Mar 15

Friday, March 15, 2019 3:11 PM

Enabling Interrupt Generation for an I/O Device

- By default an I/O device won't produce an IRQ signal and sent to the GIC. You code has to enable interrupts in each I/O device.
  E.g. KEYs port:

Address						
0xFF200050						Data
58	unused	1	1	1	1	Interrupt mask ①
5C						Edgecapture ②

① Your program stores a 1 for each KEY that should cause an interrupt
② Your KEY_ISR code will read the Edgecapture register to see which KEY was pressed. You code has to clear the bits(s) in Edgecapture (or else the same interrupt will occur repeatedly)

Summary: steps required to use Interrupts

0. Provide the Exception Vector Table (at least 0x18)
1. Initialize banked SP for IRQ mode
2. (Initialize SP for SVC mode, like before)
3. Configure the GIC to enable interrupts (code supplied)
4. Enable interrupt generation in (each) I/O device
5. Set I=0 in CPSR
6. Provide IRQ_HANDLER code, which queries the GIC to determine the source of the interrupt. It then calls the subroutine for the I/O device (e.g. KEY_ISR)
7. Provide code for the interrupt service routine for each I/O device (e.g. KEY_ISR)
8. The IRQ_HANDLER has to clear the interrupt from the GIC

```
            .global  SERVICE_IRQ
SERVICE_IRQ:
            /* save R0-R3, because subroutines called from here might modify
               these registers without saving/restoring them. Save R4, R5
               because we modify them in this subroutine */
            PUSH     {R0-R5, LR}

            /* Read the ICCIAR from the CPU interface */
            LDR      R4, =MPCORE_GIC_CPUIF
            LDR      R5, [R4, #ICCIAR]          // read the interrupt ID

PRIV_TIMER_CHECK:
            CMP      R5, #MPCORE_PRIV_TIMER_IRQ // check for HPS timer interrupt
            BNE      KEYS_CHECK

            BL       PRIV_TIMER_ISR
            B        EXIT_IRQ

KEYS_CHECK:
            CMP      R5, #KEYS_IRQ
UNEXPECTED: BNE      UNEXPECTED                 // if not recognized, stop here

            BL       KEY_ISR
EXIT_IRQ:
            /* Write to the End of Interrupt Register (ICCEOIR) */
            STR      R5, [R4, #ICCEOIR]

            POP      {R0-R5, LR}
            SUBS     PC, LR, #4
```

Save
Read Int. ID
29
73
clear int from GIC
return to main program

GIC Registers used above

Address	31 ··· 10	9 8	7 ··· 1	0	Register name
0xFFFEC100	Unused			E	ICCICR
0xFFFEC104	Unused		Priority		ICCPMR
0xFFFEC10C	Unused		Interrupt ID		ICCIAR
0xFFFEC110	Unused		Interrupt ID		ICCEOIR

ICCIAR: Who caused the interrupt?
ICCEOIR: Clear interrupt

Example of an ISR:

```
KEY_ISR:    LDR     R0, =0xFF200050     // base address of KEYs parallel port
            LDR     R1, [R0, #0xC]      // read edge capture register
            STR     R1, [R0, #0xC]      // clear the interrupt

CHK_KEY3:   TST     R1, #0b1000         // KEY 3 pressed?
            BEQ     END_KEY_ISR

            LDR     R0, =0xFFFEC600     // timer base address
            LDR     R1, [R0, #0x8]      // read timer control register
            EOR     R1, R1, #1          // toggle the enable bit
            STR     R1, [R0, #0x8]      // write to the timer control register

END_KEY_ISR:    MOV   PC, LR
```

STOPPED?
xor "E" bit (to stop/start timer)