Freescale Semiconductor
Engineering Bulletin

Document Number: EB758
Rev. 0, 10/2011

Using Qorivva Header Files in C

Avoiding Unexpected Operation

by: Martin Latal
Technical Support, Roznov Czech System Center

1 Introduction

This document describes how to use Qorivva header files in the C programming language. It highlights the instructions generated by C compilers and discusses the related consequences for the MCU's hardware. This bulletin also presents typical use cases—such as peripheral register initialization, interrupt flag clearing, and software watchdog servicing—and makes recommendations for effective MCU header file usage.

2 MCU header file

Power Architecture® Qorivva MCU header files use unions to define MCU memory-mapped registers and to provide a structure that puts all register definitions for a particular hardware component together under one structure name.

All the MCU definitions are defined with the volatile keyword; thus, they are not a subject of compiler optimizations either for code size or execution speed.

Contents

2.1 Register and module definition example

An example of a PIT Module Control Register definition and the PIT module structure, as taken from the MPC5643L.h header file, is shown below. First, a union for the PITMCR register is defined, then a PIT module structure is defined, and, lastly, the PIT module structure is assigned to the 32-bit address of the module, as specified by the hardware.

```
/****************************************************************/
/*                                                              */
/* Module: PIT_RTI  */
/*                                                              */
/****************************************************************/

typedef union {   /* PIT_RTI_PITMCR - PIT Module Control Register */
      vuint32_t R;
      struct {
          vuint32_t:30;
          vuint32_t  MDIS:1; /* Module Disable. Disable the module clock */
          vuint32_t  FRZ:1; /* Freeze. Allows the timers to be stopped when the device enters
          debug mode */
      } B;
} PIT_RTI_PITMCR_32B_tag;

typedef struct PIT_RTI_struct_tag {


      /* PIT_RTI_PITMCR - PIT Module Control Register */
      PIT_RTI_PITMCR_32B_tag PITMCR;       /* offset: 0x0000 size: 32 bit */

      ...
      ... /* there are many other registers in the module */
      ...
} PIT_RTI_tag;


#define PIT_RTI (*(volatile PIT_RTI_tag *) 0xC3FF0000UL)
```

2.2 Register instantiations

With C unions defined, there are two different instantiations that could be used in a C program:

```
<MODULE>.<REGISTER>.R = 0x00000001;
<MODULE>.<REGISTER>.B.<BIT> = 1;
```

To understand the difference between the two instantiations, we will need to have a look at the instructions generated, for each, by a C compiler.

2.2.1 Instantiation (.R)

```
PIR_RTI.PITMCR.R = 0x00000003;
```

The Freescale CodeWarrior for MPC55xx/MPC56xx compiler generates the following Power Architecture® VLE instructions:

```
se_li    r0, 3       /* load immediate value 3 to r0 */
e_lis    r3, 0xC3FF  /* load base address of the PIT module to r3 */
se_stw   r0, 0(r3)   /* store the 32-bit content of r0 to the addr pointed by r3 */
```

Note the CPU operation is a 32-bit write to a memory location.

2.2.2 Instantiation (.B)

```
PIT_RTI.PITMCR.B.FRZ = 1;
```

The Freescale CodeWarrior for MPC55xx/MPC56xx compiler generates the following Power Architecture VLE instructions:

```
se_li    r4,1         /* load immediate value 1 to r4 */
e_lis    r3,0xc3ff    /* load base address of the PIT module to r3 */
se_lwz   r0,0(r3)     /* read value pointed by r3 to r0 */
e_insrwi r0,r4,1,31   /* modify the value in r0 */
se_stw   r0,0(r3)     /* store r0 value to the address pointed by r3 */
```

Note that the CPU operation for setting one bit in the register consists of the following sub-operations:

1. Read the register value from a memory to an internal CPU register.
2. Modify the content of the internal CPU register (insert one specific bit and leave other bits of the target CPU register unchanged).
3. Write the value from the internal CPU register to the memory.

This is a typical read-modify-write sequence. Notice that to set just one bit using the Instantiation (.B), the C compiler generates five instructions. Compare this approach with the Instantiation (.R), where three instructions set all the register bits that we need.

2.3 C code reading

We can now see, just from reading the C code, that the second instantiation is actually a read-modify-write sequence. It should be considered if this is the intended operation. The next section of this document considers typical use cases and gives recommendations on the usage of the two instantiations.

3 Recommended use cases

3.1 Register initialization

Typically during system startup, a lot of hardware register initializations need to be performed by the CPU in order to initialize the system for the application.

Recommendation 1

Use Instantiation (.R) to write to a hardware register.

```
<MODULE>.<REGISTER>.R = #<immediate_value>;
```

Recommendation 2

To increase readability of the code, use C macros to define bit masks.

```
#define PIT_RTI_PITMCR_FRZ_MASK = 0x00000001;
#define PIT_RTI_PITMCR_MDIS_MASK = 0x00000002;

PIT_RTI.PITMCR.R = PIT_RTI_PITMCR_FRZ_MASK
                 | PIT_RTI_PITMCR_MDIS_MASK;
```

3.2 Status bit clearing

For this purpose, it is important NOT to use read-modify-write sequences.

NOTE

The term “status register” in this document is used for a memory-mapped peripheral register that contains one or several “write 1 to clear” (w1c) bits.

The term “status bit” in this document refers to a “write 1 to clear” (w1c) bit of a status register.

Recommendation 1

DO NOT USE Instantiation (.B)

The rationale for this recommendation is that since the status flag bits are typically cleared by writing a 1 to the bit, a read-modify-write operation could potentially clear other bits in the status register (if they are set) even if you don’t wish to do so. This kind of unwanted effect is shown in the example below:

Example 1.

```
/* starting situation: status bit0 and bit1 are set */
var1 = <MODULE>.<REGISTER>.R; /* var1 reads 0x00000003 */

/* clear the status bit0 by writing 1 to it */
<MODULE>.<REGISTER>.B.<STATUS_BIT0> = 1;

/* read the register back */
var1 = <MODULE>.<REGISTER>.R;

/* ! the register reads 0x00000000 ! */
/* the status bit1 is destroyed by the above read-modify-write operation */
```

The above example uses a status register, which has two least-significant status bits set. Status bit0 and bit1 are cleared by writing 1 to them. Note that because the read-modify-write sequence is an operation that inserts one bit and leaves other bits unchanged, the read-modify-write also clears the status bit1. See Figure 1 for an illustration.

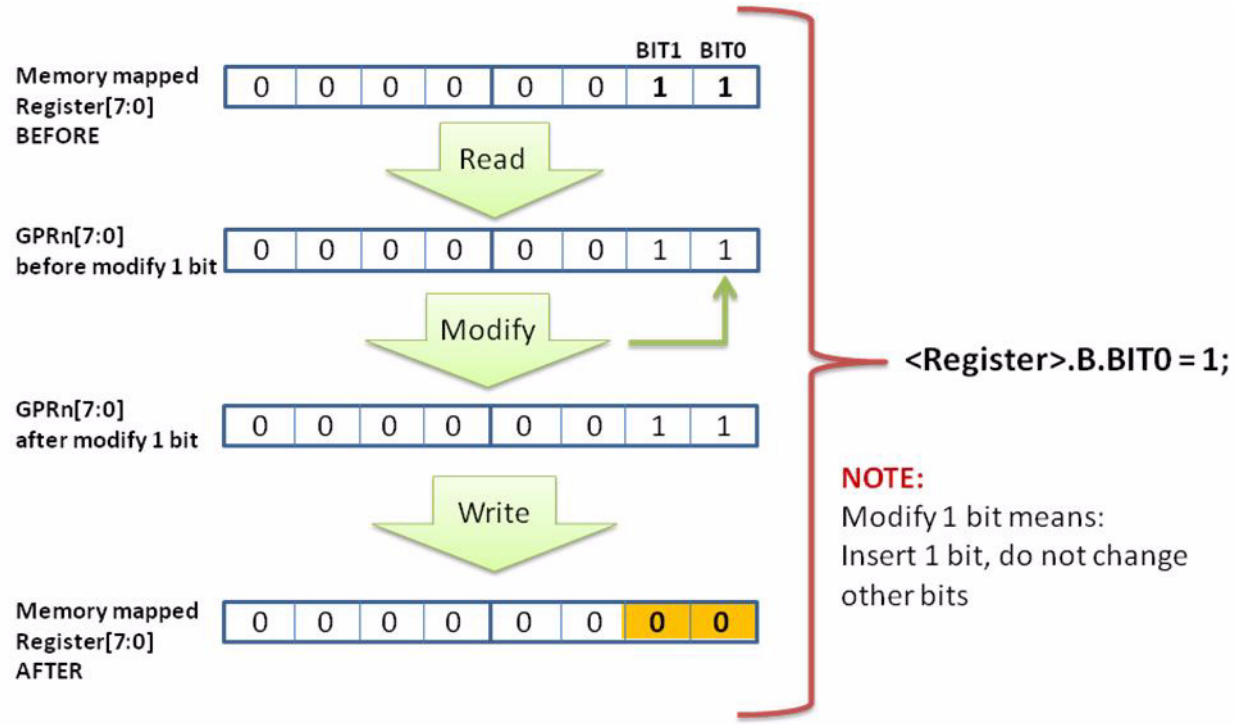

Figure 1. Unwanted effect of a read-modify-write operation on a memory-mapped status register.

Recommendation 2

Use Instantiation (.R) to clear a status bit in a hardware register.

```
<MODULE>.<REGISTER>.R = MODULE_REGISTER_BIT_MASK;
```

Example—clear the PIT Module Channel 3 TIF status bit

```
PIR_RTI.CHANNEL[3].TFLG.R = 0x00000001;
```

Or, even better:

```
#define PIT_RTI_CHANNEL_TFLG_TIF_MASK 0x00000001
PIT_RTI.CHANNEL[3].TFLG.R = PIT_RTI_CHANNEL_TFLG_TIF_MASK;
```

3.3 Bit testing

For bit testing, the Instantiation (.B) is useful because of its good readability in the source code.

Recommendation 1

Use Instantiation (.B) for conditional statements in C.

```
if (1 == <MODULE>.<REGISTER>.B.<BIT>)
{
}
```

Example—poll the PIT Module Channel 3 TIF status bit

```
while (0 == PIT_RTI.CHANNEL[3].TFLG.B.TIF)
{
```

```
}
```

Compare to the same functionality as implemented by the Instantiation (.R)

```
while (0x00000000 == (PIT_RTI.CHANNEL[3].TFLG.R & 0x00000001))
{
}
```

In both cases, the code generated by the C compiler will be the same. The only difference is the C source code.

```
e_lis    r8,0xc3ff   /* load base addr of the PIT module to r8 */
e_lwz    r0,316(r8)  /* load the PIT Channel 3 TFLG register value to r0 */
e_clrlwi r0,r0,31    /* clear all the bits but the lsb (TIF) */
se_cmpi  r0,0        /* compare with zero */
se_beq   *-14        /* conditional branch */
```

3.4 Servicing the software watchdog

The Qorivva MPC56xx Software Watchdog Timer (SWT) module requires two keyword writes to the SWT Module Service Register in order to reset the SWT timeout counter.

Recommendation 1

Use Instantiation (.R) for keyword writes to the SWT Module Service Register.

The rationale for this recommendation is that it is not necessary to generate read accesses to the SWT_SR register in between the two keyword write accesses. Therefore, a sequence of simple write accesses is preferred (instead of a sequence of read-modify-write accesses).

```
#define SWT_SR_KEYWORD1 0xC520
#define SWT_SR_KEYWORD2 0xD928

SWT.SR.R = SWT_SR_KEYWORD1;
SWT.SR.R = SWT_SR_KEYWORD2;
```

4 Conclusion

It is recommended that users of the Freescale-supplied MPC5xx/MPC55xx/MPC56xx MCU header files become familiar with the differences between the two instantiations of the MCU registers: Instantiation (.R) and Instantiation (.B).

For register writes, Instantiation (.R) produces more efficient code (smaller code size, faster startup code execution).

For status bit clearing, for example in an interrupt service routine, Instantiation (.R) is strongly recommended, because Instantiation (.B) may cause side effects to the hardware register. Instantiation (.R) is also more efficient.

Instantiation (.B) is especially useful for conditional statements in C programs (if else, while, for) due to better readability of the source code.

5 References

For further information, please refer to the documents listed in Table 1.

Table 1. References

Document	Title	Availability
EREFRM	EREF 2.0: A Programmer's Reference Manual for Freescale Power Architecture Processors	www.freescale.com
VLEPEM	Variable-Length Encoding (VLE) Programming Environments Manual	
MPC5643LRM	MPC5643L Microcontroller Reference Manual	

6 Revision history

Table 2. Changes made April 2012[1]

Section	Description
Front page	Add SafeAssure branding.
Back page	Apply new back page format.

[1] No substantive changes were made to the content of this document; therefore the revision number was not incremented.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: EB758
Rev. 0
10/2011