Exceptions | 12.3 NVIC on the ARM Cortex-... https://courses.edx.org/courses/UTAustinX/UT...

edX (https://www.edx.org)

UTAustinX: UT.6.01x Embedded Systems - Shape the World

KarenWest (/dashboard) ▼

Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware) Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)

Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum) Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)

Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)

Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)

Embedded Systems Community (/courses/UTAustinX/UT.6.01x/1T2014/e3df91316c544d3e8e21944fde3ed46c/)

Help

On the ARMCortex-M processor, **exceptions** include resets, software interrupts and hardware interrupts. Interrupts on the Cortex-M are controlled by the Nested Vectored Interrupt Controller (NVIC). Each exception has an associated 32-bit vector that points to the memory location where the ISR that handles the exception is located. Vectors are stored in ROM at the beginning of memory. Program 12.1 shows the first few vectors as defined in the **Startup.s** file. **`DCD`** is an assembler pseudo-op that defines a 32-bit constant. ROM location 0x0000.0000 has the initial stack pointer, and location 0x0000.0004 contains the initial program counter, which is also called the **reset vector**. It points to a function called the reset handler, which is the first thing executed following reset. In C, the reset handler initializes global variables and then calls your **`main()`** program. There are up to 240 possible interrupt sources and their 32-bit vectors are listed in order starting with location 0x0000.0008. From a programming perspective, we can attach ISRs to interrupts by writing the ISRs as regular assembly subroutines or C functions with no input or output parameters and editing the **Startup.s** file to specify those functions for the appropriate interrupt. For example, if we wrote a Port F interrupt service routine named **`PortFISR`**, then we would replace **`GPIOPortF_Handler`** with **`PortFISR`**. In this class, we will write our ISRs using standard function names so that the **Startup.s** file need not be edited. I.e., we will simply name the ISR for edge-triggered interrupts on Port F as **`GPIOPortF_Handler`**. The vector for this interrupt is a 32-bit pointer located at ROM address 0x0000.00B8. Because the vectors are in ROM, this linkage is defined at compile time and not at run time. For more details see the **Startup.s** files within the interrupt examples included as part of the TExaS installation.

CHECKPOINT 12.4

Where is the vector for SysTick? What is the standard name for this ISR?

Hide Answer

From Program 12.1 or Table 12.1 we see the vector is 32 bits at 0x0000003C. The standard name of the interrupt handler is SysTick_Handler.

```
   EXPORT  __Vectors
__Vectors                        ; address    interrupt
   DCD     StackMem + Stack      ; 0x00000000 Top of Stack
   DCD     Reset_Handler         ; 0x00000004 Reset Handler
   DCD     NMI_Handler           ; 0x00000008 NMI Handler
   DCD     HardFault_Handler     ; 0x0000000C Hard Fault Handler
   DCD     MemManage_Handler     ; 0x00000010 MPU Fault Handler
   DCD     BusFault_Handler      ; 0x00000014 Bus Fault Handler
   DCD     UsageFault_Handler    ; 0x00000018 Usage Fault Handler
```

```
    DCD     0                      ; 0x0000001C Reserved
    DCD     0                      ; 0x00000020 Reserved
    DCD     0                      ; 0x00000024 Reserved
    DCD     0                      ; 0x00000028 Reserved
    DCD     SVC_Handler            ; 0x0000002C SVCall Handler
    DCD     DebugMon_Handler       ; 0x00000030 Debug Monitor Handler
    DCD     0                      ; 0x00000034 Reserved
    DCD     PendSV_Handler         ; 0x00000038 PendSV Handler
    DCD     SysTick_Handler        ; 0x0000003C SysTick Handler
    DCD     GPIOPortA_Handler      ; 0x00000040 GPIO Port A
    DCD     GPIOPortB_Handler      ; 0x00000044 GPIO Port B
    DCD     GPIOPortC_Handler      ; 0x00000048 GPIO Port C
    DCD     GPIOPortD_Handler      ; 0x0000004C GPIO Port D
    DCD     GPIOPortE_Handler      ; 0x00000050 GPIO Port E
    DCD     UART0_Handler          ; 0x00000054 UART0
    DCD     UART1_Handler          ; 0x00000058 UART1
    DCD     SSI0_Handler           ; 0x0000005C SSI
    DCD     I2C0_Handler           ; 0x00000060 I2C
    DCD     PWM0Fault_Handler      ; 0x00000064 PWM Fault
    DCD     PWM0Generator0_Handler  ; 0x00000068 PWM 0 Generator 0
    DCD     PWM0Generator1_Handler  ; 0x0000006C PWM 0 Generator 1
    DCD     PWM0Generator2_Handler  ; 0x00000070 PWM 0 Generator 2
    DCD     Quadrature0_Handler    ; 0x00000074 Quadrature Encoder 0
    DCD     ADC0Seq0_Handler       ; 0x00000078 ADC0 Sequence 0
    DCD     ADC0Seq1_Handler       ; 0x0000007C ADC0 Sequence 1
    DCD     ADC0Seq2_Handler       ; 0x00000080 ADC0 Sequence 2
    DCD     ADC0Seq3_Handler       ; 0x00000084 ADC0 Sequence 3
    DCD     WDT_Handler            ; 0x00000088 Watchdog
    DCD     Timer0A_Handler        ; 0x0000008C Timer 0 subtimer A
    DCD     Timer0B_Handler        ; 0x00000090 Timer 0 subtimer B
    DCD     Timer1A_Handler        ; 0x00000094 Timer 1 subtimer A
    DCD     Timer1B_Handler        ; 0x00000098 Timer 1 subtimer B
    DCD     Timer2A_Handler        ; 0x0000009C Timer 2 subtimer A
    DCD     Timer2B_Handler        ; 0x000000A0 Timer 2 subtimer B
    DCD     Comp0_Handler          ; 0x000000A4 Analog Comp 0
    DCD     Comp1_Handler          ; 0x000000A8 Analog Comp 1
    DCD     Comp2_Handler          ; 0x000000AC Analog Comp 2
    DCD     SysCtl_Handler         ; 0x000000B0 System Control
    DCD     FlashCtl_Handler       ; 0x000000B4 Flash Control
    DCD     GPIOPortF_Handler      ; 0x000000B8 GPIO Port F
```

Program 12.1. Software syntax to set the interrupt vectors for the TM4C (only some vectors are shown, see the startup.s file for a complete list).

Program 12.2 shows that the syntax for an ISR looks like a function with no parameters. Notice that each ISR (except for SysTick) must acknowledge the interrupt in software by clearing the flag that caused the interrupt. In Program 12.2, we assume the interrupt was caused by an edge on PF4, so writing to the ICR register will clear trigger flag 4.

```
void GPIOPortF_Handler(void){
  GPIO_PORTF_ICR_R = 0x10; // ack, clear interrupt flag4
  // stuff
}
```

Program 12.2. Typical interrupt service routine.