Freescale Semiconductor
Application Note

Document Number:AN4570
Rev. 0, 08/2012

Understanding S08P Flash and EEPROM

by: William Jiang
Automotive and Industrial Solutions Group

Contents

1 Introduction to Flash and EEPROM

The S08P family introduces new Flash and EEPROM with automotive proven technology that is different from old S08 devices.

This application note presents an introduction of this new module, and then details on how to use this module with demo code provided. The demo code is developed with CodeWarrior for MCU v10.2.

The flash memory is ideal for single-supply applications allowing for field reprogramming without requiring external high voltage sources for program or erase operations. The flash is commonly used to store both code and data, while EEPROM typically for storing data, although it can be used to store code.

The flash and EEPROM operations are command based. It supports simultaneous flash and EEPROM operations. That is, code or data can be read from flash memory while an allowed EEPROM command is executing on EEPROM.

The EEPROM memory is implemented with Error Correction Codes (ECC) that can resolve single-bit faults and detect double-bit faults, providing the best safety feature among all three families.

Erase operation is performed on a sector base. Each flash sector contains 512 bytes. Each EEPROM sector consists of 2 bytes. The following table lists the device comparison on flash and EEPROM size for typical S08P devices:

Table 1. Device comparison on flash and EEPROM size

S08P devices	Flash size (bytes)	EEPROM size (bytes)	# of flash sectors	# of EEPROM sectors
S08PT60	~60 K	256	128	128
S08PT32	~32 K	256	64	128
S08PT16	~16 K	256	32	128
S08PT8	~8 K	256	16	128
S08PA4	~4 K	128	8	64
S08PA2	~2 K	128	4	64

There is NVM (Non-Volatile Memory) controller at the heart of flash and EEPROM module which is used to manage flash and EEPROM initialization as well as to handle the commands. As an example, the following figure shows flash and EEPROM block diagram on S08PT60:

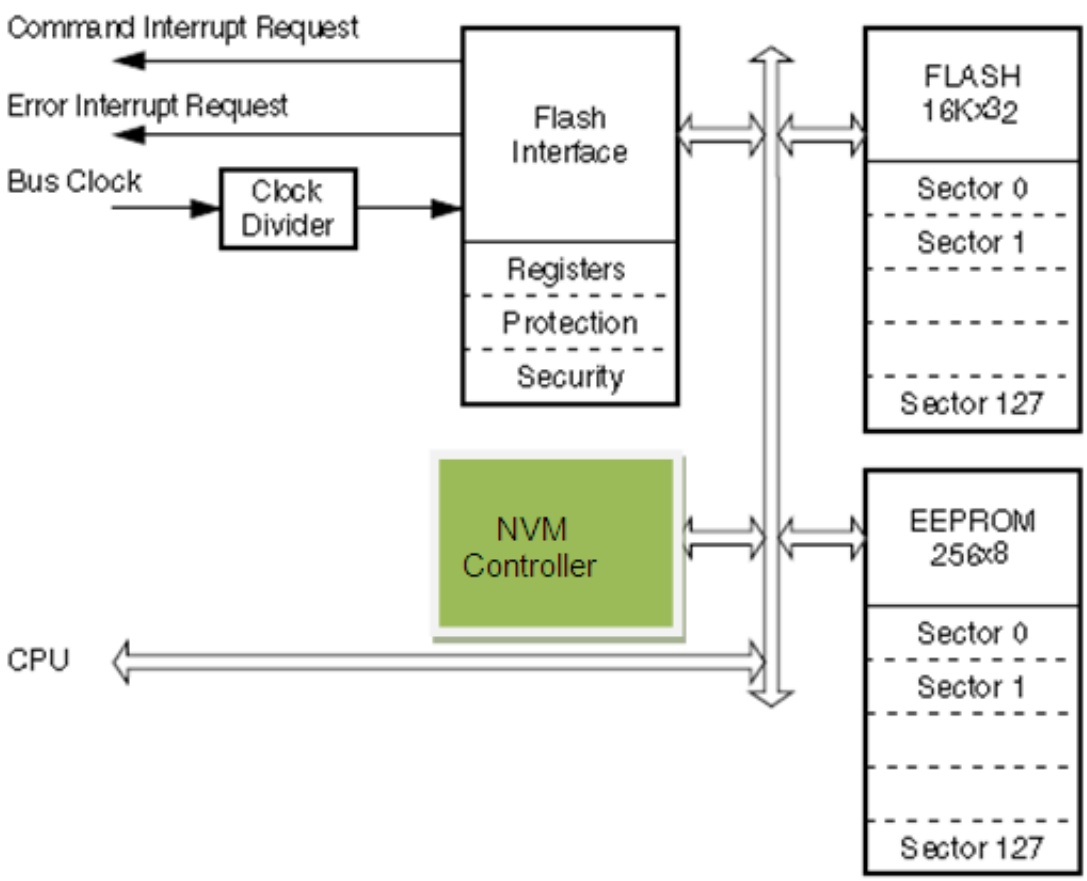

Figure 1. Flash and EEPROM block diagram

The flash memory has the following features:

- Automated program and erase algorithm with verification
- Fast sector erase and longword (32-bit) program operation
- Ability to read the flash memory while programming a byte in the EEPROM memory

- Flexible protection scheme to prevent accidental program or erase of flash memory
- Ability to ensure the flash read robustness by setting margin levels

The EEPROM memory has the following features:

- Single-bit fault correction and double-bit fault detection within a word during read operations
- Automated program and erase algorithm with verification and generation of ECC parity bits
- Fast sector erase and byte program operation
- Protection scheme to prevent accidental program or erase of EEPROM memory
- Ability to program up to four bytes in a burst sequence
- Ability to ensure the EEPROM read robustness by setting margin levels
- 500 K program/erase cycles

2 Use of flash and EEPROM

Since flash and EEPROM operations are command based, the command format should be known. The command consists of command code and command parameters that are stored in indexed 16-bit Flash Common Command Object (NVM_FCCOB) registers. There are six NVM_FCCOB registers mapped to just one global memory address. A flash Common Command Object Index register (NVM_FCCOBIX) is used to specify which NVM_FCCOB is currently used.

The NVM_FCCOB register can be divided into high byte (NVM_FCCOBHI) and low byte (NVM_FCCOBLO).

The command code is used to differentiate the command to be executed on flash/EEPROM by NVM controller. The command parameters may include global memory address, data, and any required information.

The memory controller must complete the execution of a command before the FCCOB register can be written to with a new command. When the flash command is completed and/or the flash error is detected, the memory controller will generate the corresponding interrupt if enabled.

The following table shows the general command parameter format in NVM_FCCOB and the corresponding parameter index in NVM_FCCOBIX:

Table 2. General command parameter format in FCCOB

NVM_FCCOBIX[2:0]	**FCCOB register meaning** (the least significant bits are at the right side and most significant bits must be filled 0 if not used)	
	High byte	Low byte
	NVM_FCCOBHI	**NVM_FCCOBLO**
000	Flash command code(FCMD)	memory address bits[23: 16]
001	memory address bits[15 : 8]	memory address bits[7 : 0]
010	Data0 bits[15:8]	Data0 bits[7:0]
011	Data1 bits[15:8]	Data1 bits[7:0]
100	Data2 bits[15:8]	Data2 bits[7:0]
101	Data3 bits[15:8]	Data3 bits[7:0]

The memory address specified in FCCOB as a command parameter is the global address of the flash and EEPROM in the device memory map. Some commands such as Erase Verify Block, Erase Flash Block, and Set User Margin Level use bit 23 of the memory address parameter to identify which NVM block is to be operated with the given command. If it is 0, then flash block is selected to operate on and if it is 1, then EEPROM block is selected to operate on.

Before command write sequence, NVM_FCLKDIV should be written to configure the flash clock to be between 0.8 and 1 MHz.

The command write sequence contains four steps:

1. Wait for the previous command to be completed, that is, NVM_FSTAT[CCIF] == 1.
2. Check NVM_FSTAT[ACCERR] and NVM_FSTAT[FPVIOL] bits and clear these bits if they are set by writing 0x30 to NVM_FSTAT.
3. Write the desired flash/EEPROM command and all the related parameters to NVM_FCCOB indexed by NVM_FCCOBIX[2:0] (the recommended write sequence is in assending order of the command parameter format).
4. Launch the command by writing 0x80 to NVM_FSTAT to clear NVM_FSTAT[CCIF] flag.

Some specific commands are not allowed to operate on EEPROM while some commands are running on flash block. For example, EEPROM read operation is not allowed while any flash command is running on the flash block. Another example is that it is not allowed to run code from flash to mass erase (erase block or erase all commands) EEPROM. In this case, the code must be running from RAM. A technique is introduced later to demonstrate how to copy code from flash to RAM and run from there. For additional information, refer to Table 4-19 of MC9S08PT60 Reference Manual available in http://www.freescale.com that shows simultaneous flash and EEPROM operations.

There are 13 commands for flash operation and 10 commands for EEPROM operation which can be divided into command types: erase, program, erase verify, unsecure, read/program once, and set margin.

The following sections describe the usage of typical commands. For others, refer to MC9S08PT60 Reference Manual.

2.1 Copy code from flash to RAM

There are some techniques to copy code from flash to RAM. For example, using stack to store and run code and global RAM to store and run code. This section describes the later technique.

The first step is to define a code section to include all required code by using #pragma CODE_SEG compiler directive. For example, the following code snippet defines a FLASH_ROUTINES section that includes two routines:

```
#pragma CODE_SEG FLASH_ROUTINES
uint16_t Flash_Program1LongWord(uint16_t wNVMTargetAddress, uint32_t dwData)
{
…
}
uint16_t Flash_EraseSector(uint16_t wNVMTargetAddress)
{
…
}
#pragma CODE_SEG DEFAULT
```

The next step is to allocate memory space (that is, memory segment) for this new code section and then place this section into the memory segment in the corresponding linker file. The following code snippet does this for FLASH_ROUTINES section:

```
SEGMENTS
        …
    FLASH_TO_RAM    =  READ_ONLY  0xF600 TO 0xFBFF RELOCATE_TO 0x0B00;
END

PLACEMENT
…
FLASH_ROUTINESINTO FLASH_TO_RAM;
```

Please note the above keyword RELOCATE_TO is a relocation command used to direct the linker to generate code in runtime RAM addresses starting from 0x0B00 but places code in flash.

The last step is to actually copy code from flash to RAM and hence implement the relocation of code from flash to RAM. The following code Flash_CopyInRAM is used to copy FLASH_ROUTINES to RAM:

```
/*
Start_Copy_In_RAM refers to the beginning of the segment FLASH_ROUTINES.
This segment contains the functions after they have been copied to RAM.
*/
#define Start_Copy_In_RAM __SEG_START_FLASH_ROUTINES
#define Size_Copy_In_RAM __SEG_SIZE_FLASH_ROUTINES
```

```
voidFlash_CopyInRAM(void)
{
  char *srcPtr, *dstPtr;
  uint16_t count;
  uint16_t sizeBytes = (uint16_t)Size_Copy_In_RAM;
  srcPtr = (char*)Start_Copy_In_RAM;
  dstPtr = (char*)&Flash_Program1LongWord;// must be the start address of RAM_CODE
  for (count = 0; count < sizeBytes;  count++)
  {
    *dstPtr++ = *srcPtr++;
  }
}
```

2.2 Erase flash/EEPROM

There are four erase commands: Erase flash sector command, Erase flash block command, Erase all blocks command, and Erase EEPROM sector command.

Erase flash sector command will erase all memory units in the specified flash sector, while Erase flash block command will bulk erase the entire flash memory space. Erase all blocks command will bulk erase the entire flash memory space as well as the whole EEPROM memory space.

Because of simultaneous flash and EEPROM operations, Erase flash block command and Erase all blocks command must run in RAM.

The following code snippet shows how to erase a flash sector at address *wNVMTargetAddress*:

```
uint16_t Flash_EraseSector(uint16_t wNVMTargetAddress)
{
uint16_t err = FLASH_ERR_SUCCESS;

// Check address to see if it is aligned to 4 bytes
// Global address [1:0] must be 00.
if(wNVMTargetAddress & 0x03)
{
err = FLASH_ERR_INVALID_PARAM;
return (err);
}
// Clear error flags
NVM_FSTAT = 0x30;

// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[23:16]
NVM_FCCOBHI = FLASH_CMD_ERASE_SECTOR;// EEPROM FLASH command
NVM_FCCOBLO = 0;// memory address bits[23:16]
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX = 0x1;
// Write the lower byte memory address bits[15:0]
NVM_FCCOB = wNVMTargetAddress;

// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Check error status
if(NVM_FSTAT & NVM_FSTAT_ACCERR_MASK)
{
err |= FLASH_ERR_ACCESS;
}
…
```

```
return (err);
}
```

The following code snippet shows how to erase a EEPROM sector at address

```
wNVMTargetAddress:
uint16_t EEPROM_EraseSector(uint16_t wNVMTargetAddress)
{
uint16_t err = FLASH_ERR_SUCCESS;

// Clear error flags
NVM_FSTAT = 0x30;

// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[23:16]
NVM_FCCOBHI = EEPROM_CMD_ERASE_SECTOR;// EEPROM FLASH command
NVM_FCCOBLO = 0;// memory address bits[23:16]
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX = 0x1;
// Write the lower byte memory address bits[15:0]
NVM_FCCOB = wNVMTargetAddress;

// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Check error status
if(NVM_FSTAT & NVM_FSTAT_ACCERR_MASK)
{
err |= FLASH_ERR_ACCESS;
}
…
return (err);
}
```

The following code snippet shows how to erase a flash block or EEPROM block at address *wNVMTargetAddress*:

```
uint16_t NVM_EraseBlock(uint16_t wNVMTargetAddress, uint8_t bIsEEPROM)
{
uint16_t err = FLASH_ERR_SUCCESS;

// Clear error flags
NVM_FSTAT = 0x30;

// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[23:16]
NVM_FCCOBHI = NVM_CMD_ERASE_BLOCK;// erase FLASH block command
NVM_FCCOBLO = 0;// memory address bits[23:16] with bit23 = 0 for Flash block, 1 for EEPROM
block
if(bIsEEPROM)
{
NVM_FCCOBLO |= 0x80;// bit 23 = 1 for EEPROM block
}
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX = 0x1;
// Write the lower byte memory address bits[15:0]
NVM_FCCOB = wNVMTargetAddress;

// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Check error status
if(NVM_FSTAT & NVM_FSTAT_ACCERR_MASK)
```

```
{
err |= FLASH_ERR_ACCESS;
}
…

return (err);
}
```

The following code snippet shows how to erase verify a flash block or EEPROM block at address *wNVMTargetAddress*:

```
uint16_t NVM_EraseVerifyBlock(uint16_t wNVMTargetAddress, uint8_t bIsEEPROM)
{
uint16_t err = FLASH_ERR_SUCCESS;

// Clear error flags
NVM_FSTAT = 0x30;

// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[23:16]
NVM_FCCOBHI = NVM_CMD_ERASE_VERIFY_BLOCK;// erase FLASH block command
NVM_FCCOBLO = 0;// memory address bits[23:16] with bit23 = 0 for Flash block, 1 for EEPROM
block
if(bIsEEPROM)
{
NVM_FCCOBLO |= 0x80;// bit 23 = 1 for EEPROM block
}
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX = 0x1;
// Write the lower byte memory address bits[15:0]
NVM_FCCOB = wNVMTargetAddress;

// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Check error status
if(NVM_FSTAT & NVM_FSTAT_ACCERR_MASK)
{
err |= FLASH_ERR_ACCESS;
}
…

return (err);
}
```

The following code snippet shows how to erase verify all blocks including both flash and EEPROM:

```
uint16_t NVM_EraseVerifyAll(void)
{
uint16_t err = FLASH_ERR_SUCCESS;

// Clear error flags
NVM_FSTAT = 0x30;

// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[23:16]
NVM_FCCOBHI = NVM_CMD_ERASE_VERIFY_ALL;// erase verify all flash & EEPROM blocks

// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Check error status
…
```

```
return (err);
}
```

The following code snippet shows how to erase verify a EEPROM section with *uiByteCount* number of bytes at *wNVMTargetAddress*:

```
uint16_t EEPROM_EraseVerifySection(uint16_t wNVMTargetAddress, uint16_t uiByteCount)
{
uint16_t err = FLASH_ERR_SUCCESS;
// Check address to see if it is aligned to 4 bytes
// Global address [1:0] must be 00.
if(wNVMTargetAddress & 0x03)
{
err = FLASH_ERR_INVALID_PARAM;
return (err);
}
// Clear error flags
NVM_FSTAT = 0x30;

// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[23:16]
NVM_FCCOBHI = EEPROM_CMD_ERASE_VERIFY_SECTION;// erase verify FLASH section command
NVM_FCCOBLO = 0;// memory address bits[23:16] with bit23 = 0 for Flash block, 1 for EEPROM
block
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX = 0x1;
// Write the lower byte memory address bits[15:0]
NVM_FCCOB = wNVMTargetAddress;
// Write index to specify the # of longwords to be verified
NVM_FCCOBIX = 0x2;
// Write the # of longwords
NVM_FCCOB = uiByteCount;
// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Check error status
if(NVM_FSTAT & NVM_FSTAT_ACCERR_MASK)
{
err |= FLASH_ERR_ACCESS;
}
…

return (err);
}
```

2.3 Program flash/EEPROM

Program flash command will program up to two previously erased longwords in the flash memory using an embedded algorithm. The memory locations to be programmed must be a longword or two longwords and their addresses must be aligned to longword (4 bytes) boundary, that is, the lower two memory address bits[1:0] must be 0.

The following function Flash_Program1LongWord is used to program a longword in flash memory at wNVMTargetAddress with a longword dwData:

```
uint16_t Flash_Program1LongWord(uint16_t wNVMTargetAddress, uint32_t dwData)
{
uint16_t err = FLASH_ERR_SUCCESS;

// Check address to see if it is aligned to 4 bytes
// Global address [1:0] must be 00.
if(wNVMTargetAddress & 0x03)
```

```
{
err = FLASH_ERR_INVALID_PARAM;
return (err);
}
// Clear error flags
NVM_FSTAT = 0x30;

// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = FLASH_CMD_PROGRAM;// program FLASH command
NVM_FCCOBLO = 0;// memory address bits[17:16]
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX = 0x1;
// Write the lower byte memory address bits[15:0]
NVM_FCCOB = wNVMTargetAddress;
// Write index to specify the word0 (MSB word) to be programmed
NVM_FCCOBIX = 0x2;
// Write the word  0
NVM_FCCOB = (dwData>>16) & 0xFFFF;
// Write index to specify the word1 (LSB word) to be programmed
NVM_FCCOBIX = 0x3;
// Write the word1
NVM_FCCOB = (dwData) & 0xFFFF;
// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Check error status
if(NVM_FSTAT & NVM_FSTAT_ACCERR_MASK)
{
err |= FLASH_ERR_ACCESS;
}
…

return (err);
}
```

The following function Flash_Program2LongWord is used to program two longwords in flash memory at wNVMTargetAddress with dwData0 and dwData1:

```
uint16_t Flash_Program2LongWords(uint16_t wNVMTargetAddress, uint32_t dwData0, uint32_t
dwData1)
{
uint16_t err = FLASH_ERR_SUCCESS;

// Check address to see if it is aligned to 4 bytes
// Global address [1:0] must be 00.
if(wNVMTargetAddress & 0x03)
{
err = FLASH_ERR_INVALID_PARAM;
return (err);
}
// Clear error flags
NVM_FSTAT = 0x30;

// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = FLASH_CMD_PROGRAM;// program FLASH command
NVM_FCCOBLO = 0;// memory address bits[17:16]
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX = 0x1;
// Write the lower byte memory address bits[15:0]
NVM_FCCOB = wNVMTargetAddress;
// Write index to specify the word0 (MSB word) to be programmed
NVM_FCCOBIX = 0x2;
// Write the word 0
```

```
NVM_FCCOB = (dwData0>>16) & 0xFFFF;
// Write index to specify the word1 (LSB word) to be programmed
NVM_FCCOBIX = 0x3;
// Write word 1
NVM_FCCOB = (dwData0) & 0xFFFF;

// Write index to specify the word0 (MSB word) to be programmed
NVM_FCCOBIX = 0x4;
// Write the word2
NVM_FCCOB = (dwData1>>16) & 0xFFFF;
// Write index to specify the word1 (LSB word) to be programmed
NVM_FCCOBIX = 0x5;
// Write word 3
NVM_FCCOB = (dwData1) & 0xFFFF;
// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Check error status
if(NVM_FSTAT & NVM_FSTAT_ACCERR_MASK)
{
err |= FLASH_ERR_ACCESS;
}
…

return (err);
}
```

The following code *EEPROM_Program1Byte* shows how to program a byte bData at address *wNVMTargetAddress* in EEPROM memory:

```
uint16_t EEPROM_Program1Byte(uint16_t wNVMTargetAddress, uint8_t bData)
{
uint16_t err = FLASH_ERR_SUCCESS;

// Clear error flags
NVM_FSTAT = 0x30;

// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = EEPROM_CMD_PROGRAM;// EEPROM FLASH command
NVM_FCCOBLO = 0;// memory address bits[17:16]
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX = 0x1;
// Write the lower byte memory address bits[15:0]
NVM_FCCOB = wNVMTargetAddress;
// Write index to specify the byte0 (MSB word) to be programmed
NVM_FCCOBIX = 0x2;
// Write the byte 0
NVM_FCCOB = bData;
// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Check error status
if(NVM_FSTAT & NVM_FSTAT_ACCERR_MASK)
{
err |= FLASH_ERR_ACCESS;
}
…

return (err);
}
```

The following code EEPROM_ProgramUpto4Bytes shows how to program up to 4 bytes at address wNVMTargetAddress in EEPROM memory where 4 bytes data is stored in the array pointed to by pbData :

```
uint16_t EEPROM_ProgramUpto4Bytes(uint16_t wNVMTargetAddress, uint8_t *pbData, uint8_t
bByteCount)
{
int i;
uint16_t err = FLASH_ERR_SUCCESS;
if(bByteCount >4 || bByteCount == 0)
{
err = FLASH_ERR_INVALID_PARAM;
return (err);
}
// Clear error flags
NVM_FSTAT = 0x30;

// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = EEPROM_CMD_PROGRAM;// EEPROM FLASH command
NVM_FCCOBLO = 0;// memory address bits[17:16]
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX = 0x1;
// Write the lower byte memory address bits[15:0]
NVM_FCCOB = wNVMTargetAddress;

for (i = 0; i < bByteCount; i++)
{
// Write index to specify the byte0 (MSB word) to be programmed
NVM_FCCOBIX = 0x2+i;
// Write the byte 0
NVM_FCCOB = *pbData++;
}
// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Check error status
if(NVM_FSTAT & NVM_FSTAT_ACCERR_MASK)
{
err |= FLASH_ERR_ACCESS;
}
…
return (err);
}
```

2.4 Program Once and Read Once

The Program Once and Read Once commands are used to access a reserved 64-byte (8 phrases) field in the nonvolatile information register (IFR) in the flash block. This reserved field can not be erased and can be used to store the product ID or any other information that can be written only once. So, the Program Once command can be used only once.

The following code snippet shows how to program and read the reserved 64 bytes field of IFR:

```
uint16_t Flash_ProgramOnce(uint8_t bPhraseIndex, uint8_t *pData8Bytes)
{
uint16_t err = FLASH_ERR_SUCCESS;
int i;

// Check the phrase index to if it is out of boundary
if(bPhraseIndex > 7)
{
err = FLASH_ERR_INVALID_PARAM;
return (err);
}
// Clear error flags
NVM_FSTAT = 0x30;
```

```
// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = FLASH_CMD_PROGRAMONCE;// command
NVM_FCCOBLO = 0;// memory address bits[17:16]
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX = 0x1;
// Write the phrase index
NVM_FCCOB = bPhraseIndex;

// Write 4 words
for(i = 0; i < 4; i++)
{
// Write index to specify the word (MSB word) to be programmed
NVM_FCCOBIX = 0x2+i;
// Write the word 0
NVM_FCCOB = ((uint16_t)pData8Bytes[(i<<1)+1]<<8) | pData8Bytes[(i<<1)];
}
// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Check error status
…

return (err);
}

uint16_t Flash_ReadOnce(uint8_t bPhraseIndex, uint8_t *pData8Bytes)
{
uint16_t err = FLASH_ERR_SUCCESS;
int i;

// Check the phrase index to if it is out of boundary
if(bPhraseIndex > 7)
{
err = FLASH_ERR_INVALID_PARAM;
return (err);
}
// Clear error flags
NVM_FSTAT = 0x30;

// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = FLASH_CMD_READONCE;// command
NVM_FCCOBLO = 0;// memory address bits[17:16]
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX = 0x1;
// Write the phrase index
NVM_FCCOB = bPhraseIndex;
// Launch the command
NVM_FSTAT = 0x80;
// Wait till command is completed
while (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));
// Read 4 words
for(i = 0; i < 4; i++)
{
// Read the word (MSB word) indexed by FCCOBIX
NVM_FCCOBIX = 0x2+i;
pData8Bytes[i<<1] = NVM_FCCOBLO;
pData8Bytes[(i<<1)+1] = NVM_FCCOBHI;
}
// Check error status

…

return (err);
}
```

2.5 User margin level

The user margin is a small delta to the normal read reference level and, in effect, is a minimum safety margin. If the reads pass at the tighter tolerances of the user margins, then the normal reads have at least some safety margin before users experience data loss. There are two user margin levels: user margin-1 level and user margin-0 level. User margin-1 level is the read margin to the erased state, while user margin-0 level is the read margin to the programmed state.

Users can use the Set User Margin level command to set the related user margin for future read operation from flash/ EEPROM. If user margin-1 level is set, and the future flash/EEPROM read returns a single-bit value of zero, then the flash read robustness is not guaranteed. If user margin-0 level is set, and the future flash/EEPROM read returns a bit of 1, then the flash read robustness is also not guaranteed. This feature helps to ensure flash memory read robustness, and users can take the necessary action for safety purposes.

The NVM_FCCOBLO[7] bit is used to select between flash memory and EEPROM memory to be applied with the set user margin level: 0 – flash memory to be applied; 1 – EEPROM memory to be applied.

The following code snippet shows how to set the user margin-0 level for EEPROM block at address:

```
/*  Set user margin-0 level for the future flash operations
*/
// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = 0x0D;// set user margin level command
NVM_FCCOBLO = 0x80;// Bit 7 = 1  selects EEPROM memory
// Write index to specify EEPROM address
NVM_FCCOBIX  = 0x1;
// Write the EEPROM address
NVM_FCCOB = 0x3100;//
// Write index to specify user margin level
NVM_FCCOBIX  = 0x2;
// Write the user margin-0 level
NVM_FCCOB = 0x0002;//
// Launch the command
NVM_FSTAT = 0x80;

// Wait till command is completed
While (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));
```

2.6 Secure and unsecure the device

To secure S08P device in the application code, program security byte of the flash configuration field at 0xFF7F (NV_FSEC) with bit 1 and 0 should not be equal to 0b10. It is recommended to use modifier "const" when defining the security byte variable with the initialized security value. The following code line is used to enable backdoor key access and security of the device:

```
const byte flash_security_byte @0xFF7F = 0xBC;
```

To unsecure S08P device in the application code, program security byte NV_FSEC with bit 1 and 0 should be equal to 0b10. After reset, the device is in unsecure state. To program security bits in NV_FSEC to 0b10, the application code should check whether the security bits are 0b11 or not. If they are 0b11, just use Program flash command. Otherwise, the whole sector from 0xFE00, which is also the last flash sector, must be erased. Before erasing this last sector, the interrupt vector table and the other locations not to be changed must be saved to RAM. After erasing the sector, change the contents at the required locations in RAM and then program the whole sector with the saved contents and changed values. The following code snippet shows how to do this:

```
// Save the last flash sector
for(i = 0; i < FLASH_SECTOR_SIZE; i++)
{
bSectorSave[i] = pSector[i];
}
wNVMTargetAddres = 0xFE00;
wOffset2SecurityByte = (uint16_t)pSecurityByteInFlashConfigField -
            (uint16_t)pSector;

if(((bSectorSave[wOffset2SecurityByte])& 0x03) != 0x02u)
{
// flash is secured
if(((bSectorSave[wOffset2SecurityByte])& 0x03) != 0x3)
{
// This flash sector needs to be erased
// Now sector erase this area
err = Flash_EraseSector(wNVMTargetAddres);
if(err)
{
printf("\nErase flash sector failed!\n");
return;
}
}
// Change security bits to 0b10
bSectorSave[wOffset2SecurityByte] &=  0xfe;

// now program flash last sector
err =  Flash_Program(wNVMTargetAddres, bSectorSave, FLASH_SECTOR_SIZE);
if(err)
{
printf("\nProgram flash failed!\n");
}
else
{
printf("\nDevice is unsecured,please restart to take effect!\n");
}
}
```

To temporarily unsecure the device, enable backdoor key access by programming NV_FSEC bit 7 and 6 to 0b10 as well as eight backdoor access keys from 0xFF70 (NV_BACKKEY0) to 0xFF77(NV_BACKKEY7). Then reset the device followed by issuing Verify backdoor access key command with the correct 8 backdoor access keys. After successfully executing this command, the device is put in unsecured state. If any reset occurs afterward, then the device is put in the state indicated by NV_FSEC bit 1 and 0.

Using backdoor key access the eight keys can be provided through any serial communication interface such as SCI and SPI. The BDM can not be used as a means to provide the backdoor access keys. However, it can be used to unsecure the device by following the procedure outlined in Section 4.4.2.7.2 Unsecuring the MCU using BDM, MC9S08PT60/32 Reference Manual.

2.7 ECC parity check

EEPROM supports ECC parity check. It supports single fault bit and double fault bits detection and also resolves single fault bit. This feature provides additional safety mechanism for the system.

Both errors are reflected in NVM_FERSTAT register. NVM_FERSTAT[DFDIF] indicates a double fault bit error has occurred, while NVM_FERSTAT[SFDIF] indicates a single fault bit error has occurred if NVM_FCNFG[IGNSF] is cleared.

Both ECC errors can generate the corresponding interrupt if the corresponding bit in NVM_FERCNFG register is set.

There is a one cycle delay in storing the ECC DFDIF] and SFDIF fault flags in NVM_FERSTAT register. At least one clock cycle is required after a flash memory read before checking NVM_FERSTAT for the occurrence of ECC errors.

2.8 Protection

Flash protection is used to protect regions in the flash memory from accidental programming or erasing. If a location is protected, the program and erase commands can not be executed. In this case, the protection violation error will occur and the FPVIOL bit will be set in the FSTAT register.

Flash protection is performed on three separate memory regions in the flash memory on S08PT60/32 and their phantom devices: one growing upward from global address 0x8000 (called low range), one growing downward from global address 0xFFFF (high range), and the remaining memory region.

There are two protection fixed ends: one at 0xFFFF in high range and the other is at 0x8000 in low range. The protection region grows up and/or down from the protection fixed end by protected size in K bytes for the given range.

The protected address region can only be added and can not be reduced once it is implemented.

For S08PT16/8, S08PA4/2 and their phantom devices, two separate memory regions, one growing downward from global address 0xFFFF in the flash memory (the high region), and the remaining addresses in the flash memory can be activated for protection.

The higher address region is mainly targeted to hold the bootloader code because it covers the interrupt vector table.

The protected size varies from different families. For S08PT60/32 devices, the protected size in KB for high range is 2, 4, 8, 16; for low range it is 1, 2, 4, 8. For S08PT16/8 devices, the protected size in KB is 2, 4, 8, 16. For S08PA4/2 devices, the protected size in KB is 1, 2, 3, 4.

Flash protection and unprotection are controlled via NVM_FPROT register bits: FPOPEN, FPHDIS, and FPLDIS (only applicable for S08PT60/32).

The NVM_FPROT register bits are writable with the restriction that the size of the protected region can only be increased.

EEPROM protection is controlled by NVM_EEPROT register bits: DPOPEN and DPS. The protected region can be increased, but can not be reduced. The NVM_EEPROT[DPOPEN] bit enables or disables the EEPROM protection, while NVM_EEPROT[DPS2-0] bits define the protected size which is 32, 64, 96, 128, 160, 192, 224, or 256 bytes. These register bits are writable with the restriction that the size of the protected region can only be increased. Writes must increase the DPS value and the DPOPEN bit can only be written from 1 (protection disabled) to 0 (protection enabled).

3 Conclusion

The Flash and EEPROM module on S08P has many new features as outlined in Introduction to Flash and EEPROM. Use of this module requires deep understanding of the flash command functions and their formats. The most common commands include erase flash/EEPROM and program flash/EEPROM. With the example code snippet and routines provided in this note, it is easier to understand this module and quick start with the flash operations in an application code.

4 References

1. MC9S08PT60/32 Reference Manual, available at http://www.freecsale.com
2. MC9S08PT16/8 Reference Manual, available at http://www.freecsale.com
3. MC9S08PA4/2 Reference Manual, available at http://www.freecsale.com
4. AN4347,Transitioning Applications from S08AC and S08FL Family to S08PT Family, available at http://www.freecsale.com

5 Glossary

Table 3. Abbreviations used in this document

BDM	Background Debug Mode
ECC	Error Correction Codes
FCCOB	Flash Common Command Object

How to Reach Us:

Home Page:
www.freescale.com

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

USA/Europe or Locations Not Listed:
Freescale Semiconductor
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

Document Number: AN4570
Rev. 0, 08/2012