Freescale Semiconductor
Application Note

Document Number: AN3958
Rev. 0, 10/2009

Integrating the MC9S08JS16/8 USB Bootloader to Your Application

by: Derek Liu
Application Engineer

Freescale provides the USB bootloader solution for the JS series of MCUs. This solution comprises the ROM-based USB bootloader and the PC GUI. (Flash-based USB bootloader is designed for the JM family of MCUs, refer to AN3561 and AN3748). You can use it easily without any modifications. The PC GUI example provided by Freescale is a stand-alone program which can run on the Windows XP platform (SP2 or later version). The ROM-based USB bootloader cannot be customized, but the software on the PC side can be replaced according to different requirements.

Some knowledge or details about the USB bootloader working scheme and protocol are discussed in this application note, furthermore, how to re-use the flash operation algorithm in 9S08JS16 boot ROM is also involved. This provides the necessary information for creating a new application with USB bootloader or integrating the USB bootloader to an application on a different platform (Linux or Mac).

Contents

freescale™ semiconductor

1 Understanding the USB Bootloader Protocol

To minimize the bootloader code size, a simple protocol has been worked out. Based on this protocol, all bootloader work can be implemented on the Endpoint 0 by the control transfer. The firmware download or upgrade process is composed of a series of vendor-defined USB requests.

The USB bootloader code does not need to implement any USB standard classes, or specific code for processing the other USB endpoints' communication. The enumeration process comprises some USB standard requests, and the handling of the USB standard request is essential to all USB devices.

USB standard request is implemented by the USB control transfer on Endpoint 0. The USB control transfer comprises of three stages: setup stage, data stage, and status stage. The setup and status stages are essential, while the data stage can be omitted, or can include one or more USB transactions. In Figure 1, Transfer 0 is a USB-control transfer without data stage. Transfer 1 has three stages, transaction 103 is in setup stage, transaction 104 is in data stage, and transaction 105 is in status stage. For detailed description of the control transfer, please refer to chapter 5 of the USB specification 2.0 — USB Data Flow Model.

The processing of the standard request and vendor-defined request is similar; this helps to decrease the footprint of the bootloader code by code reuse. Once the enumeration process is completed, the USB bootloader GUI can send the vendor-defined requests to implement the bootloader, such as erasing flash page, programming, and verifying.

1.1 Implementation of the USB Bootloader Command

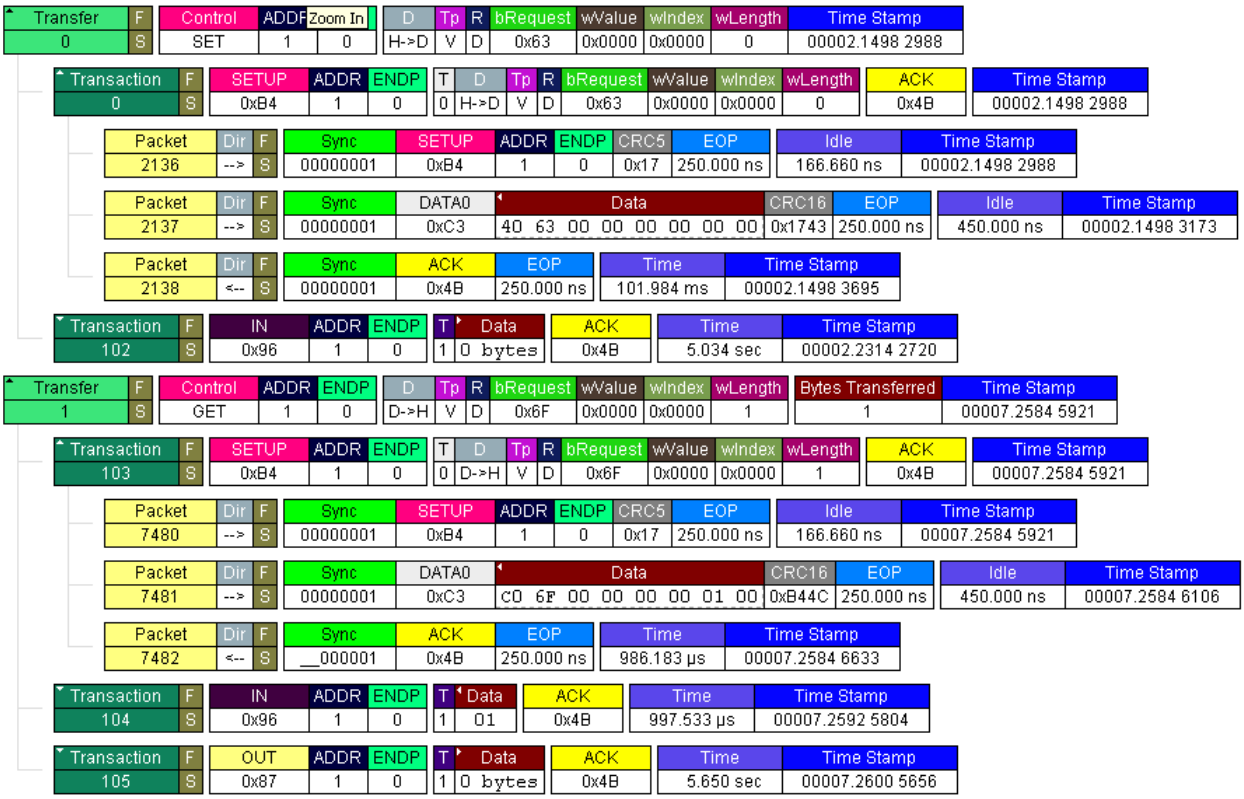

Figure 1. The USB Transfer for Mass-Erase Command

Figure 1 illustrates the implementation of the mass-erase command.

The first transfer (0) sends the mass-erase command to the bootloader, and the second transfer (1) inquiries the result of the mass-erase command.

The vendor-defined request (transaction 0, packet 2137 in Figure 1) is sent to the bootloader. The data 0x40 in packet 2137 indicates it is a vendor-defined request and the direction is from host to device; the data 0x63 is the mass-erase command. You can refer to chapter 9 of USB specification 2.0 (USB device framework) for more information about the USB request.

In the transfer 1, a vendor-defined request (transaction 103) is issued for getting the result of the mass-erase command. The 0xC0 indicates it is a vendor-defined request and will get data from device.

Transaction 104 includes the responding data for the request in the setup stage (transaction 103). Data 0x01 is sent to host from device, which indicates that the mass-erase command is successfully executed.

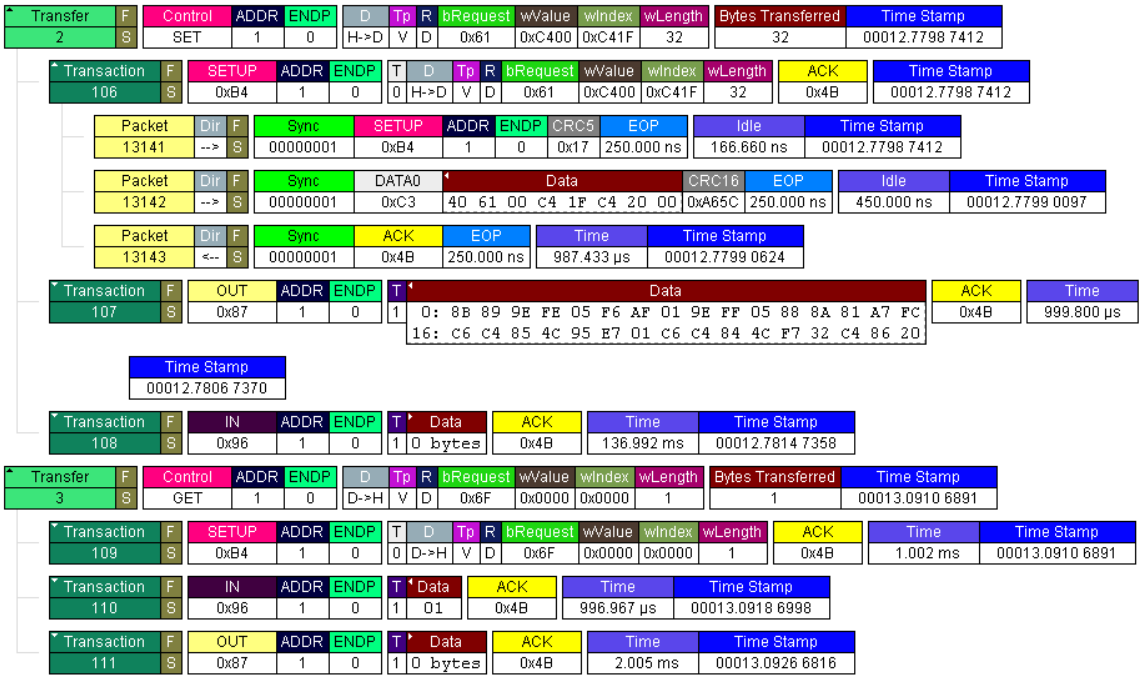

Figure 2. The USB Transfer for Program Command

Figure 2 illustrates the USB transfer for program command. The data in packet 13142 of transfer 2 is a vendor-defined USB request. The 0x61 (bRequest field) is the program command, 0xC400 (wValue: 00 C4) is the start address for programming, 0xC41F (wIndex: 1F C4) is the last address, 0x0020 (wLength: 20 00) is the data length.

In the transaction 107, 32 bytes are sent to the bootloader device. The programming result is returned to the host in the transfer 3.

1.2 USB Bootloader Protocol

All vendor-defined requests for the JS family MCU USB bootloader are listed in the Table 1. It is simple and easy to be implemented without too much code.

Table 1. The USB Bootloader Protocol for the JS Family of MCUs

Command	BmRequest	bRequest	wValue	wIndex	wLength	Data	Return value
Program	0x40	0x61	Start Address	End Address	Data Length	Data	0x01: Success 0xF4: Fail
Mass Erase	0x40	0x63	0x0000	0x0000	0x0000	N/A	0x01: Success 0xF0: Fail
Partial Erase	0x40	0x64	0x0000	0x0000	0x0000	N/A	0x01: Success 0xF1: Fail
Reset	0x40	0x68	0x0000	0x0000	0x0000	N/A	N/A
CRC check	0x40	0x69	0x0000	0x0000	0x0000	N/A	0x01: Success 0xF5: Fail
Get Result	0xC0	0x6F	0x0000	0x0000	0x0001	Result	Return the execution result of the last command

Note: the return value column lists the value when executing the Get_Result request.

The mass-erase command is similar to the partial erase command from the view of host. The only difference is that the host needs to wait more time before inquiring the result, because the boot code erases the flash page by page.

Unlike the program command, the mass-erase, partial erase, checksum, and reset commands have no data stage in the control transfer.

After the erase, mass-erase, program, and checksum commands are sent to the bootloader successfully, the Get result command must be issued to check whether the command is executed successfully or not.

The boot code executes an illegal CPU instruction (0x8D, which is not an HCS08 CPU instruction) to trigger the ILOP (illegal operator) reset after receiving the Reset command (0x6F). The host will lose the USB connection with the MCU once the control transfer for the reset command is completed.

The USB bootloader does not support uploading the code back to the PC GUI for security reasons.

2 Customizing the USB Bootloader Tool

With the protocol introduced in Section 1, “Understanding the USB Bootloader Protocol,” you can develop your own USB bootloader tools based on specific requirements, such as operating system, GUI, a command line program, ActiveX, and so on. The bootloader tool maintains the connection between the host and the bootloader device, selects S record file, and downloads it to the MCU. Development of a bootloader tool has two important parts: USB driver and S record file parser.

2.1 USB Driver

Refer to the USB bootloader protocol described in Section 1, "Understanding the USB Bootloader Protocol." All bootloaders' communication can be finished on the endpoint 0 by the control transfer, a USB driver which supports control transfers is enough. The USB bootloader GUI provided by Freescale is based on the WinUSB libraries from Microsoft. You can develop your own USB driver, or use a free USB driver.

For flash-based USB bootloader (9S08JM family of MCUs), you can modify the product ID (PID), vendor ID (VID) and string descriptor, and the method to enter the bootloader mode. But, you cannot change the VID, PID, and string descriptor for ROM-based USB bootloader (9S08JS family) because they are kept in the ROM. You can refer to AN3561, AN3748, and the 9S08JS reference manual for more information. The VID and PID for the 9S08JS family bootloader are 0x15A2 and 0038.

WinUSB (from Microsoft) and LibUSB are free popular USB drivers. You can download them from the Internet and use them in your GUI application.

2.2 S Record File

The bootloader is designed to support the S record file. The USB boot tool reads the S record file to get the start address, end address, code, data length, and transfers them to the bootloader device based on the protocol in Table 1.

The S record file is an ASCII file; you must understand its format before using it. The following is a brief description for the S record file.

The S record file has three kinds of format:

- S1 — for 16-bit address.
- S2 — for 24-bit address.
- S3 — for 32-bit address.

S0 is the header record.

for example:

S00600004844521B

Description:

- S0 — Identifies the record as a header record.
- 06 — The number of bytes following this one.
- 0000 — The address field, which is ignored.
- 484452 — The string HDR in ASCII.
- 1B — The checksum.

Data record:

1	2	3	4	5	6	7	8	9	.....	8+n	9+n	10+n
S	1	byte count (n)		address				data 1		data n	check sum	

1	2	3	4	5	6	7	8	9	10	11	.....	10+n	11+n	12+n
S	2	byte count (n)		address						data 1			data n	check sum

1	2	3	4	5..12	12	.....	12+n	13+n	14+n
S	3	byte count (n)		address	data 1			data n	check sum

- S9 is the terminator for 16-bit address S record.
- S8 is the terminator for 24-bit address S record.
- S7 is the terminator for 32-bit address S record.

The S9 (8 or 7) includes the start address after the MCU reset; it's of no use in this bootloader solution. They can be omitted in the application.

It's enough for you to develop the bootloader tool with the above introduction. You can refer to the other documents to get a detailed description for the S record file. (for example, the help system of CodeWarrior has a detailed description for the S record).

2.3 Develop Your Bootloader Tool

After understanding the USB bootloader working scheme, you can design your own bootloader tool based on the USB driver and the S record file parser. Generally, follow the steps below to develop the bootloader tool:

1. Load the USB driver.
2. Check if the USB device is connected.
3. Select the S19 file to be downloaded.
4. Execute the mass-erase, partial-erase, program, checksum, and reset command according to the correct sequence when the bootloader device is connected and working in the bootloader mode. The S19 file needs to be parsed when programming.

You must set the appropriate and enough timeout value for the control transfer when the command is sent to the USB bootloader device. The USB boot code executes the command immediately after it received it. Different command may take different time to complete. The bootloader code responses the last transaction (state stage) of control transfer only when the command is executed completely. So the control transfer may fail if the last transaction is not responded because of the timeout.

For the bootloader of the MC9S08JS16, the timeout limitation for mass-erase command must be longer than 120 ms. (You can refer to the mass erase time declared in the reference manual.) That for partial erase is at least 650 ms (this value increases with the flash size because the flash is erased page by page). The program and reset command do not take much time; 1 ms time interval is enough. You can select the mass-erase or partial-erase according to your application. The bootloader tool must make sure the flash is erased before programming.

3 Utilizing the Flash Operation Algorithm in Boot Rom

The 9S08JS MCU reserves the first 1 KB in its flash space for emulating the EEPROM. You can use this section to keep the calibration information or some parameters which may change in the working process, but hold after power off.

Because of the limitation of the flash (can not erase, program, and run the code in one flash module at the same time), the general method is to copy the flash operation algorithm to RAM and run it in RAM (refer to AN2295, AN3561, and AN3748). This gains at least 50 bytes of RAM space for an application.

The flash operation algorithm in the boot ROM is designed only for USB bootloader in design phase, but now it can be used with a backdoor. The MCU can jump to boot code section and execute the program/erase algorithm. The following section provides an interface to call the flash algorithm in boot ROM. It can be found in attached CodeWarrior project (AN3958SW).

3.1 Variables Declaration

The flash operation algorithm uses variables and pointers to locate the fixed address. The data and object address are transferred to the erase or program sub-routine through these parameters (*flash_operation_api.c*):

```
char Src_Data[32]@0x18C0; //save the data to be programmed
char Data_Len @0x8C //the actual data length to be programmed
unsigned int Obj_Start_Addr@0x0088;   //the start address  for programming or erasing
char Stat@0x87; // the result of erase or program; success or fail
char Index@0x86; //occupied by programming sub-routine
```

where:

- **Src_Data** keeps the data which will be programmed to flash.
- **Data_Len** is the length of the data which will be written to flash.
- **Obj_Start_Addr** is the start address which will be programmed.
- **Stat** is the result for flash operation.
- **Index** is used by the programming sub-routine; its value will be changed after calling the programming routine.

The flash algorithm in bootloader code supports to program up to 32 bytes at one time. The data buffer is fixed start from 0x18C0. You can put the data which will be programmed to this buffer. In general, the Src_Data (at 0x18C0) is located in USB RAM, so this must be taken into consideration.

NOTE

Make sure the data in the area of 0x18C0–0x18DF cannot be overwritten in programming, in addition, the MCU interrupt must be disabled during flash program or erase operation.

3.2 Calling the Flash Operation Routine

The following routine (in *flash_operation_api.c*) is defined for calling the flash operation algorithm located in the boot ROM.

```
char Flash_Erase(unsigned char* Addr)
{
  Obj_Start_Addr =  (unsigned int)Addr;//Initialize the Start Address
  asm
  {
    LDHX Obj_Start_Addr //Load the page address to HX register
    JSR  PAGE_ERASE_ROUTINE_ADDR  //Call page erase routine
    STA  Stat                      //save the return value
  }
  return Stat;
}
```

In above routines, PAGE_ERASE_ROUTINE_ADDR is the entry address for the flash-erase operation. The Flash_Erase routine returns 0 when the flash page is erased successfully. The Addr parameter can be any address in the flash page to be erased.

The following code (in *flash_operation_api.c*) defines the method to call the program algorithm in the boot ROM. Flash_Program calls the programming routine located at PROGRAM_ROUTINE_ADDR. After that, it returns the programming result, 0x01 is successful, and all the other values are failed.

```
char Flash_Program(void)
{
  asm
  {
      JSR PROGRAM_ROUTINE_ADDR// call program routine
  }
  return Stat;    //return the value, 0x01: success, other : failed
}
```

3.3 Program or Erase Operation

The following code example (in *main.c*) shows how to erase the flash page and program the data.

```
  DisableInterrupts;       //disable interrupt
  Ret= Flash_Erase((unsigned char*)0xC000);     // erase the 0xC000 page

  if(Ret== 0x00)           //if the page is erased successfully
  {
    Data_Len = 32;         //programming length
    Obj_Start_Addr = 0xC000;//programming start address

    Ret = Flash_Program();//Call programming routine

     for(i= 0; i<32; i++)        //initialize the data which will be programmed
```

```
        *(pData+i)  = i;

    Ret = Flash_Program();   //continue to program the data

}
```

Interrupts are disabled before starting erasing or programming, and the flash page must be erased before programming. The programming operation is executed when the flash page is erased successfully.
The Obj_Start_Addr will be changed to a new address by the programming algorithm.
The Obj_Start_Addr value will be changed to 0xC020 after the first Flash_Program routine is called.

The code above can be used as an example; the data, address, and data length can be changed according to the application.

4 Summary

The USB bootloader makes the firmware upgrade easy and fast. The ROM-based USB bootloader saves much of the flash space, because you need not consider the flash operation algorithm, interrupt redirection, flash protection, and so on (refer to AN3561 and AN3748 for flash-based USB bootloader).

With the principle, protocol of the USB bootloader and the S record file format described in this document, you can communicate with the USB bootloader, develop your own USB bootloader GUI, and integrate the USB bootloader to your applications very easy.

By re-using the flash operation algorithm in boot ROM, the RAM usage is decreased in your application.

Appendix A The CodeWarrior Project for Using the Flash Algorithm in Boot ROM

The CodeWarrior project for how to use the flash operation algorithm in boot ROM is attached (AN3958SW.zip). This project demonstrates how to call the flash operation routine in boot Rom, to erase and program the first two flash pages located at 0xC000.

This project can be compiled with CodeWarrior 6.2 or later version, and it can be downloaded to the MC9S08JS16 demo board for verification. The data buffer content at 0x18C0 can be set to different values by pushing the PTG2 button on demo board, and updating the flash at 0xC000 and 0xC200 page when the PTG3 button is pressed.

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
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3958
Rev. 0
10/2009

freescale™
semiconductor