

Building Bare-Metal ARM Systems with GNU

Part 8: Low-level Interrupt Wrapper Functions

Miro Samek

In this part I describe the low-level interrupt wrapper functions `ARM_irq()` and `ARM_fiq()`. These functions have been introduced in part 6 of this article series, and their purpose is to allow handling of nested interrupts in the ARM architecture, which the GNU `gcc __attribute__((interrupt("IRQ")))` cannot do.

Perhaps the most interesting aspect of the implementation that I present here is its close compatibility with the new ARM v7-M architecture (e.g., Cortex-M3) [4]. Specifically, the low-level wrapper functions deal with all the ARM-magic internally, so that the interrupt service routines (ISRs) that you hook to the interrupt controller can be regular C-functions. The C-level ISRs run in the same processor mode (SYSTEM) as the code called from `main()` (task-level). Also, the assembler wrapper functions expressly avoid using the IRQ/FIQ stacks and instead nest interrupts of all types (IRQs and FIQs) on the SYSTEM/USER stack. The interrupt context saved to the stack is optimized for high-level languages and just like in the ARM v7-M specification, the wrapper functions save only the 8 registers clobbered in the ARM Architecture Procedure Calling Standard (AAPCS) [4]. In fact, the interrupt wrapper functions generate in software the exact same interrupt stack frame (SPSR, PC, LR, R12, R3, R2, R1, R0) as the ARM v7-M processors generate in hardware [4].

I should perhaps note right away that the ARM interrupt handling implementation described here goes off the beaten path established by the traditional approaches [1,2,3,5]. Most published implementations recommend initializing the ARM vector table to use the “auto-vectoring” feature of the interrupt controller (see part 6 of this article series). Consequently the ISRs that you hook to the interrupt controller require very special entry and exit sequences, so they cannot be regular C-functions. Also, the interrupt handlers execute in IRQ or FIQ mode and use the IRQ and FIQ stacks (at least for some portion of the saved context). Consequently the stack frames as well as stack usage in the traditional implementations are quite different compared to ARM v7-M.

1. The IRQ Interrupt Wrapper `ARM_irq`

The interrupt “wrapper” function `ARM_irq()` for handling the IRQ-type interrupts is provided in the file `arm_exc.s` included in the code accompanying this article series. Listing 1 shows the entire function.

```
.equ    NO_IRQ,      0x80          /* mask to disable IRQ */
.equ    NO_FIQ,     0x40          /* mask to disable FIQ */
.equ    NO_INT,     (NO_IRQ | NO_FIQ) /*mask to disable IRQ and FIQ */
.equ    FIQ_MODE,   0x11
.equ    IRQ_MODE,   0x12
.equ    SYS_MODE,   0x1F

. . .
.text
(1) .code 32
(2) .section .text.fastcode      . . .

ARM_irq:
/* IRQ entry {{{ */
(3)  MOV     r13,r0              /* save r0 in r13_IRQ */
(4)  SUB     r0,lr,#4           /* put return address in r0_SYS */
(5)  MOV     lr,r1              /* save r1 in r14_IRQ (lr) */
(6)  MRS     r1,spsr           /* put the SPSR in r1_SYS */

(7)  MSR     cpsr_c,#(SYS_MODE | NO_IRQ) /* SYSTEM mode, no IRQ/FIQ enabled! */
(8)  STMFD   sp!,{r0,r1}      /* save SPSR and PC on SYS stack */
```

```

(9)   STMFDB sp!, {r2-r3, r12, lr} /* save AAPCS-clobbered regs on SYS stack */
(10)  MOV    r0, sp /* make the sp_SYS visible to IRQ mode */
(11)  SUB    sp, sp, #(2*4) /* make room for stacking (r0_SYS, r1_SYS) */

(12)  MSR    cpsr_c, #(IRQ_MODE | NO_IRQ) /* IRQ mode, IRQ/FIQ disabled */
(13)  STMFDB r0!, {r13, r14} /* finish saving the context (r0_SYS, r1_SYS) */

(14)  MSR    cpsr_c, #(SYS_MODE | NO_IRQ) /* SYSTEM mode, IRQ disabled */
/* IRQ entry }}} */

/* NOTE: BSP_irq might re-enable IRQ interrupts (the FIQ is enabled
 * already), if IRQs are prioritized by an interrupt controller.
 */
(15)  LDR    r12, =BSP_irq
(16)  MOV    lr, pc /* copy the return address to link register */
(17)  BX    r12 /* call the C IRQ-handler (ARM/THUMB) */

/* IRQ exit {{{ */
(18)  MSR    cpsr_c, #(SYS_MODE | NO_INT) /* SYSTEM mode, IRQ/FIQ disabled */
(19)  MOV    r0, sp /* make sp_SYS visible to IRQ mode */
(20)  ADD    sp, sp, #(8*4) /* fake unstacking 8 registers from sp_SYS */

(21)  MSR    cpsr_c, #(IRQ_MODE | NO_INT) /* IRQ mode, both IRQ/FIQ disabled */
(22)  MOV    sp, r0 /* copy sp_SYS to sp_IRQ */
(23)  LDR    r0, [sp, #(7*4)] /* load the saved SPSR from the stack */
(24)  MSR    spsr_cxsf, r0 /* copy it into spsr_IRQ */

(25)  LDMFDB sp, {r0-r3, r12, lr}^ /* unstack all saved USER/SYSTEM registers */
(26)  NOP /* can't access banked register immediately */
(27)  LDR    lr, [sp, #(6*4)] /* load return address from the SYS stack */
(28)  MOVS  pc, lr /* return restoring CPSR from SPSR */
/* IRQ exit }}} */

```

Listing 1 The `ARM_irq()` assembly wrapper function defined in the file `arm_exc.s`.

The highlights of the implementation are as follows:

(1) The low-level IRQ/FIQ handlers must be written in the 32-bit instruction set (ARM), because the ARM core automatically switches to the ARM state when IRQ/FIQ is recognized.

(2) The `ARM_irq` wrapper function is defined in the special section `(.text.fastcode)`, which the linker script locates in RAM (see part 3 of this article series) for faster execution.

(3) The IRQ stack is not used, so the banked stack pointer register `r13_IRQ` (`sp_IRQ`) is used as a scratchpad register to temporarily hold `r0` from the SYSTEM context.

NOTE: As part of the IRQ startup sequence, the ARM processor sets the I bit in the CPSR (`CPSR[7] = 1`), but leaves the F bit unchanged (typically cleared), meaning that further IRQs are disabled, but FIQs are not. This implies that FIQ can be recognized while the ARM core is in the IRQ mode. The FIQ handler `ARM_fiq` discussed in the next section can safely preempt `ARM_irq` in all places where FIQs are not explicitly disabled.

(4) Now `r0` can be clobbered with the return address from the interrupt that needs to be saved to the SYSTEM stack.

(5) At this point the banked `lr_IRQ` register can be reused to temporarily hold `r1` from the SYSTEM context.

(6) Now `r1` can be clobbered with the value of `spsr_IRQ` register (Saved Program Status Register) that needs to be saved to the SYSTEM stack.

(7) Mode is changed to SYSTEM with IRQ interrupt disabled, but FIQ explicitly enabled. This mode switch is performed to get access to the SYSTEM registers.

NOTE: The F bit in the CPSR is intentionally cleared at this step (meaning that the FIQ is explicitly enabled). Among others, this represents the workaround for the Problem 2 described in ARM Tech-

(22) The SYSTEM stack pointer is copied to the banked sp_IRQ, which thus is set to point to the top of the SYSTEM stack

(23-24) The value of SPSR is loaded from the stack (please note that the SPSR is now 7 registers away from the top of the stack) and placed in SPSR_irq.

(25) The 6 registers are popped from the SYSTEM stack. Please note the special version of the LDM instruction (with the '^' at the end), which means that the registers are popped from the SYSTEM/USER stack. Please also note that the special LDM(2) instruction does not allow the write-back, so the stack pointer is not adjusted. (For more information please refer to Section "LDM(2)" in the "ARM Architecture Reference Manual" [5].)

Error! Reference source not found.(26) It's important not to access any banked register after the special LDM(2) instruction.

Error! Reference source not found.(27) The return address is retrieved from the stack. Please note that the return address is now 6 registers away from the top of the stack.

Error! Reference source not found.(28) The interrupt return involves loading the PC with the return address and the CPSR with the SPSR, which is accomplished by the special version of the MOVS pc,lr instruction.

2. The FIQ Interrupt Wrapper ARM_fiq

The interrupt "wrapper" function ARM_fiq() for handling the FIQ-type interrupts is provided in the file arm_exc.s included in the code accompanying this article series. Listing 2 shows the entire function.

```
(1) ARM_fiq:
    /* FIQ entry {{{ */
    MOV    r13,r0          /* save r0 in r13_FIQ */
    SUB    r0,lr,#4        /* put return address in r0_SYS */
    MOV    lr,r1           /* save r1 in r14_FIQ (lr) */
    MRS    r1,spsr         /* put the SPSR in r1_SYS */

(2)    MSR    cpsr_c,#(SYS_MODE | NO_INT) /* SYSTEM mode, IRQ/FIQ disabled */
    STMFD  sp!,{r0,r1}      /* save SPSR and PC on SYS stack */
    STMFD  sp!,{r2-r3,r12,lr} /* save APCS-clobbered regs on SYS stack */
    MOV    r0,sp            /* make the sp_SYS visible to FIQ mode */
    SUB    sp,sp,#(2*4)     /* make room for stacking (r0_SYS, SPSR) */

    MSR    cpsr_c,#(FIQ_MODE | NO_INT) /* FIQ mode, IRQ/FIQ disabled */
    STMFD  r0!,{r13,r14}   /* finish saving the context (r0_SYS,r1_SYS)*/

    MSR    cpsr_c,#(SYS_MODE | NO_INT) /* SYSTEM mode, IRQ/FIQ disabled */
/* FIQ entry }}} */

    /* NOTE: NOTE: BSP_fiq must NEVER enable IRQ/FIQ interrupts! */
    LDR    r12,=BSP_fiq
    MOV    lr,pc            /* store the return address */
(3)    BX    r12           /* call the C FIQ-handler (ARM/THUMB)

/* FIQ exit {{{ */
    MOV    r0,sp            /* both IRQ/FIQ disabled (see NOTE above) */
    ADD    sp,sp,#(8*4)    /* make sp_SYS visible to FIQ mode */
    /* fake unstacking 8 registers from sp_SYS */

    MSR    cpsr_c,#(FIQ_MODE | NO_INT) /* FIQ mode, IRQ/FIQ disabled */
    MOV    sp,r0            /* copy sp_SYS to sp_FIQ */
    LDR    r0,[sp,#(7*4)]  /* load the saved SPSR from the stack */
    MSR    spsr_cxsf,r0    /* copy it into spsr_FIQ */

    LDMFD  sp,{r0-r3,r12,lr}^ /* unstack all saved USER/SYSTEM registers */
```

```

NOP                                     /* can't access banked reg immediately */
LDR    lr, [sp, #(6*4)]                 /* load return address from the SYS stack */
MOVS   pc, lr                           /* return restoring CPSR from SPSR */
/* FIQ exit }}} */

```

Listing 2 The ARM_fiq() assembly wrapper function defined in the file arm_exc.s.

The ARM_fiq() “wrapper” function is very similar to the IRQ handler (Listing 1), except the FIQ mode is used instead of the IRQ mode. The following comments explain only the slight, but important differences in disabling interrupts and the responsibilities of the C-level handler BSP_fiq().

- (1) The FIQ handler is always entered with both IRQ and FIQ disabled, so the FIQ mode is not visible in any other modes. The ARM_fiq handler keeps the IRQ and FIQ locked at all times.
- (2) The mode is switched to SYSTEM to get access to the SYSTEM stack pointer. Please note that both IRQ and FIQ interrupts are kept disabled throughout the FIQ handler.
- (3) The C-function BSP_fiq() is called to perform the interrupt processing at the application-level. Please note that BSP_fiq() is now a regular C function in ARM or THUMB. Unlike the IRQ, the FIQ interrupt is often not covered by the priority controller, therefore the BSP_fiq() should NOT unlock interrupts.

NOTE: The BSP_fiq() function is entered with both IRQ and FIQ interrupts disabled and it should **NEVER** enable any interrupts. Typically, the FIQ line to the ARM core does not have a priority controller, even though the FIQ line typically goes through a hardware interrupt controller.

3. Coming Up Next

In the next part of this article I’ll wrap up the interrupt handling for ARM by presenting examples of the interrupt service routines (ISRs) in C, as well as the initialization of the vector table and the interrupt controller. I’ll also discuss a rudimentary policy of handling other ARM Exceptions, such as Undefined Instruction or Data Abort.

4. References

- [1] ARM Technical Support Note “Writing Interrupt Handlers” available online at www.arm.com/-support/faqdev/1456.html
- [2] Philips Application Note AN10381 “Nesting of Interrupts in the LPC2000” available online at www.nxp.com/acrobat_download/applicationnotes/AN10381_1.pdf
- [3] Atmel Application Note “Interrupt Management: Auto-vectoring and Prioritization” available online at www.atmel-grenoble.com/dyn/resources/prod_documents/DOC1168.PDF
- [4] ARM Limited, “ARM v7-M Architecture Application Level Reference Manual”, available from www.arm.com/products/CPUs/ARM_Cortex-M3_v7.html .
- [5] Seal, David Editor, “ARM Architecture Manual, 2nd Edition”, Addison Wesley 2000.

Miro Samek is the President of [Quantum Leaps, LLC](http://Quantum_Leaps,LLC), the provider of lightweight, state machine-based, event-driven frameworks for embedded systems. He is the author of “Practical Statecharts in C/C++” (CMP Books, 2002), has written numerous articles for magazines, and is a regular speaker at the Embedded Systems Conferences. He welcomes contact at miro@quantum-leaps.com.