Freescale Semiconductor
Application Note

Document Number: AN4838
Rev. 0, 12/2013

How to Use Bit-band and BME on the KE04 and KE06 Subfamilies

by *Cheng Yangtao*

Asia Pacific Microcontroller Solutions Group

1 Introduction

As the bit-band region is optional on the ARM Cortex-M0+ processor, Freescale has implemented an Upper SRAM (SRAM_U) bit-band region on the KE04 and KE06 subfamily devices. The bit-band region operation is like that of the ARM Cortex-M3 and ARM Cortex-M4 processors. It maps a complete word of memory onto a single bit in the bit-band region. For example, writing to one of the alias words sets or clears the corresponding bit in the bit-band region. It enables individual bits to be toggled without performing a read/modify/write sequence.

The Bit Manipulation Engine (BME) provides hardware operations for an atomic read/modify/write peripheral and SRAM_U address space. The atomic read/modify/write operation is an indivisible "read followed by a write" bus sequence. The BME hardware microarchitecture is a 2-stage pipeline design that matches the protocol of the AMBA-AHB system bus interfaces. By combining the basic load and store instruction support in the ARM Cortex-M0+ instruction set architecture with the concept of decorated storage provided by the BME. BME can implement robustness and efficient read-modify-write capability on KE series Cortex-M0+ core microcontrollers.

Contents

2 Bit-Band

The following mapping formula demonstrates how to match each word in the alias region to a corresponding bit or target bit in the bit-band region.

bit_word_offset = (byte_offset x 32) + (bit_number × 4)

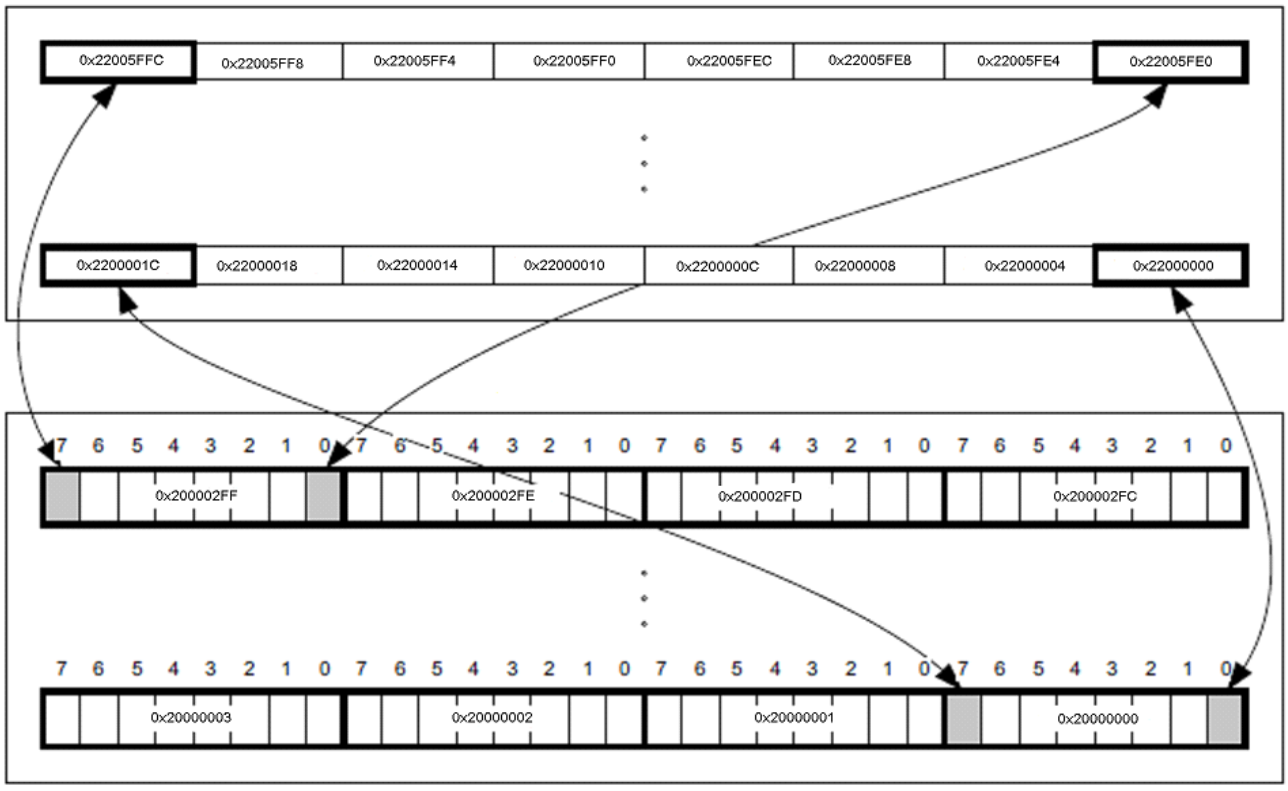
bit_word_addr = bit_band_base + bit_word_offset

Where:

- bit_word_offset is the position of the target bit in the bit-band memory region.
- bit_word_addr is the address of the word in the alias memory region that maps to the targeted bit.
- bit_band_base is the starting address of the alias region.
- byte_offset is the number of the byte in the bit-band region that contains the targeted bit.
- bit_number is the bit position, 0 to 7, of the targeted bit.

For example, the alias word at 0x22000000 maps to bit [0] of the bit-band byte at 0x20000000: 0x22000000 = 0x22000000 + (0*32) + 0*4.

Figure 1. KE04 bit-band mapping



2.1 Bit-Band Operation

Writing to a word in the alias region has the same effect as a read/modify/write operation on the targeted bit in the bit-band region.

Bit [0] of the value written to a word in the alias region determines the value written to the targeted bit in the bit-band region. Writing a value with bit [0] set writes a 1 to the bit-band bit, and writing a value with bit [0] cleared writes a 0 to the bit-band bit.

Reading a word in the alias region returns either 0x01 or 0x00. A value of 0x01 indicates that the targeted bit in the bit-band region is set. A value of 0x00 indicates that the targeted bit is clear.

2.2 Bit-band library

The user can define simple bit-band access with a macro such as:

#define Bit_Band_Set(Addr, Bit) {*(volatile uint32_t *)(0x22000000 + (((uint32_t)Addr)&0x3FF)*32 + ((uint8_t)Bit)*4) = 0x01; } /*set the bit*/

#define Bit_Band_Clear(Addr, Bit) {*(volatile uint32_t *)(0x22000000 + (((uint32_t)Addr)&0x3FF)*32 + ((uint8_t)Bit)*4) = 0; } /*clear the bit*/

For example, if the user wants to set the 0x20000000 bit1, the C code is:

Bit_Band_Set(0x20000000, 1) ;

There is a bit-band library (bitband.h) in the attached the code in which the user can refer to the bit-band demo code.

3 BME Introduction

BME decorated references are only available on system bus transactions generated by the processor core and targeted at the standard 512 KB peripheral address space at 0x40000000~0x4007FFFF and SRAM_U space based at 0x20000000. The decoration semantic is embedded into address bits[28:19], creating a 448 MB space at addresses 0x44000000~0x5FFFFFFF for AIPS and a 448 MB space at addresses 0x24000000~0x3FFFFFFF for SRAM_U. These bits are stripped out of the actual address sent to the peripheral bus controller and used by the BME to define and control its operation.

BME support decorated loads and decorated stores operation. The decorated loads including unsigned bit field extract (UBFX), load-and-clear 1 bit (LAC1), and load-and-set 1 bit (LAS1) operations. The decorated stores including AND, OR, XOR and bit field insert (BFI) operations.

Table 1. Base Address and BME operations

Modules	Base Address	Decorated Address Space	Decorated Stores				Decorated Load		
—	—	—	AND	OR	XOR	BVFI	LAC1	LAS1	UBFX
SRAM_U	0x20000000	0x24000000-0x3FFFFFFF	Y	Y	Y	Y	Y	Y	Y
Peripherals	0x4000F000	0x44000000-0x4FFFFFFF	Y	Y	Y	Y	Y	Y	Y

Table 1. Base Address and BME operations (continued)

Modules	Base Address	Decorated Address Space	Decorated Stores				Decorated Load		
GPIO	0x4000FF000	0x440000000-0x4FFFFFFF	Y	Y	y	N	Y	Y	N
	0x4000F0000	0x50000000-0x5FFFFFFF	Y	Y	Y	Y	Y	Y	Y

Note: 0x4000F000 is the base address of GPIO controller and is aliased to 0x400FF000.
Note: Y indicates that this operation is feasible.
Note: N indicates that this operation is infeasible.

The user must write or read target data from the decorated address. Each operation has a fixed style. The user must make up the correct 32-bit decorated address as shown below.

Operations	Decorated addrress													
	Bit[31]	Bit[30:29]	Bit[28]	Bit[27:26]	Bit[25:24]	Bit[23]	Bit[22:21]	Bit[20]	Bit[19]	Bit[18:16]	Bit[15:12]	Bit[11:8]	Bit[7:4]	Bit[3:0]
AND	0	addr[30:29]	0	01	--	-	--	-	addr[19:0]					
OR	0	addr[30:29]	0	10	--	-	--	-	addr[19:0]					
XOR	0	addr[30:29]	0	11	--	-	--	-	addr[19:0]					
LAC1	0	addr[30:29]	0	10	b			-	addr[19:0]					
LAS1	0	addr[30:29]	0	11	b			-	addr[19:0]					
BFI	0	addr[30:29]	1	b			w		addr[18:0]					
UBFX	0	addr[30:29]	0	b			w		addr[18:0]					

Figure 2. Decorated address make up

Note: addr[19:0] and addr[18:0] are peripheral adress or SRAM_U address.
Note: b is LSB position. It indicates the operation will begin from this bit. For example there is 11101111 in 0x20000000, if the user wants to set the bit 4 from 0 to 1, the b should be 4.
Note: w is the bit field width minus 1 identifier. For example, if the bit filed is “1001”, the w should be 3.
Note: -bit can be 0 or 1, it is an insignificant bit.
Note: addr[30:29] is SRAM_U or peripheral option, addr[30:29]=01 is SRAM_U, addr[30:29]=10 is peripheral. Addr[30:29] is identical in decorated address and target address.

3.1 Decorated store operations

The decorated store includes three common logical operations: AND, OR , XOR, and a bit field insert. Each operation is a 2-cycle atomic read-modify-write sequence. The data size can be 8-, 16-, or 32-bits.

For the 16-bit write mode, bit[0] of SRAM_U or the peripheral address should be 0. For the 32-bit write mode, bit[1:0] of SRAM_U or the peripheral address should be 00. There is no similar limitation in the 8-bit write mode.

3.1.1 AND

AND command performs a logical AND operation. See the example in the following section.

3.1.1.1 Byte (8-bit) write mode

SRAM_U memory address is 0x20000001, the original data in 0x20000001 is 0xA5, write data is 0x5A. The result is 0xA5&0x5A. How to make up the decorated address:

addr[19:0] = 0x00001, addr[27:26] = 0x01, addr[30:29] = 0x01, so the decorated address is:

addr[31:0] = 0010 0100 0000 0000 0000 0000 0000 0001 =0x24000001. We can see that the decorated address and memory address is idential in addr[30:29] =0x01. The 8-bit write mode C code as below:

```
(*(volatile uint8_t *)(uint32_t)0x20000001) = 0xF5; /* put 0xF5 to 0x20000001 */
printf("0x%x\n", (*(volatile uint8_t *)(uint32_t)0x20000001));
(*(volatile uint8_t *)(uint32_t)0x24000001) = 0x5A; /* write 0x5A to decorated address */
printf("0x%x\n", (*(volatile uint8_t *)(uint32_t)0x20000001));
```

After AND opertion, the data in 0x20000001 should be 0x50.

3.1.1.2 Halfword (16-bit) write mode

The bit[0] of SRAM_U or peripheral address should be 0.

For example, SRAM_U memory address is 0x20000002, the original data in 0x20000002~0x20000003 is 0xF5F5, write data is 0x5A5A. The result is 0xF5F5&0x5A5A. How to make up the decorated address:

addr[19:0] = 0x00002, addr[27:26] = 0x01, addr[30:29] = 0x01, so the decorated address is:

addr[31:0] = 0010 0100 0000 0000 0000 0000 0000 0010 =0x24000002. The 16-bit write mode C code as below:

```
\ (*(volatile uint16_t *)(uint32_t)0x20000002) = 0xF5F5; /* put 0xF5F5 to 0x20000002 */
printf("0x%x\n", (*(volatile uint16_t *)(uint32_t)0x20000002));
 (*(volatile uint16_t *)(uint32_t)0x24000002) = 0x5A5A; /* write 0x5A5A to decorated address */
printf("0x%x\n", (*(volatile uint16_t *)(uint32_t)0x20000002));
```

After AND opertion, the data in 0x20000002~0x20000003 should be 0x5050.

3.1.1.3 Word (32-bit) write mode

The bit[1:0] of SRAM_U or peripheral address should be 00.

For example, SRAM_U memory address is 0x20000004, the original data in 0x20000004 ~0x20000007 is 0xF5F5F5F5, write data is 0x5A5A5A5A. The result is 0xF5F5F5F5&0x5A5A5A5A. How to make up the decorated address:

addr[19:0] = 0x00004, addr[27:26] = 0x01, addr[30:29] = 0x01, so the decorated address is:

addr[31:0] = 0010 0100 0000 0000 0000 0000 0000 0100 =0x24000004. The word(32-bit) write mode C code is:

```
(*(volatile uint32_t *)(uint32_t)0x20000004) = 0xF5F5F5F5; /* put 0xF5F5F5F5 to 0x20000002
*/
```

printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x20000004));

(*(volatile uint32_t *)(uint32_t)0x24000004) = 0x5A5A5A5A; /* write 0x5A5A5A5A to decorated address */

printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x20000004));

After AND opertion, the data in 0x20000004 ~0x20000007 should be 0x50505050.

3.1.2 OR

OR command performs a logical OR operation. It also supports 8-, 16-, or 32-bit access modes.

For example, SRAM_U memory address is 0x20000008, the original data in 0x20000008~0x2000000B is 0xA5A5A5A5, write data is 0x5A5A5A5A. The result is 0xA5A5A5A5|0x5A5A5A5A. Next steps to make up the decorated address:

addr[19:0] = 0x00008, addr[27:26] = 0x02, addr[30:29] = 0x01, so the decorated address is:

addr[31:0] = 0010 0100 0000 0000 0000 0000 0000 1000 =0x28000008. The word (32-bit) write mode C code is:

(*(volatile uint32_t *)(uint32_t)0x20000008) = 0xA5A5A5A5; /* put 0xA5A5A5A5 to 0x20000008 */

printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x20000008));

(*(volatile uint32_t *)(uint32_t)0x28000008) = 0x5A5A5A5A; /* write 0x5A5A5A5A to decorated address */

printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x20000008));

After OR opertion, the data in 0x20000008~0x2000000B should be 0xFFFFFFFF.

3.1.3 XOR

A XOR command performs a logical XOR operation. XOR supports 8-, 16-, and 32-bit access modes.

For example, SRAM_U memory address is 0x2000000C, the inital data in 0x2000000C~0x2000000F is 0xA5A5A5A5, write data is 0xFFFFFFFF. The result is 0xA5A5A5A5^0xFFFFFFFF. Next steps to make up the decorated address:

addr[19:0] = 0x0000C, addr[27:26] = 0x03, addr[30:29] = 0x01, so the decorated address is:

addr[31:0] = 0010 1100 0000 0000 0000 0000 0000 1100 =0x2C00000C. Because the word (32-bit) write mode C code is:

(*(volatile uint32_t *)(uint32_t)0x2000000C) = 0xA5A5A5A5; /* put 0xA5A5A5A5 to 0x2000000C */

printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x2000000C));

(*(volatile uint32_t *)(uint32_t) 0x2C00000C) = 0xFFFFFFFF; /* write 0xFFFFFFFF to decorated address */

printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x2000000C));

After XOR opertion, the data in 0x2000000C~0x2000000F should be 0x5A5A5A5A.

3.2 BFI

The BFI operation can be use to insert data into SRAM_U or peripheral registers. There is 0xFFFFF55F in 0x20000010~0x20000013. We want to insert 0xFF from 4th bit, so the b=4, w=7. And we hope to get 0xFFFFFFFF. Next steps to make up the decorated address:

addr[18:0] = 0x00010, addr[22:19] = 0x07, addr[27:23]=0x04 , addr[30:29]=0x01, so the decorated address is:

addr[31:0] = 0011 0010 0011 1000 0000 0000 0001 0000 =0x32380010. The word(32-bit) write mode C code is:

```
(*(volatile uint32_t *)(uint32_t)0x20000010) = 0xFFFFF55F; /* put 0xFFFFF55F to 0x20000010 */
printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x20000010));
(*(volatile uint32_t *)(uint32_t)0x32380010) = (0xFF<<4); /* write 0xFF to decorated address */
printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x20000010));
```

The write data should shift to left-corresponding.

After the BFI opertion, the data in 0x20000010~0x20000013 should be 0xFFFFFFFF.

3.3 Decorated Load Operations

Decorared load includes LAC1, LAS, and UBFX operations. Each opperation supports 8-, 16-, and 32-bit write modes.

LAC1 and LAS1 convert to a two-cycle atomic read-modify-write sequence, but UBFX is only a single data read and not an read-modify-write.

3.3.1 LAC1

The LAC1 operation can be use to clear the bit in SRAM_U or peripheral registers.

For example, There is 0xFFFFFFFF in SRAM_U 0x20000014~0x20000017 space. We want to clear the 3rd bit and get 0xFFFFFFF7. Next steps to make up the decorated address:

addr[19:0] = 0x00014, addr[25:21] = 0x03, addr[30:29]=0x01, so the decorated address is:

addr[31:0] = 0010 1000 0110 0000 0000 0000 0001 0100 =0x28600014. The word(32-bit) write mode C code is:

```
(*(volatile uint32_t *)(uint32_t)0x20000014) = 0xFFFFFFFF; /* put 0xFFFFFFFF to 0x20000014 */
printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x20000014));
u32Temp = (*(volatile uint32_t *)(uint32_t)0x28600014) ; /* read decorated address */
printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x20000014));
```

After LAC1 opertion, the data in 0x20000010~0x20000013 should be 0xFFFFFFF7.

3.3.2 LAS1

The LAS1 command can be use to set the bit in SRAM_U or peripheral registers.

For instance, there is 0xFFFFFFF7 in SRAM_U 0x20000018~0x2000001B space. We want to set the 3rd bit and get 0xFFFFFFFF. Next steps to make up the decorated address:

addr[19:0] = 0x00014, addr[25:21] = b=0x03, addr[30:29]=0x01, so the decorated address is:

addr[31:0] = 0010 1100 0110 0000 0000 0000 0001 1000 =0x28600014.

The word (32-bit) write mode C code is:

```
(*(volatile uint32_t *)(uint32_t)0x20000018) = 0xFFFFFFF7; /* put 0xFFFFFFF7 to 0x20000018 */
printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x20000018));
u32Temp = (*(volatile uint32_t *)(uint32_t)0x2C600018) ; /* read decorated address */
printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x20000018));
```

After LAS1 opertion, the data in 0x20000018~0x2000001B should be 0xFFFFFFFF.

3.3.3 UBFX

UBFX command is used to extract a bit field from SRAM_U or peripheral registers. For instance, the initial data is 0x5555AAAA in SRAM_U 0x2000001C~0x2000001F. If the user wants to extract the bit field 0x5A from this space, the decorated address is:

addr[18:0] = 0x0001C, addr[23:19] = w=0x07, addr[27:23]=b=0x0C , addr[30:29]=0x01, so the decorated address is:

addr[31:0] = 0011 0110 0011 1000 0000 0000 0001 1100 =0x3638001C.

The word (32-bit) write mode C code is:

```
(*(volatile uint32_t *)(uint32_t)0x2000001C) = 0x5555AAAA; /* put 0x5555AAAA to 0x2000001C */
printf("0x%x\n", (*(volatile uint32_t *)(uint32_t)0x2000001C));
u32Temp = (*(volatile uint32_t *)(uint32_t) 0x3638001C) ; /* read extract data */
printf("0x%x\n", u32Temp);
```

After BFI opertion, the extracted data should be 0x5A.

3.4 GPIO ACCESS

The GPIO can be accessed by the core through the crossbar/AIPS interface at 0x400FF000 and at an aliased slot (15) at address0x4000F000. All BME operations to the GPIO space can be accomplished referencing the aliased slot (15) at address 0x4000F000. Only some of the BME operations can be accomplished referencing GPIO at address 0x400FF000, such as AND, OR, XOR, LAC1, and LAS1. But BFI and UBFX operations only can be realized at 0x4000F000.

3.5 BME Library

There is a BME library (BME.h) in the attched code. The head file defines all the decorated operations under 8-, 16-, and 32-bit write modes. It is convenient to migrate it to the application code. The following example code shows how to utilize the BME library to access the peripheral registers and GPIO.

Example 1: Disable low-voltage detect in PMC_SPMSC1 register after reset.

The normal C code is:

```
PMC->SPMSC1 &= ~PMC_SPMSC1_LVDE_MASK;
```

The BME C code using library:

```
u8Temp = BME_BIT_CLEAR_8b(&PMC->SPMSC1, 0x02);
```

Example 2: Toggle PTA1 output.

The normal C code is:

```
GPIOA->PDOR ^= 0x02;
```

The BME C code using library:

```
BME_XOR(&GPIOA->PDOR) = 0x02;
```

4 Contrasting Normal C and BME operations

As BME operations are more efficient than normal C code, we can contrast the disassembly code of the above two examples in IAR Embedded WorkBench V6.60.

Example 1:

```
PMC->SPMSC1 &= ~PMC_SPMSC1_LVDE_MASK;
LDR.N     R0, ??DataTable2
LDRB      R1, [R0]
MOVS      R2, #251
ANDS      R2, R2, R1
STRB      R2, [R0]

u8Temp = BME_BIT_CLEAR_8b(&PMC->SPMSC1, 0x02);
LDR.N     R0, ??DataTable2_1
LDRB      R0, [R0]
```

Example 2:

```
GPIOA->PDOR ^= 0x02;
```

```
MOVS     R0, #2
LDR.N    R1, ??DataTable2_2
LDR      R2, [R1]
EORS     R2, R2, R0
STR      R2, [R1]
BME_XOR(&GPIOA->PDOR) = 0x02;
LDR.N    R1, ??DataTable2_3
STR      R0, [R1]
```

5 Conclusion

The bit-band and BME performs with higher efficiency than a normal C function, and it is convient to migrate the drivers to customer's application code.

6 Demo Code

The demo code demonstrates the performance with BME hardware operations and Normal C code operations. The demo code is built under IAR Embedded Workbench V6.60. The user can evaluate it with FRDM-KE04Z hardware platform. If the user has another application board, they should change the project options and select the correct debug tool.

7 References

MKE04Z24M48SF0RM Reference Manual

8 Glossary

Table 8-2. Glossary

Term	Definition
SRAM_U	Upper SRAM
BME	Bit manipulation engine
UBFX	Unsigned bit field extract
LAC	Load-and-clear 1 bit
LAS1	Load-and-set 1 bit

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4838
Rev. 0
12/2013