Freescale Semiconductor
应用笔记

Document Number: AN4838
Rev. 0, 12/2013

如何在 KE04 和 KE06 子系列上使用 bit-band 和 BME

作者：Cheng Yangtao

亚太微控制器解决方案小组

1 简介

由于 ARM Cortex-M0+ 处理器上的 bit-band 功能为可选项，但 Freescale 在 KE04 和 KE06 子系列器件上均实现了高位 SRAM（SRAM_U）bit-band 功能。bit-band 功能的操作与 ARM Cortex-M3 和 ARM Cortex-M4 处理器上的操作相似。它可以将存储器中的一个完整的字映射到 bit-band 功能中的单个位。例如，写入一个别名区存储位置会置位或清零 bit-band 功能中的相应位。它使得单个位无需执行 “读取-修改-写入” 序列便能进行切换。

位操作引擎 （BME）提供原子 “读取/修改/写入” 外设和 SRAM_U 地址空间的硬件操作。原子 “读取/修改/写入” 操作是一个不能分割的 “先读后写” 总线序列。 BME 硬件的微架构采用二级流水线设计，符合 AMBA-AHB 系统总线接口协议。通过将 ARM Cortex-M0+ 指令集架构中的基本加载和存储指令支持与 BME 提供的已修饰存储概念相整合， BME 可以在 KE 系列 Cortex-M0+ 内核微控制器上实现可靠而高效的 “读取-修改-写入” 功能。

目录

2 bit-band

以下映射公式演示了如何将别名区域中的每个字与 bit-band 功能中的相应位或目标位相匹配。

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

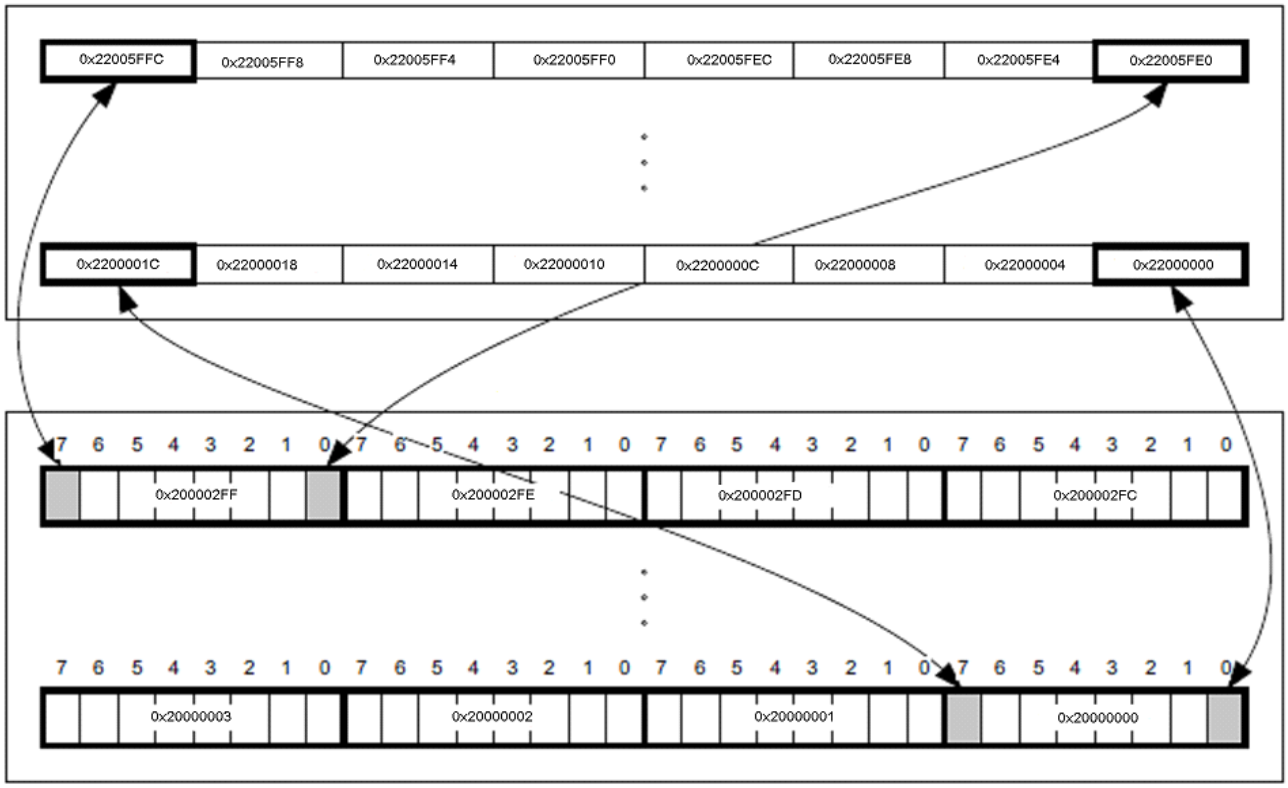
bit_word_addr = bit_band_base + bit_word_offset

其中：

- bit_word_offset 是 bit-band 存储器区中的目标位位置。
- bit_word_addr 是别名存储器区域中映射到目标位的字地址。
- bit_band_base 是别名区域的起始地址。
- byte_offset 是 bit-band 功能中包含目标位的字节数。
- bit_number 是目标位的位位置 （0 至 7）。

例如，位于 0x22000000 地址的别名字映射到位于 0x20000000 地址的 bit-band 字节的位 [0]：0x22000000 = 0x22000000 + (0*32) + 0*4。

图 1. KE04 bit-band 映射



2.1 bit-band 操作

别名区的字写操作与对 bit-band 功能中目标位进行的 “读取/修改/写入”操作的作用相同。

写入别名区中一个字的值的位 [0] 决定了写入 bit-band 功能内目标位的值。写入位 [0] 置位的值会向 bit-band 位写入 1，写入位 [0] 清零的值会向 bit-band 位写入 0。

读取别名区中的字会返回 0x01 或 0x00。值 0x01 表示 bit-band 功能中的目标位置位。值 0x00 表示目标位清零。

2.2 bit-band 库

用户可以通过宏定义简单的 bit-band 访问：

```
#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*/
```

例如，如果用户想置位 0x20000000 的位 1，则 C 语言代码为：

```
Bit_Band_Set(0x20000000, 1 ) ;
```

附带的代码中提供了一个 bit-band 库 （bitband.h），用户可以参考其中的 bit-band 演示代码。

3 BME 简介

仅处理器内核产生的系统总线事务提供 BME 已修饰参考，其目标是基地址为 0x40000000~0x4007FFFF 的标准 512 KB 外设地址空间和基地址为 0x20000000 的 SRAM_U 空间。该修饰语义嵌入到地址位 [28:19]，为 AIPS 创建一个大小为 448 MB、地址范围为 0x44000000~0x5FFFFFFF 的空间，为 SRAM_U 创建一个大小为 448 MB、地址范围为 0x24000000~0x3FFFFFFF 的空间。这些位从实际地址提出并发送到外设总线控制器， BME 使用它们来定义和控制其操作。

BME 支持已修饰加载和已修饰存储操作。已修饰加载包括无符号位字段提取 （UBFX）、加载并清零 1 位（LAC1）和加载并置位 1 位（LAS1）操作。已修饰存储包括 AND、OR、XOR 和位字段插入 （BFI）运算。

表 1. 基地址和 BME 操作

模块	基地址	已修饰地址空间	已修饰存储				已修饰加载		
—	—	—	AND	OR	XOR	BVFI	LAC1	LAS1	UBFX
SRAM_U	0x20000000	0x24000000-0x3FFFFFFF	是	是	是	是	是	是	是
外设	0x4000F000	0x44000000-0x4FFFFFFF	是	是	是	是	是	是	是
GPIO	0x4000FF000	0x440000000-0x4FFFFFFF	是	是	是	否	是	是	否
	0x4000F0000	0x50000000-0x5FFFFFFF	是	是	是	是	是	是	是

注： 0x4000F000 为 GPIO 控制器的基地址，别名地址为 0x400FF000。
注： “是”表示该操作可行。
注： “否”表示该操作不可行。

用户必须从已修饰地址处写入或读取目标数据。每个操作均具有固定形式。用户必须使用正确的 32 位已修饰地址，如下所示。

操作	已修饰地址													
	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]					

图 2. 已修饰地址的组成

注： addr[19:0] 和 addr[18:0] 为外设地址或 SRAM_U 地址。
注： b 为 LSB 的位置。表示从该位开始操作。例如，地址 0x20000000 的值为 11101111，如果用户希望将位 4 从 0 置为 1，则 b 应为 4。
注： w 为位字段宽度减 1 的标识符。例如，如果位字段为 “1001”，则 w 应为 3。
注： “-”位可为 0 或 1，这是无关位。
注： addr[30:29] 为 SRAM_U 或外设 option，addr[30:29]=01 表示 SRAM_U，addr[30:29]=10 表示外设。已修饰地址和目标地址中的 Addr[30:29] 的值是一致的。

3.1 已修饰存储操作

已修饰存储包括三种通用逻辑运算：AND、 OR 和 XOR，以及位字段插入。每种操作均为一个双周期原子 “读取-修改-写入”序列。数据大小可以为 8、 16 或 32 位。

对于 16 位写模式，SRAM_U 或外设地址的位 [0] 应为 0。对于 32 位写模式，SRAM_U 或外设地址的位 [1:0] 应为 00。 8 位写模式没有类似限制。

3.1.1 AND

AND 命令执行逻辑与运算。请参见以下部分提供的示例。

3.1.1.1 字节 （8 位）写模式

SRAM_U 存储器地址为 0x20000001， 0x20000001 中的原始数据为 0xA5，写入数据为 0x5A。结果为 0xA5&0x5A。如何编写已修饰地址：

addr[19:0] = 0x00001、 addr[27:26] = 0x01 且 addr[30:29] = 0x01，因此已修饰地址为：

addr[31:0] = 0010 0100 0000 0000 0000 0000 0000 0001 =0x24000001。我们可以看到已修饰地址和存储器地址的 addr[30:29] 相同，均为 0x01。 8 位写模式的 C 语言代码如下：

(*(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));

经过逻辑与运算后， 0x20000001 中的数据应为 0x50。

3.1.1.2 半字 （16 位）写模式

SRAM_U 或外设地址的位 [0] 应为 0。

例如， SRAM_U 存储器地址为 0x20000002， 0x20000002~0x20000003 中的原始数据为 0xF5F5，写入数据为 0x5A5A。结果为 0xF5F5&0x5A5A。如何编写已修饰地址：

addr[19:0] = 0x00002、 addr[27:26] = 0x01 且 addr[30:29] = 0x01，因此已修饰地址为：

addr[31:0] = 0010 0100 0000 0000 0000 0000 0000 0010 =0x24000002。 16 位写模式的 C 语言代码如下：

\ (*(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));

经过逻辑与运算后， 0x20000002~0x20000003 中的数据应为 0x5050。

3.1.1.3 字 （32 位）写模式

SRAM_U 或外设地址的位 [1:0] 应为 00。

例如， SRAM_U 存储器地址为 0x20000004， 0x200000024~0x20000007 中的原始数据为 0xF5F5F5F5，写入数据为 0x5A5A5A5A。结果为 0xF5F5F5F5&0x5A5A5A5A。如何编写已修饰地址：

addr[19:0] = 0x00004、 addr[27:26] = 0x01 且 addr[30:29] = 0x01，因此已修饰地址为：

addr[31:0] = 0010 0100 0000 0000 0000 0000 0000 0100 =0x24000004。字（32 位）写模式的 C 语言代码如下：

```
(*(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));
```

经过逻辑与运算后， 0x20000004~0x20000007 中的数据应为 0x50505050。

3.1.2 OR

OR 命令执行逻辑或运算。它还支持 8、 16 或 32 位访问模式。

例如， SRAM_U 存储器地址为 0x20000008， 0x20000008~0x2000000B 中的原始数据为 0xA5A5A5A5，写入数据为 0x5A5A5A5A。结果为 0xA5A5A5A5|0x5A5A5A5A。下一步是编写已修饰地址：

addr[19:0] = 0x00008、 addr[27:26] = 0x02 且 addr[30:29] = 0x01，因此已修饰地址为：

addr[31:0] = 0010 0100 0000 0000 0000 0000 0000 1000 =0x28000008。字（32 位）写模式的 C 语言代码如下：

```
(*(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));
```

经过逻辑或运算后， 0x20000008~0x2000000B 中的数据应为 0xFFFFFFFF。

3.1.3 XOR

XOR 命令执行逻辑异或运算。 XOR 支持 8、 16 或 32 位访问模式。

例如， SRAM_U 存储器地址为 0x2000000C， 0x2000000C~0x2000000F 中的初始数据为 0xA5A5A5A5，写入数据为 0xFFFFFFFF。结果为 0xA5A5A5A5^0xFFFFFFFF。下一步是编写已修饰地址：

addr[19:0] = 0x0000C、 addr[27:26] = 0x03 且 addr[30:29] = 0x01，因此已修饰地址为：

addr[31:0] = 0010 1100 0000 0000 0000 0000 0000 1100 =0x2C00000C。字 （32 位）写模式的 C 语言代码如下：

(*(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));

经过逻辑异或运算后， 0x2000000C~0x2000000F 中的数据应为 0x5A5A5A5A。

3.2 BFI

BFI 操作可用来向 SRAM_U 或外设寄存器插入数据。 0x20000010~0x20000013 中包含数据 0xFFFFF55F。我们想从第 4 位插入 0xFF，因而 b=4 且 w=7。然后，我们希望获得 0xFFFFFFFF。下一步是编写已修饰地址：

addr[18:0] = 0x00010、 addr[22:19] = 0x07、 addr[27:23]=0x04 且 addr[30:29]=0x01，因此已修饰地址为：

addr[31:0] = 0011 0010 0011 1000 0000 0000 0001 0000 =0x32380010。字（32 位）写模式的 C 语言代码如下：

(*(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));

写入数据应相应左移。

经过 BFI 操作后， 0x20000010~0x20000013 中的数据应为 0xFFFFFFFF。

3.3 已修饰加载操作

已修饰加载包括 LAC1、 LAS 和 UBFX 操作。每种操作均支持 8、 16 和 32 位写模式。

LAC1 和 LAS1 转换为双周期原子 “读取-修改-写入”序列，而 UBFX 仅为单个数据读取操作，而非 “读取-修改-写入”序列。

3.3.1 LAC1

LAC1 操作可用来清零 SRAM_U 或外设寄存器中的位。

例如， SRAM_U 0x20000014~0x20000017 空间中包含数据 0xFFFFFFFF。我们想清零第三位，并获得值 0xFFFFFFF7。下一步是编写已修饰地址：

addr[19:0] = 0x000014、 addr[25:21] = 0x03 且 addr[30:29] = 0x01，因此已修饰地址为：

addr[31:0] = 0010 1000 0110 0000 0000 0000 0001 0100 =0x28600014。字（32 位）写模式的 C 语言代码如下：

```
(*(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));
```

经过 LAC1 操作后， 0x20000010~0x20000013 中的数据应为 0xFFFFFFF7。

3.3.2 LAS1

LAS1 命令可用来置位 SRAM_U 或外设寄存器中的位。

例如， SRAM_U 0x20000018~0x2000001B 空间中包含数据 0xFFFFFFF7。我们想置位第三位，并获得值 0xFFFFFFFF。下一步是编写已修饰地址：

addr[19:0] = 0x00014、 addr[25:21] = b=0x03 且 addr[30:29]=0x01，因此已修饰地址为：

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

字 （32 位）写模式的 C 语言代码如下：

```
(*(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));
```

经过 LAS1 操作后， 0x20000018~0x2000001B 中的数据应为 0xFFFFFFFF。

3.3.3 UBFX

UBFX 命令用于从 SRAM_U 或外设寄存器中提取位字段。例如， SRAM_U 0x2000001C~0x2000001F 中的初始数据为 0x5555AAAA。如果用户想从该空间提取位字段 0x5A，则已修饰地址为：

addr[18:0] = 0x0001C、addr[23:19] = w=0x07、addr[27:23]=b=0x0C 且 addr[30:29]=0x01，因此已修饰地址为：

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

字 （32 位）写模式的 C 语言代码如下：

```
(*(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);
```

经过 UBFX 操作后，所提取的数据应为 0x5A。

3.4 GPIO 访问

可由内核通过 0x400FF000 处的交叉开关/AIPS 接口以及地址 0x4000F000 处的别名 slot （15）对 GPIO 进行访问。对 GPIO 空间的所有 BME 操作都可以通过引用地址 0x4000F000 处的别名 slot （15）完成。只有部分 BME 操作可通过引用地址 0x400FF000 上的 GPIO 完成，如 AND、 OR、XOR、 LAC1 和 LAS1。而 BFI 和 UBFX 操作只能在 0x4000F000 处实现。

3.5 BME 库

附件代码中包含一个 BME 库 （BME.h）。头文件定义了在 8、16 和 32 位写模式下的所有已修饰操作。因而便于移植到应用程序代码中。以下示例代码显示了如何利用 BME 库访问外设寄存器和 GPIO。

例 1：复位后在 PMC_SPMSC1 寄存器中禁止低电压检测。

一般的 C 语言代码为：

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

使用 BME 库的 C 语言代码为：

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

例 2：切换 PTA1 输出。

一般的 C 语言代码为：

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

使用 BME 库的 C 语言代码为：

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

4 C 语言和 BME 操作的对比

我们在 IAR Embedded WorkBench V6.60 中对上述两个示例的反汇编代码进行对比，可以发现 BME 操作比 C 语言代码更高效。

例 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]
```

例 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 结语

bit-band 和 BME 的执行效率高于一般的 C 语言函数，并且可以十分方便地将驱动程序移植到客户的应用程序代码。

6 演示代码

该演示代码演示了利用 BME 硬件操作和一般 C 语言代码操作的性能。演示代码在 IAR Embedded Workbench V6.60 中进行编译。用户可以利用 FRDM-KE04Z 硬件平台对其进行评估。如果用户使用的是其他应用电路板，则需要修改工程 options 并选择正确的调试工具。

7 参考文献

MKE04Z24M48SF0RM 参考手册

8 术语表

表 2. 术语表

术语	定义
BME	位操作引擎
LAC	加载并清零 1 位
LAS1	加载并置位 1 位
SRAM_U	高地址 SRAM 区域
UBFX	无符号位字段提取

How to Reach Us:

Home Page:

Freescale.com

Web Support:

Freescale.com/support

Document Number: AN4838
Rev 0, 12/2013