Operating systems and concurrency B03

David Kendall

Northumbria University

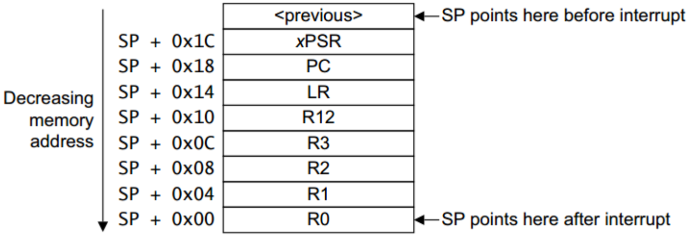
Introduction

- A key function of OS is *interrupt handling*
- We will build understanding of interrupt handling by looking in detail at:
  - Installing and executing an interrupt handler (ISR)
  - Configuring a timer as an interrupt source
- Example microcontroller – NXP LPC4088

Interrupt entry

Recall from the last lecture how the ARM Cortex M3 manages an IRQ interrupt

- Microcontroller peripheral raises interrupt;
- NVIC causes ISR vector to be fetched from vector table and put into R15 (PC);
- At same time, CPU pushes key registers onto the stack and stores special ‘return code’ in link register (R14/LR)

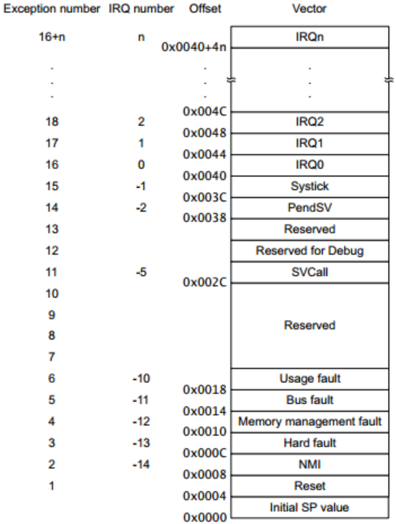


	<previous>
SP + 0x1C	*x*PSR
SP + 0x18	PC
SP + 0x14	LR
SP + 0x10	R12
SP + 0x0C	R3
SP + 0x08	R2
SP + 0x04	R1
SP + 0x00	R0

SP points here before interrupt
Decreasing memory address
SP points here after interrupt

- If ISR needs to save more context, it must do so itself

ARM Cortex M3 Vector Table



Exception number	IRQ number	Offset	Vector
16+n	n	0x0040+4n	IRQn
.		.	.
.		.	.
.		.	.
		0x004C	
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

ARM, Cortex-M3 Devices Generic User Guide, ARM 2010 (p.2-24)

LPC4088 Vector Table – details

```
; Vector Table Mapped to Address 0 at Reset
                  AREA     RESET, DATA, READONLY
                  EXPORT   __Vectors
__Vectors DCD __initial_sp        ; Top of Stack
  DCD Reset_Handler        ; Reset Handler
           DCD NMI_Handler         ; NMI Handler
           DCD HardFault_Handler   ; Hard Fault Handler
  DCD MemManage_Handler    ; MPU Fault Handler
  DCD BusFault_Handler     ; Bus Fault Handler
  DCD UsageFault_Handler   ; Usage Fault Handler
           DCD 0xEFFFF39E          ; Reserved− vector sum
  DCD 0                    ; Reserved
  DCD 0                    ; Reserved
  DCD 0                    ; Reserved
  DCD SVC_Handler          ; SVCall Handler
  DCD DebugMon_Handler     ; Debug Monitor Handler
  DCD 0                    ; Reserved
  DCD PendSV_Handler       ; PendSV Handler
  DCD SysTick_Handler      ; SysTick Handler
           ; External Interrupts
  DCD WDT_IRQHandler       ; 16: Watchdog Timer
  DCD TIMER0_IRQHandler    ; 17: Timer0

                    AREA |.text|, CODE, READONLY

Default_Handler PROC
           ...
           EXPORT TIMER0_IRQHandler    [WEAK]
           ...
TIMER0_IRQHandler
           B .
           ENDP
```

In the Keil MDK-ARM tools, this code appears in the file `startup_LPC407x_8x_177x_8x.s`

LPC4088 Vector Table - notes

- The startup file gives default entries for all elements in the vector table
- For the external interrupt handlers, like `TIMER0_IRQHandler`, the code associated with the handler is just a simple empty loop (i.e. it does nothing except loop back to itself)
- The address of this handler is stored in its slot in the vector table and exported to the rest of the program as a WEAK symbol - this means that it can be overwritten by our own handler, using the same name.

Installing our own interrupt handler

- So, to install our own handler for any interrupt, we look in the file `startup_LPC407x_8x_177x_8x.s` at the vector table to find the name of the handler function, e.g. `TIMER0_IRQHandler`. We then write our own C function with the same name, e.g.

```
void TIMER0_IRQHandler() {
...
}
```

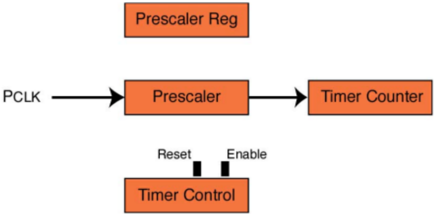
- The interrupt must also be enabled in the NVIC. We can use a predefined function to do that . . .

```
NVIC_EnableIRQ(TIMER0_IRQn);
```

Now that we know how to install an interrupt handler, let's see how to get one of the peripheral devices to generate an interrupt for us to handle . . .

LPC408x/7x Timers

- Four timers all with same structure
- Default clock source is APB peripheral clock (PCLK)
- Prescaler increments on each PCLK tick
- When prescaler value is equal to value in prescaler register, timer counter is incremented by 1 and prescaler is reset

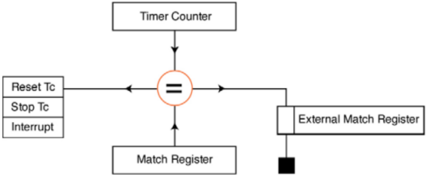


Timer modes

- Timers can be used in
  - capture mode
  - counter mode
  - match mode
- When used in match mode, the timer can be used to trigger some event when the value in the timer counter matches some preset value
- event can be a timer action (reset, stop, interrupt) or external action (e.g. set, clear, reset pin)
- match mode details to follow

Timer match mode

- Each timer has up to four match channels
- Each match channel has a match register containing 32-bit number
- When current value of timer counter matches value in match register an event is triggered



Some Timer0 Registers

Name	Function	Notes
TCR	Timer Control	Bit 0: 0 disables counter, 1 enables; Bit 1: 0 counter runs freely, 1 counter is reset
PR	Prescale	Value here controls when timer counter is incremented based on PCLK
CTCR	Count Control	Bits 0:1, 00 selects timer mode
MR0	Match	Write value here to be matched in order to cause event
IR	Interrupt	Writing 1 resets interrupt; writing 0 has no effect
MCR	Match Control	Bits 0:1, 11 causes interrupt and reset of counter on match event

- see LPC408x User Manual Chapter 24 for details of timer registers

How to initialise TIMER0

```
void timer0Init(uint32_t tickHz) {
  LPC_SC->PCONP |= (1UL << 1); /* ensure power to TIMER0 */
  LPC_TIM0->TCR = 0; /* disable the timer during configuration */
  LPC_TIM0->PR = 0; /* don't scale peripheral clock */
  LPC_TIM0->CTCR = 0; /* select timer mode, not counter mode */
  LPC_TIM0->MR0 = PeripheralClock / tickHz - 1; /* set match register */
  LPC_TIM0->MCR = 0x03UL; /* interrupt and reset on match */
  LPC_TIM0->IR = 0x3F; /* reset all TIMER0 interrupts */
  NVIC_EnableIRQ(TIMER0_IRQn); /* enable the TIMER0 interrupt in the NVIC */
  LPC_TIM0->TCR |= (1UL << 0); /* start the timer */
}

void TIMER0_IRQHandler(void) {
  /*
   * do whatever you want to handle the interrupt
   * e.g. flash an LED, execute an OS scheduler etc.
   */
  LPC_TIM0->IR |= (1UL << 0); /* clear the interrupt on MR0 */
}
```

- Notice that we're using the standard CMSIS register names here.
- More on CMSIS next time.

Acknowledgements

- Trevor Martin, The Designer's Guide to the Cortex-M Processor Family: A Tutorial Approach, Newnes, 2013