 0x00C0 to 0x00FF.

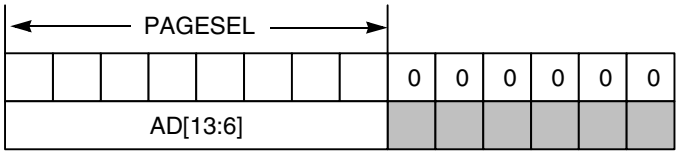

Figure 4. Figure14-bit Address for Paging Access

Occupying the first row of whole flash memory, the data storage reserved in the above PRM file must be accessed through the page window. Table 2 shows how to page access the whole memory.

Table 2. Page Access

PAGESEL	Memory Address
0x00	0x0000 ~ 0x003F
0x01	0x0040 ~ 0x007F
0x02	0x0080 ~ 0x00BF
0x03	0x00C0 ~ 0x00FF
... ...	... ...
0x80	0x2000 ~ 0x203F
0x81	0x2040 ~ 0x207F
0x82	0x2080 ~ 0x20BF
0x83	0x20C0 ~0x20FF
... ...	... ...
0xC0	0x3000 ~ 0x303F
... ...	... ...
0xFE	0x3F80 ~ 0x3FBF
0xFF	0x3FC0 ~ 0x3FFF

To access the flash unit 0x2001, PAGESEL must be written to 0x80; therefore flash memory address 0x2000 to 0x203F is mapped to the direct address window 0xC0-0xFF. If the direct address is 0xC1, the physical address 0x2001 is accessed. For easy paged access operators HIGH_6_13 and MAP_ADDR_6 are defined in the RS08 assembly.

```
MOV#HIGH_6_13(data), PAGESEL;
LDAMAP_ADDR_6(data)
```

The above instructions calculate and move high 8 bits of the data address to the PAGESEL register and then load its value from the page window in the direct page. The two operators calculate automatically the page address and low 6-bit address of a location.

In C, the compile calculates the logical address for every variable.

3.3 Run the Code in RAM

Besides 14 bytes of fast access RAM, the KA8 has 114 bytes direct RAM before the page window (0x002F ~ 0x00BF), and 96 bytes paged RAM after the page window (0x0100 ~ 0x015F). The code can be executed in direct RAM or page RAM. This depends on the RAM usage in an application.

An array can be defined to contain all binary codes for the flash program. Refer to Figure 5.

Special care must be taken for prog_ram[6], prog_ram[9], prog_ram[17], prog_ram[20], prog_ram[22], and prog_ram[25].

Prog_ram[22] and prog_ram[25] are a page address and a direct address in the page window of a designated flash unit that needs to be programmed. Prog_ram[6] and prog_ram[9] are a page address and a direct address in the page window of any location in the same row of the destination flash unit. They can also be addresses of the destination unit itself.

Prog_ram[17] and prog_ram[20] are a page address and a direct address in the page window of source data. Due to the limitation on the RS08 calling convention, it is not passed by stack or any registers, but directly accessed by its address. Make sure to update prog_ram[17] and [20] to the correct values accor ing to the map file complier generated every time a project is created, or do the following every time before prog_ram is called:

```
prog_ram[20] = (unsigned char)&_data;       // ensure that _data in direct page
                                            // otherwise prog_ram[17] must
                                            // be updated either.
```

```
/**
 ** MAP: location - ???, size - 0x2C (44 bytes)
 ** USAGE: A, X (0x0F), PAGESEL, FLCR
 ** INPUT: data (its address must be updated in prog_ram[20] according to the
 **          map file every time project is made.
 ** OUTPUT: FLASH unit (address 0x2000) - be aware of PRM setting to reserve
 **         unit location.
 **/
volatile unsigned char prog_ram[] =
{
  0x3E, 0x08, 0x1F,      // pagesel(FLCR)
  0x10, 0xD1,            // *bset 0, FLCR              ;set PGM (bit0)

  0x3E, 0x80, 0x1F,      // pagesel($2000)             ;
  0xB7, 0xC0,            // *sta map_addr_6($2000)     ;dummy write

  0xAC,                  // nop (1 cycle)              ;delay tnvs 5
  0x3E, 0x08, 0x1F,      // pagesel(FLCR) (4 cycles)

  0x16, 0xD1,            // *bset 4, map_addr_6(FLCR)  ;set HVEN

/*****************  loop to program more bytes *********************/
  0x3E, 0x00, 0x1F,      // page_sel(data) (4 cycles)  ;delay tpgs 10
  0xB6, 0x60,            // !LDA data (3 cycles)       ;get data
  0x3E, 0x80, 0x1F,      // page_sel($2000) (4 cycles) ;
  0xB7, 0xC0,            // *STA addr                  ;write to $3000

  0x3E, 0x04, 0x0F,      // ldx #$02 (4 cycles)        ;delay tprog 20
  0x3B, 0x0F, -3,        // dbnzx relative (7 cycles)
  0x3E, 0x08, 0x1F,      // page_sel(FLCR) (4 cycles)
/*****************  loop to program more bytes *********************/

  0x11, 0xD1,            // *bclr PGM                  ;clear PGM

  0x30, 0x00,            // brn $ (3 cycles)           ;delay tnvh 5
  0x30, 0x00,            // brn $ (3 cycles)

  0x17, 0xD1,            // *bclr HVEN                 ;clear HVEN
  0xBE                   // rts (3 cycles)             ;delay trcv 1
}
```

Figure 5. Array that Contains Binary Codes for the Program

To call this RAM-resident routine use an inline command:

```
__asm JSR prog_ram;
```

3.4 Program Multi-Bytes

If more than one byte needs to be programmed to flash locations, the commands between the two loops to program more bytes must be repeated. The maximum number can not be more than 64.

3.5 Housekeeping for Implementation

To make implementation of in-application-programmable data storage work, the following must be considered.

3.5.1 Code and Data Safety

Flash program routine in the application codes (Figure 5) could potentially harm user code and data. In case this routine runs incorrectly, the user code or data may change. Because there is only one copy of the routine in the RAM, destroy the routine after it is called. Clear every element of the array and fill with a known instruction code such as NOP (0xAC). The data storage can be programmed once, therefore the routine is not used again.

The Vpp requirement may be used to protect against inadvertent program / erase. In this case an add tional protection mechanism must be adopted because an external Vpp can be present.

3.5.2 Bus Frequency

For easy implementation, set the bus frequency to a lower value before calling the routine and restore it to its original value after calling. Bus frequency can be 1 MHz for easy calculation and the necessary delay time.

```
ICSC1 = 0b00000110;        // bus frequency = 1 MHz
ICSC2 = 0b11000000;
```

3.5.3 COP and CPU Interrupt

The MCU must not enter STOP mode while the flash program routine is running. In case execution time is longer than the COP overflow time, the COP must be disabled to avoid COP reset. Evaluate COP overflow time and routine execution time. Feed the COP before calling.

Code sequence can not be interrupted by exceptions or external conditions because RS08 CPU does not support real interrupt.

3.5.4 Modes of Operation

Different running modes in the user program can be designed for system configuration and/or sensor calibration. The program enters a different mode according to conditions after reset. An external condition can be a key button on the circuit board that displays a different level when the system is powered up. See Figure 6.

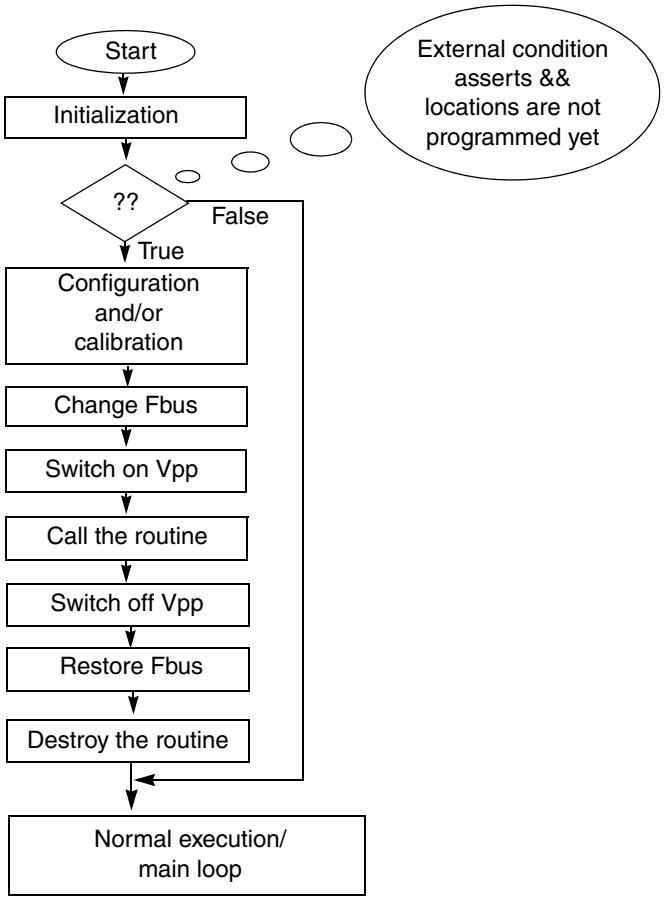

Figure 6. User Program Flow Chart

3.5.5 Reuse of RAM

If more than 40 bytes are used as a dedicated container of the flash program routine RAM may not be sufficient in some cases because on-chip resources are usually limited on the RS08.

After power-up reset, the data can be programmed once. It is possible to reuse RAM in the remaining code flow.

A good approach in C programming is to use a memory overlay technique to fully use the limited RAM resource. The binary codes can be stored in the flash ROM. If needed for normal application purposes copy the code bytes to the designated RAM buffer that overlays with all other variables. After the routine is called and returns memory that is allocated for prog_ram[] it is released and free for any other variables. All RAM units can be re-used automatically by updating the a project prm file and accordingly allocating variables to different segments. For example, in the original prm file of a KA8 project the segment RAM is defined as:

```
RAM=READ_WRITE0x0030 TO 0x00BF;
```

A new segment duplicated with RAM can be appended:

```
CODE_RAM=READ_WRITE0x0030 TO 0x00BF;
```

Then add a new user segment into:

```
PRG_CODE_DATAINTO CODE_RAM;
```

In the source file define prog_ram[] and other variables as:

```
#pragma DATA_SEG            PRG_CODE_DATA
unsigned char prog_ram[];
#pragma DATA_SEG default
unsigned char i, j, k;
```

prog_ram[] and global variables i, j, k may be duplicated while allocated. No compilation error is generated.

Do not touch any of the units in prog_ram[] code array before the routine is called. They can be used as general data buffer in the main function after the routine is called.

3.5.6 Possible Multi-Time Programming

A rolling-storage mechanism can be implemented to achieve limited multi-time programming when a few bytes of data are saved in a flash row of 64. The user can design a unique algorithm to put all the data into a group with a tag indicating its validity.

3.5.7 Considerations for Migration

To migrate this implementation to another RS08 MCU consider the following points:

- Memory map of the target MCU
- Change linker parameter file
- Other specifications

4 Conclusion

This application note implements a non-volatile in-application-programmable data storage on the MC9RS08KA8. This data storage can be used to store the data generated in the application such as sytem configuration and sensor calibration. After the device has been manufactured or installed, this is useful for applications where important data must be automatically calculated and stored to non-volatile memory.

5 Reference

MC9RS08KA8, Revision 0 Draft G, 11/2007, Datasheet.

MC9RS08KA8RM, Revision 0 Draft I, 10/2007, Reference Manual.

Freescale website, www.freescale.com

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3741
Rev. 0
08/2008