Freescale Semiconductor
Application Note

Document Number: AN3967

How to Write Flash Programming Applets

1. Introduction

Contents

1.1. Purpose

The purpose of this document is to help understand and create Flash configuration files for the Flash Programming interface.

1.2. Scope

The flash operations are coded in microcontroller specific files called the Flash Configuration files (extension XML). The `*.XML` files contain microcontroller- dependent parameters and applets to handle internal flash modules.
Applets are small programs which are loaded into the on-chip RAM. They will run at well-defined speed (depending on the microcontroller clock speed) and can collect data about the state of the memory modules, can control the memory modules (including erasing and programming).
There are three applets for each flash module:

- INFO applet — Collects information about the module.

- CONTROL applet — Controls and erases the module.
- PROGRAM applet — Programs the module.

Parameters are passed to these applets at fixed memory addresses, which are relative to the workspace. `*.XML` files are written by Freescale and are delivered within the installation.

2. Flash Configuration Files General Description

2.1. Basic Concept

A Flash Configuration file is an XML format file. Its file extension is always “XML”. It is possible to open and edit this file with any text editor. This file is a “handmade” file and is not created automatically with any tool.

Some microcontrollers can address the flash in multiple ways — banked and paged. Therefore, multiple overlapping devices can be defined for one microcontroller. Each of them is defined in its own Flash Configuration XML file.

2.2. Structure Overview

Each Flash Configuration file has a main section named `content` where the global parameters are defined. It also contains two sections:

- `Sectors` — Describes sector organization.
- `Organization` — Defines the applets used and their parameters.

2.3. Structure Step-by-Step

Listing 1 shows the main structure of a Flash Configuration file. The microcontroller used in this application note is `MC9SO8QE128`.

Listing 1. Main Flash Configuration File Structure

```
<?xml version="1.0" encoding="UTF-8"?>
<device-file>
<device>
<content>
    <name>MC9S08QE128_FLASH_C000</name>
    <manufacturerid>103e</manufacturerid>
    <hardware_mmu>1</hardware_mmu>
    <sectors>
         <sectorcount>1</sectorcount>
         <sectorsize>4000</sectorsize>
    </sectors>
    <buffer_start>80</buffer_start>
```

```
    <buffer_size>1780</buffer_size>
    <organizationcount>1</organizationcount>
    <organization>
       ……
    </organization>
</content>
</device>
</device-file>
```

The tags and their description are:

- `content` — Base tag for the device description.
- `name` — Flash device name that will be displayed in Code Warrior Flash Programmer user interface. It starts with the name of the device that uses it – `MC9S08QE128` and is followed by the module name – `FLASH_C000`. The Flash Configuration filename must match this name.
- `manufactureid` — Microcontroller's identification (MCUID) defined in the hexadecimal format.
- `hardware_mmu` — If Memory Management Unit is present, the value is 1 else 0.
- `sectors` — Describes the module's sector organization. It uses 2 tags: `sectorcount` and `sectorsize`. The first defines how many sectors of `sectorsize` size follow. If we have sectors of multiple sizes, the tag pair will be repeated.
- `buffer_start` — Address in hexadecimal for RAM memory buffer where the applets and their parameters are loaded.
- `buffer_size` — Size of the memory buffer in hexadecimal. Usually, this matches the full microcontroller RAM. A larger memory buffer can improve the speed of programming the microcontroller.
- `organization_count` — Specifies the number of organizations. An organization is a way to access the flash module. HCS08 and RS08 microcontroller's have only one so this field will always be one.
- `organization` — Details the applets used and their parameters.
- `vpp` – Placed in the `<content>` tag. Used to activate programming voltage on derivatives that need it, usually RS08 Microcontrollers. If not required, should be skipped or set to false. See `MC9RS08KA2` xml files from the microcontroller layout.

2.4. Advanced Information

The applets are loaded in the Microcontroller's RAM/SRAM. Therefore, the buffer start address (here `0x80`) set in the `buffer_size` tag is the default RAM start address of the Microcontroller. However, it is also possible to set a different buffer starting address.

If the applet code is relocatable, you can modify the buffer address parameter. Otherwise, the execution of the applet code is hazardous.

The buffer size (here `0x1780`) matches the full Microcontroller RAM size. The programmed application is first loaded in the RAM into a buffer placed after the applet code. The applet will copy it to the desired location in flash. The buffer should be as big as possible for a faster programming. The Flash Programmer will download the data to the buffer andthenstarts the applet. The Microcontroller is stopped after programming the buffer. The Flash Programmer will place new data in the buffer and the process continues with starting the applet until no more data to be programmed is left. If the buffer is larger, we have less buffers therefore the total time is shorter due to less interrupts.

3. Applet Creation

3.1. Overview

Applets are small programs which are loaded into the on-chip RAM. They run at well-defined speed (depending on the microcontroller clock speed) and can:

- collect data about the state of memory modules and
- control the memory modules (including erasing and programming).

There are three applets for each module to:

- collect information about the module,
- control (includes erasing) the module, and
- program the module.

Parameters are passed to these applets at fixed memory addresses, relative to the buffer start address.

The applet consists of three areas: the parameter block, the code, and eventually a buffer which may hold the data to be programmed.

The first two areas, parameter block with initial values and the code are loaded into the buffer area on the target Microcontroller‘s RAM/SRAM. The parameters contain instructions for the applet and are set before starting the applet. The buffer with data to be programmed is written after the code section. Finally, the applet code is started. It will read the parameters, interpret them and execute the desired function. The applet may now terminate or run continuously until it is stopped.

These are the requirements for applets:

- Consists of the parameters, code, temporary storage (stack, registers), and a buffer for the data to be programmed (for program applet only).
- Shall fit into the on-chip RAM/SRAM (which is available on all Microcontrollers today).
- Acts as one function (one entry point) which is started to execute commands. It will stop after execution of each command (except for the clock speed sensing and fast download mode).
- Flash programmer sets the required parameters of the command into the parameter area prior to the start of the applet. The results will be in the parameter block.
- Code of the applets is loaded from a `*.abs` or `*.s19` file.

- Target interface independent.

3.2. Applet Parameter Structure

The applet parameters are defined in the `organization` tag. For `MC9S08QE128`, it is included in Listing 2.

Listing 2. *organization* Tag for MC9S08QE128

```
<organization>
        <name>FLASH_C000</name>
        <id>103e</id>
        <begin_address>c000</begin_address>
        <end_address>ffff</end_address>
        <applet>
                <program>PROGRAM_GENERIC_NEW_COP.abs</program>
                <control>CONTROL_GENERIC_SECTOR.abs</control>
                <info>INFO_FLASH_GENERIC_TO_REMAP_NEW_COP.abs</info>
        </applet>
        <params>
              <global>
                    <name>S_SIZE</name>
                    <value>1</value>
              </global>
              <global>
                    <name>F_SIZE</name>
                    <value>1</value>
              </global>
              <global>
                    <name>C_SIZE</name>
                    <value>1</value>
              </global>
              <global>
                    <name>ADR_SIZE</name>
                    <value>3</value>
              </global>
              <global>
                    <name>DCNT_SIZE</name>
                    <value>2</value>
              </global>
              <global>
                    <name>CNT_SIZE</name>
                    <value>4</value>
              </global>
              <global>
                    <name>NVMIF_VERSION</name>
                    <value>2</value>
              </global>
              <global>
                    <name>NVMIF2_BUFADR_SIZE</name>
```

```
                <value>2</value>
            </global>
              <program>
                  <name>LoopCycles</name>
                  <value>1</value>
              </program>
              <program>
                  <name>DelayTime</name>
                  <value>1000</value>
              </program>
              <program>
                  <name>BaseCycles</name>
                  <value>0</value>
              </program>
              <control>
                  <name>LoopCycles</name>
                  <value>1</value>
              </control>
              <control>
                  <name>DelayTime</name>
                  <value>1000</value>
              </control>
              <control>
                  <name>BaseCycles</name>
                  <value>0</value>
              </control>
              <info>
                  <name>RefTime</name>
                  <value>12</value>
              </info>
        </params>
    </organization>
```

Following tags are available in the `organization` section:

- `name` — Name of the module; displayed near the full name from content section.
- `id` — Microcontroller's `mcuid` defined in hexadecimal format.
- `begin_address` — Start address for the flash module in hexadecimal format.
- `end_address` — End address for the flash module in hexadecimal format.
- `applet` — Defines the applets used for accessing the flash module. All applets must be placed in *{CodeWarriorInstallDir}*`\MCU\bin\plugins\support\Flash_Programmer\HC08` or *{CodeWarriorInstallDir}*`\MCU\bin\plugins\support\Flash_Programmer\RS08`

  It uses the tags:

  - `program` — Applet used to program the module.
  - `info` — Applet used to get information regarding the flash status and Microcontroller's speed.
  - `control` — Applet used to erase the module.

- params – Defines the applet parameters. They can be specific to one applet or apply to all of them. Besides its scope, each parameter has two tags: name and value.

3.3. Global Parameters Defined in Flash Configuration File

The params section defines applet-specific or global parameters.

The global parameters defined within the *global* tag are:

- S_SIZE — Size of state in bytes. The state contains the current device state. Example: erased, programed.
- F_SIZE — Size of flags in bytes. Used to control applet execution. It selects operations that will be performed.
- C_SIZE — Size of command in bytes. Used by control and program applets to know what operation must be done.
- ADR_SIZE — Address size in bytes. Keeps address size access to target.
- DCNT_SIZE — Size of delay counter in bytes.
- CNT_SIZE — Size of loop counter in bytes.
- NVMIF_VERSION — Must be set to 2 for the HCS08 Microcontroller and 3 for the RS08 Microcontroller. It is used to compute various parameters in a specific way for each of the architecture.
- NVMIF2_BUFADR_SIZE — Size of buffer address.

3.4. INFO Applet

The INFO applet collects the module-specific data (Microcontroller speed, flags, and state). It is possible to run an infinite loop where a 32-bit counter is incremented to evaluate the clock speed of the Microcontroller.

3.4.1. Insertion in Configuration File

In the params tag all INFO-specific parameters are defined within the info tag. The info tag defines the name and the value of the parameter.

The RefTime parameter matches the number of CPU cycles of the infinite loop in the applet. For information on how to evaluate this constant, refer to Listing 3.

Listing 3. Evaluating Constant

```
<info>
      <name>RefTime</name>
      <value>12</value>
</info>
```

Here, the parameter name is RefTime and the value 12.

If the RefTime value is set to null, the CPU speed is not evaluated (peculiar Microcontroller cases) by the Flash Programmer.

If the RefTime is a positive integer, the speed loop must be implemented in the INFO applet and triggered on the speed test flag.

3.4.2. INFO Source File

The info applet is used to obtain information about Microcontroller. One possible implementation for the MC9S08QE128 derivative is shown here.

```
            org   $80
```

where, base is the first SRAM address, similar to the one specified in the Flash Configuration with buffer_start.

```
cnthi:      dc.w  $0000
cntlo:      dc.w  $0000
            dc.b  0
baseAdr:    dc.w  FbaseAdr
            dc.b  0
size:       dc.w  Fsize
flags:      dc.b  $80
state:      dc.b  $00
```

- cnthi and cntlo fields are usually used to evaluate the Microcontroller's speed. In case not used, the fields still must be present in order to keep the applet structure aligned with the interface.
- baseAdr – start address for the flash device. The size of baseAdr is 2-, 3-, or 4-bytes long as defined by the ADR_SIZE global parameter.
- size – the flash size.
- flags – command to be executed.
- state – the returning state.

All values are populated by the Flash Programmer before starting the applet. If any of them is constant, it can be declared from the code.

Flags must be set as below:

- bit 2: 0 — No enable/disable possible on module, 1 — Enable/disable allowed on module.
- bit 1: 0 — Module not protectable, 1 — Module protectable.
- bit 0: 0 — No fast load on module, 1 — Fast load on module (when interactive R/W is possible (BDM)).

Other bits are initially set to 0.

These variables let you interface with different operations that can be performed on the module. Therefore, these operations must be implemented in the CONTROL applet. For example, if bit 1 is set in flags, it means that the assembler code to unprotect the module is available in the CONTROL applet source file.
The bit 7 of flags is tested to decide if the program must enter (before all tests) an infinite loop for speed Microcontroller evaluation. If the bit is not set, the different module test can be performed.

```
BRSET 7,flags,CountLoop ; if count command bit set (flags & 0x80) start counting loop
```

The different module states returned in the state variable are:

- bit 2: 0 — Module unprotected, 1— Module protected.
- bit 1: 0 — Module programmed, 1 — Module blank.
- bit 0: 0 — Module disabled, 1 — Module enabled.

b0, b1, and b3 bits will be returned to the interface.

Other bits are not used.

NOTE It is important that state and flags are properly adapted. If you set b1 in flags to protect/unprotect availability, the returned value b2 in state should be correctly set. Otherwise, the status interpreted by the interface is not relevant.

```
Done:       BGND
            BRA  NotBlank
```

The applet program ends when it encounters a background instruction encapsulated in an endless loop.

The loop below is the Microcontroller speed evaluation loop. cntlo and cnthi are incremented permanently. The loop is halted by the interface itself.

The RefTime parameter is defined from here.
Speed loop for CPU speed detection runs an endless loop that increments the cnthi/cntlo variable. The loop is not compulsory, but must be present if the RefTime parameter is not null.

```
CountLoop:  LDHX #0
Loop:       AIX  #1
            STHX <cntlo
            CPHX #0
            BNE  Loop
            LDHX cnthi
            AIX  #1
            STHX <cnthi
            BRA CountLoop
```

Stack workspace at the end of the applet code:

```
stack:      dc.w  0,0,0,0,0,0,0,0     ; stack
stackEnd:
```

The `INFO` applet collects the module-specific data (start address, end address, flags, state) of the module.

To evaluate the clock speed of the Microcontroller, it is possible to run an infinite loop where a 32-bit counter is incremented.

3.5. CONTROL Applet

The `CONTROL` applet implements some commands to control the state of the memory block. At best, it is possible to erase, disable, enable, protect, and unprotect a module.

3.5.1. Insertion in Flash Configuration File

The `CONTROL` applet must define three parameters in the `params` section: `DelayTime`, `LoopCycles`, and `BaseCycles`.

They are used to parameterize the timing of the applet code (delay loops) according to the real Microcontroller clock-frequency.

The value of the parameters is:

- `DelayTime` — Time in microseconds by which you want to delay the applet when the delay function is called. For example, if you want to add a millisecond delay in your applet, `DelayTime` must be 1000.
- `LoopCycles` — Number of cycles the delay loop takes to execute one loop.
- `BaseCycles` — Number of cycles required to call and exit the delay loop (cycles to execute the delay function with a count of one). These values are used to calculate the value of the counter of the delay loop to generate a delay of `DelayTime` in the applet.

The following expression is used:

```
counter = clock-frequency (Hz) * DelayTime (µs) / LoopCycles / 1000000 - BaseCycles;
```

The formula is evaluated by the interface, according to the real value estimated or the given value of the Microcontroller clock-frequency. The counter value parameter is then passed to the applet by the interface.

NOTE All the parameters (variables) of the CONTROL applets except state and command are overwritten as soon as the CONTROL applet is called. Therefore, initialization of these parameters is only indicative. However, all parameters evaluated by the INFO applet overwrites the parameters of the CONTROL applet.

The command parameter is passed by the interface, when the APPLET call is performed.

3.5.2. CONTROL Source File

The Control applet erases the flash contents. A possible implementation for HCS08QE128 is shown below.

```
            org    $80
                        base        equ    $0080        ; SRAM start
org    base ;
```

where, base is the first SRAM address and is similar to the one specified in the Flash Configuration file.

```
; Parameters
            dc.b   0
baseAdr:    DC.W   $c000
            dc.b   0
size:       DC.W   $4000
command:    DC.B   $01
state:      DC.B   $00
ClockSpeed: DC.W   0
```

As explained above, these parameters are overwritten. The state parameter must be set to null. The size of baseAdr is 2-, 3-, or 4-bytes.

```
;command dispatcher
BRSET command,#$01,Erase
BRSET command,#$02,Disable
BRSET command,#$04,Enable
BRSET command,#$08,Unprotect
BRSET command,#$10,Protect
DEC    state          ; command not implemented, return -1/FF
LBRA   Wait
```

All operation routines must be implemented according to the Microcontroller availabilities, the dispatcher, and the INFO flags parameter settings. The following code erases one module.

```
Operate:
    LDA     #$FF
    STA     FPROT
    LDA     #FPVIOL+FACCERR
    STA     FSTAT
    LDA #$FE
FDest:
    STA $1234
cmdpatch:
    LDA     #ERASE
    STA     FCMD
    LDA     #FCBEF
    STA FSTAT
    LDA     FSTAT
    AND     #FPVIOL+FACCERR
    BEQ     WaitEnd
Problem:
    BRA BadArray
WaitEnd:
    LDA     FSTAT
    AND     #FCCF
    BEQ     WaitEnd
    LDA     FSTAT
    AND     #FCBEF
    BEQ     WaitEnd

        LDA     FDest+1
        ADD     #2
        BCS     ExitOperate
        STA     FDest+1
        LDHX    FDest+1
MaxAddress:
        CPHX    #$1234
        BHI     ExitOperate
        BRA     Operate
ExitOperate:
    RTS
```

The applet quits when the operation status returns to the interface. The state encoding is given below:

- 0 — Operation failed (jumping to Wait).
- 1 — Operation passed (jumping to Done).
- 2 — Module is bad (module damaged/weird).
- 3 — No vpp (no programming voltage detected).
- 4 — Not possible in this Microcontroller configuration (protected mode, etc).
- 0xFF/-1 — Command not implemented (returned by the dispatcher when the command is not implemented).

The applet program ends when it encounters a background instruction encapsulated in an endless loop.

Stack workspace at the end of the applet code:

```
stack:      dc.w  0,0,0,0,0,0,0,0     ; stack
stackEnd:
```

3.6. PROGRAM Applet

The PROGRAM applet is used to program the individual byte/words from the module. There might be two different methods depending on the capabilities of the target interface or the Microcontroller. If the target memory can be read without disturbing the timing, the applet is not required to stop when a block is programmed. However, it may wait in a loop for the next block to be programmed. This matches the fast load mode (b0 bit of *flags* parameter) in the INFO applet.

3.6.1. Insertion in Flash Configuration File

The PROGRAM applet must define the following parameters in the params section: DelayTime, LoopCycles, and BaseCycles. They are used to parameterize the timing of the applet code (delay loops) and to define the start address of the data buffer. They have the same meaning as for CONTROL applet.

NOTE: All parameters of the PROGRAM applets except state are overwritten as soon as the PROGRAM applet is called. Therefore, initialization of these parameters is only indicative. However, all parameters evaluated by the INFO applet will overwrite the parameters of the PROGRAM applet. The state parameter must be initially set to $01. The interface writes $00 to state to indicate that the buffer is ready and to start programming.

3.6.2. PROGRAM Source File

The program applet programs a buffer in flash. A possible implementation is shown next.

```
BASE        EQU   $80            ; SRAM start
            ORG   BASE           ; Entry point
```

where, base is the first SRAM address and is similar to the one specified in the Flash Configuration file.

bStart is the pointer to the programming buffer to be copied. The offset of + $100 from base matches exactly the program buffer offset computed by the Flash Programmer. In the RAM/SRAM buffer, the parameters are placed first, then the code, and finally the buffer that will be programmed.

```
            dc.b  0
fStart:     dc.w  $c000
            dc.b  0
bSize:      dc.w  7
ClockSpeed: dc.w  1500
bufAdr:     dc.w  $400
flags:      dc.b  $01
state:      dc.b  $00
```

As explained above, these parameters are overwritten. The `state` parameter must be set to `$01`. The size of `fStart` and `bSize` variables is 2-, 3-, or 4-bytes as defined by the `ADR_SIZE` global parameter.

For *flags*, only the least semnificative bit is used. It indicates that the fast programming mechanism is used. After a block is programmed, the applet will not stop but wait for the next block to be programmed. All further bits are unused and should be set to zero.

NOTE: The least semnificative bit is set by the interface. It estimates according to the `b0` bit in *flags* in the `INFO` applet and to the target interface itself (not all interfaces support fast programming) if fast programming is possible.

If fast programming is not possible on the Microcontroller (for example, no background reading/writing), clear the `b0` bit in *flags* in the `INFO` applet. The programming applet must stop on a `BGND` instruction loop.
If fast programming is enabled, the programming applet waits till `state != $00` to start programming.

Starting of the program:

```
stack:      dc.w  0,0,0     ; stack
stackEnd:
```

The stack pointer is loaded. The stack space is initialized at the end of the assembler code. For more information, refer to the `stackEnd` label below.

The program waits for state to be null to start programming.

```
WaitForProgram:
    LDA     state           ; wait until state is 0 to start
    BNE     WaitForProgram
```

After the programming code, different exits are possible when the first buffer programming has been performed. The applet quits on the operation status return to the interface. The state encoding is given below:

- 1 — Operation passed (jumping to Done).
- 2 — Module is bad (module damaged/weird) (jumping to BadArray).

- 3 — No vpp (no programming voltage detected) (jumping to NoVfp).
- 4 — Not possible in this Microcontroller configuration (protected mode, etc) (jumping to BadConf).

After returning the state, if fast programming is forced by the interface, the program jumps to the entry code of the applet to start programming again.
Otherwise, the applet program ends when it encounters a background instruction encapsulated in an endless loop.

The program code is

```
Program:
PLoop:
    LDHX    bSize
    AIX     #-1
    STHX    bSize
    CPHX    #$FFFF
    BEQ     Done
WaitTillEmpty:
    LDA     FSTAT
    AND     #FCCF
    BEQ     WaitTillEmpty
    LDA     FSTAT
    AND     #FCBEF
    BEQ     WaitTillEmpty

    LDA     #FPVIOL+FACCERR
    STA     FSTAT
bStartPatch:
    LDA     $0101
fStartPatch:
    STA     $C000
    LDA     #PROG
    STA     FCMD
    LDA     #FCBEF
    STA FSTAT
    LDA     FSTAT
    AND     #FPVIOL+FACCERR
    BNE     BadArray
    LDHX    bStartPatch+1
    AIX #1
    STHX    bStartPatch+1
    LDHX    fStartPatch+1
    AIX #1
    STHX    fStartPatch+1
    BRA     PLoop
```

The algorithm loops through the buffer and programs the data at requested address.

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: AN3967

21 September 2011