Interrupt-Driven Input/Output on the STM32F407 Microcontroller

Textbook: Chapter 11 (Interrupts)

ARM Cortex-M4 User Guide (Interrupts, exceptions, NVIC)
  Sections 2.1.4, 2.3 – Exceptions and interrupts
  Section 4.2 – Nested Vectored Interrupt Controller

STM32F4xx Tech. Ref. Manual:
  Chapter 8: External interrupt/wakeup lines
  Chapter 9: SYSCFG external interrupt config. registers
Outline

- Interrupt vectors and vector table
- Interrupt masks and priorities
- Cortex Nested Vectored Interrupt Controller (NVIC)
- STM32F4 external interrupt signals (EXTI0 – EXTI15)
- System design when interrupts used
Prioritized, vectored interrupts

Interrupt vectors determine what function is executed for each type of interrupt request.

- Vector = address of interrupt handler
- Vectors arranged by interrupt # in the “Vector Table”

Priorities determine what interrupt gets the CPU first.
Interrupt vectors

- **Interrupt vector** = *address* of handler function
  - Allow different devices to be handled by different code.

- **Interrupt vector table:**
  - Directly supported by CPU architecture and/or
  - Supported by a separate interrupt-support device/function

```
Interrupt vector table
    head
    address of handler 0
    address of handler 1
    address of handler 2
    address of handler 3
```
## Cortex-M CPU and peripheral exceptions

<table>
<thead>
<tr>
<th></th>
<th>Priority $^1$</th>
<th>IRQ$^2$</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>-3/fixed</td>
<td></td>
<td>Power-up or warm reset</td>
</tr>
<tr>
<td>NMI</td>
<td>-2/fixed</td>
<td>-14</td>
<td>Non-maskable interrupt from peripheral or software</td>
</tr>
<tr>
<td>HardFault</td>
<td>-1/fixed</td>
<td>-13</td>
<td>Error during exception processing or no other handler</td>
</tr>
<tr>
<td>MemManage</td>
<td>0/settable</td>
<td>-12</td>
<td>Memory protection fault (MPU-detected)</td>
</tr>
<tr>
<td>BusFault</td>
<td>1/settable</td>
<td>-11</td>
<td>AHB data/prefetch aborts</td>
</tr>
<tr>
<td>UsageFault</td>
<td>2/settable</td>
<td>-10</td>
<td>Instruction execution fault - undefined instruction, illegal</td>
</tr>
<tr>
<td>UsageFault</td>
<td>2/settable</td>
<td>-10</td>
<td>Instruction execution fault - undefined instruction, illegal</td>
</tr>
<tr>
<td>UsageFault</td>
<td>2/settable</td>
<td>-10</td>
<td>Instruction execution fault - undefined instruction, illegal</td>
</tr>
<tr>
<td>SVCcall</td>
<td>3/settable</td>
<td>-5</td>
<td>System service call (SVC) instruction</td>
</tr>
<tr>
<td>DebugMonitor</td>
<td>4/settable</td>
<td></td>
<td>Break points/watch points/etc.</td>
</tr>
<tr>
<td>PendSV</td>
<td>5/settable</td>
<td>-2</td>
<td>Interrupt-driven request for system service</td>
</tr>
<tr>
<td>SysTick</td>
<td>6/settable</td>
<td>-1</td>
<td>System tick timer reaches 0</td>
</tr>
<tr>
<td>IRQ0</td>
<td>7/settable</td>
<td>0</td>
<td>Signaled by peripheral or by software request</td>
</tr>
<tr>
<td>IRQ1 (etc.)</td>
<td>8/settable</td>
<td>1</td>
<td>Signaled by peripheral or by software request</td>
</tr>
</tbody>
</table>

$^1$ Lowest priority # = highest priority  
$^2$ IRQ# used in CMSIS function calls

---

**Vendor peripheral interrupts**  
IRQ0 .. IRQ44
## STM32F4xx Peripherals: Interrupt Vector Table

### Tech. Ref. Manual: Table 43

Also - refer to startup code

<table>
<thead>
<tr>
<th>Position</th>
<th>Priority</th>
<th>Type of priority</th>
<th>Acronym</th>
<th>Description</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>settable</td>
<td>WWDG</td>
<td>Window Watchdog interrupt</td>
<td>0x0000 0040</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>settable</td>
<td>PVD</td>
<td>PVD through EXTI line detection interrupt</td>
<td>0x0000 0044</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>settable</td>
<td>TMP_STAMP</td>
<td>Tamper and TimeStamp interrupts through the EXTI line</td>
<td>0x0000 0048</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>settable</td>
<td>RTC_WKUP</td>
<td>RTC Wakeup interrupt through the EXTI line</td>
<td>0x0000 004C</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>settable</td>
<td>FLASH</td>
<td>Flash global interrupt</td>
<td>0x0000 0050</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>settable</td>
<td>RCC</td>
<td>RCC global interrupt</td>
<td>0x0000 0054</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>settable</td>
<td>EXTI0</td>
<td>EXTI Line0 interrupt</td>
<td>0x0000 0058</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>settable</td>
<td>EXTI1</td>
<td>EXTI Line1 interrupt</td>
<td>0x0000 005C</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>settable</td>
<td>EXTI2</td>
<td>EXTI Line2 interrupt</td>
<td>0x0000 0060</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>settable</td>
<td>EXTI3</td>
<td>EXTI Line3 interrupt</td>
<td>0x0000 0064</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>settable</td>
<td>EXTI4</td>
<td>EXTI Line4 interrupt</td>
<td>0x0000 0068</td>
</tr>
<tr>
<td>11</td>
<td>18</td>
<td>settable</td>
<td>DMA1_Stream0</td>
<td>DMA1 Stream0 global interrupt</td>
<td>0x0000 006C</td>
</tr>
</tbody>
</table>

| 26       | 33       | settable         | TIM1_TRG_COM_TIM11 | TIM1 Trigger and Commutation interrupts and TIM11 global interrupt | 0x0000 00A8 |
| 27       | 34       | settable         | TIM1_CC           | TIM1 Capture Compare interrupt                         | 0x0000 00AC |
| 28       | 35       | settable         | TIM2              | TIM2 global interrupt                                  | 0x0000 00B0 |
| 29       | 36       | settable         | TIM3              | TIM3 global interrupt                                  | 0x0000 00B4 |
| 30       | 37       | settable         | TIM4              | TIM4 global interrupt                                  | 0x0000 00B8 |
| 31       | 38       | settable         | I2C1_EV           | I2C1 event interrupt                                  | 0x0000 00BC |
| 32       | 39       | settable         | I2C1_ER           | I2C1 error interrupt                                  | 0x0000 00C0 |
| 33       | 40       | settable         | I2C2_EV           | I2C2 event interrupt                                  | 0x0000 00C4 |
| 34       | 41       | settable         | I2C2_ER           | I2C2 error interrupt                                  | 0x0000 00C8 |
| 35       | 42       | settable         | SPI1              | SPI1 global interrupt                                  | 0x0000 00CC |

**External interrupts**

**Timer interrupts**
## STM32F4 vector table from startup code (partial)

### Vectors

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCD</td>
<td>__initial_sp</td>
<td>Top of Stack</td>
</tr>
<tr>
<td>DCD</td>
<td>Reset_Handler</td>
<td>Reset Handler</td>
</tr>
<tr>
<td>DCD</td>
<td>NMI_Handler</td>
<td>NMI Handler</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCD</td>
<td>SVC_Handler</td>
<td>SVCall Handler</td>
</tr>
<tr>
<td>DCD</td>
<td>DebugMon_Handler</td>
<td>Debug Monitor Handler</td>
</tr>
<tr>
<td>DCD</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>DCD</td>
<td>PendSV_Handler</td>
<td>PendSV Handler</td>
</tr>
<tr>
<td>DCD</td>
<td>SysTick_Handler</td>
<td>SysTick Handler</td>
</tr>
</tbody>
</table>

### External Interrupts

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCD</td>
<td>WWDG_IRQHandler</td>
<td>Window WatchDog</td>
</tr>
<tr>
<td>DCD</td>
<td>PVD_IRQHandler</td>
<td>PVD via EXTI Line detection</td>
</tr>
<tr>
<td>DCD</td>
<td>TAMP_STAMP_IRQHandler</td>
<td>Tamper/TimeStamps via EXTI</td>
</tr>
<tr>
<td>DCD</td>
<td>RTC_WKUP_IRQHandler</td>
<td>RTC Wakeup via EXTI line</td>
</tr>
<tr>
<td>DCD</td>
<td>FLASH_IRQHandler</td>
<td>FLASH</td>
</tr>
<tr>
<td>DCD</td>
<td>RCC_IRQHandler</td>
<td>RCC</td>
</tr>
<tr>
<td>DCD</td>
<td>EXTI0_IRQHandler</td>
<td>EXTI Line0</td>
</tr>
<tr>
<td>DCD</td>
<td>EXTI1_IRQHandler</td>
<td>EXTI Line1</td>
</tr>
<tr>
<td>DCD</td>
<td>EXTI2_IRQHandler</td>
<td>EXTI Line2</td>
</tr>
<tr>
<td>DCD</td>
<td>EXTI3_IRQHandler</td>
<td>EXTI Line3</td>
</tr>
</tbody>
</table>
Prioritized interrupts

- Up to 256 priority levels
- 8-bit priority value
- Implementations may use fewer bits
  STM32F4xx uses upper 4 bits of each priority byte => 16 levels
- NMI & HardFault priorities are fixed
- Lowest # = Highest priority
Special CPU registers

Special Cortex-M Assembly Language Instructions

- **CPSIE I** ; Change Processor State/Enable Interrupts (sets PRIMASK = 0)
- **CPSID I** ; Change Processor State/Disable Interrupts (sets PRIMASK = 1)

CMSIS\(^1\) C functions to clear/set PRIMASK

- `__enable_irq(); // enable interrupts (set PRIMASK=0)`
- `__disable_irq(); // disable interrupts (set PRIMASK=1)`

(double-underscore at beginning)

Prioritized Interrupts Mask Register (PRIMASK)

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>PRIMASK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PRIMASK = 1 prevents (masks) activation of all exceptions with configurable priority
PRIMASK = 0 permits (enables/unmasks) exceptions

Processor Status Register (PSR)

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Z</td>
<td>C</td>
<td>V</td>
<td>Q</td>
<td>Reserved</td>
<td>ICI/IT</td>
<td>ISR_NUMBER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) Cortex Microcontroller Software Interface Standard – Functions for all ARM Cortex-M CPUs, defined in project header files: `core_cmFunc.h, core_cm3.h`
## Interrupt Program Status Register (ISPR)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits 31:9</td>
<td>Reserved</td>
</tr>
<tr>
<td>Bits 8:0</td>
<td>ISR_NUMBER:</td>
</tr>
</tbody>
</table>

This is the number of the current exception:

- 0: Thread mode
- 1: Reserved
- 2: NMI
- 3: Hard fault
- 4: Memory management fault
- 5: Bus fault
- 6: Usage fault
- 7: Reserved
- ....
- 10: Reserved
- 11: SVCall
- 12: Reserved for Debug
- 13: Reserved
- 14: PendSV
- 15: SysTick
- 16: IRQ0(1)
- ....

**Cortex CPU interrupts**

- No active interrupt

**User (vendor) interrupts IRQ0 – IRQ239**
ARM Cortex-M Interrupts

In the Device:

- Each potential interrupt source has a separate **arm (enable)** bit
  - Set for those devices from which interrupts, are to be accepted
  - Deactivate in those devices from which interrupts are not allowed
- Each potential interrupt source has a separate **flag** bit
  - hardware sets the flag when it wishes to request an interrupt (an “event” occurs)
  - software clears the flag in ISR to signify it is processing the request
  - flags can be tested by software if interrupts not desired

In the CPU:

- Cortex-M CPUs receive interrupt requests via the **Nested Vectored Interrupt Controller (NVIC)**
  - NVIC sends highest priority request to the CPU
- Interrupt **enable** conditions in processor
  - Global interrupt enable bit, I, in PRIMASK register
  - Priority level, BASEPRI, of allowed interrupts (0 = all)
Interrupt Conditions

1. **Trigger**: hardware action sets source-specific flag in the peripheral device
2. **Arm**: control bit for each possible source is set within the peripheral device
3. **Level**: interrupt level must be less than BASEPRI (base priority)
4. **Enable**: interrupts globally enabled in CPU (I=0 in PRIMASK)

Interrupt remains **pending** if trigger is set but any other condition is not true

- Interrupt serviced once all conditions become true

**Need to acknowledge** interrupt
- Clear trigger flag to prevent endless interrupts!
Nested Vectored Interrupt Controller

- NVIC manages and prioritizes external interrupts in Cortex-M
  - 82 IRQ sources from STM32F4xx peripherals
- NVIC interrupts CPU with IRQ# of highest-priority IRQ signal
  - CPU uses IRQ# to access the vector table & get intr. handler start address
Nested Vectored Interrupt Controller (NVIC)

- Hardware unit that coordinates interrupts from multiple sources
  - Separate **enable flag** for each interrupt source
    - Set/clear via **NVIC_ISERx/ICERx** registers
  - Separate **priority level** for each interrupt source
    - Define in **NVIC_IPRx** registers
  - Set/clear interrupts to/from **pending state**
    - Pending state entered when interrupt request detected
    - Pending state **cleared automatically** when ISR complete, unless subsequent interrupt request detected while in the ISR
    - Manually set/clear pending state via **NVIC_ISPRx/ICPRx** registers
    - Can also trigger interrupts through software if desired

- Higher priority interrupts can interrupt lower priority ones
  - Lower priority interrupts are not sent to the CPU until higher priority interrupt service has been completed
Nested Vectored Interrupt Controller (NVIC)

- Each interrupt source is in one of four states
  - **Inactive** – no interrupt service requested or in progress
  - **Pending** – interrupt request latched by NVIC; not yet serviced by CPU
    - Become pending if interrupt signal HIGH and interrupt not active
    - Also become pending if rising edge detected on interrupt signal
  - **Active** – interrupt service by CPU is in progress
    - State changes from Pending to Active when CPU enters the ISR
    - State changes from Active to Inactive when CPU exits the ISR, unless:
      - State changes from Active to Pending if interrupt signal still HIGH when CPU exits the ISR or if state is “Pending and Active” (can re-enter the ISR)
  - **Pending and Active** – new interrupt request detected (rising edge or pulse) while CPU is servicing a previous request (IRQ can be re-entered)
NVIC Interrupt Enable Registers

- Three “set interrupt enable” registers – 
  \texttt{NVIC\_ISER0} , \texttt{NVIC\_ISER1} , \texttt{NVIC\_ISER2}
- One “enable” bit per IRQ - with 32 per register
- Write 1 to a bit to set the corresponding interrupt enable bit
- \textbf{Writing 0 has no effect}

<table>
<thead>
<tr>
<th>Registers</th>
<th>IRQ numbers</th>
<th>Interrupt numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIC_ISER0/NVIC_ICER0</td>
<td>0-31</td>
<td>16-47</td>
</tr>
<tr>
<td>NVIC_ISER1/NVIC_ICER1</td>
<td>32-63</td>
<td>48-79</td>
</tr>
<tr>
<td>NVIC_ISER2/NVIC_ICER2</td>
<td>64-95</td>
<td>80-111</td>
</tr>
</tbody>
</table>

- Three corresponding “clear interrupt enable” registers
  \texttt{NVIC\_ICER0} , \texttt{NVIC\_ICER1} , \texttt{NVIC\_ICER2}
- Write 1 to clear the interrupt enable bit (disable the interrupt)
- \textbf{Writing 0 has no effect}
NVIC Set-Clear Pending Registers

- Three interrupt “clear-pending” registers –
  \textbf{NVIC\_ICPR0, NVIC\_ICPR1, NVIC\_ICPR2}
  - One “pending” flag for each IRQ (32 in each register)
  - Write 1 to a bit to remove a pending state from an interrupt
  - Writing 0 has no effect

- Pending state normally \textit{cleared automatically} when the processor enters the interrupt handler (ISR)
  - If interrupt signal still active when CPU returns from the ISR, the state \textit{changes to pending again} (new interrupt triggered)
  - If interrupt signal pulses while CPU is in the ISR, the state \textit{changes to pending again} (new interrupt triggered)

- Three corresponding interrupt “set-pending” registers
  \textbf{NVIC\_ISPR0, NVIC\_ISPR1, NVIC\_ISPR2}
  - Write 1 to force the interrupt state to pending
  - Writing 0 has no effect
Interrupt Priority Registers NVIC_IPRx (x=0..20)

- 8-bit priority field for each interrupts (4-bit field in STM32F4)
  - Four 8-bit priority values per register
  - STMicroelectronics uses upper 4 bits of each byte
  - 0 = highest priority level
  - IPR Register# x = IRQ# DIV 4
  - Byte offset within the IPR register = IRQ# MOD 4

Example: IRQ45

- 45/4 = 11 with remainder 1 (register NVIC_IPR11, byte offset 1)
  - Write priority<<8 to NVIC_IPR11
- 45/32 = 1 with remainder 13:
  - Write 1<<13 to NVIC_ISER1 to enable the interrupt
NVIC register addresses

NVIC_ISER0/1/2 = 0xE000E100/104/108
NVIC_ICER0/1/2 = 0xE000E180/184/188
NVIC_ISPR0/1/2 = 0xE000E200/204/208
NVIC_ICPR0/1/2 = 0xE000E280/284/288
NVIC_IPR0/1/2/…/20 = 0xE00E400/404/408/40C/…./500

;Example – Enable EXTI0 with priority 5 (EXTI0 = IRQ6)
NVIC_ISER0  EQU  0xE000E100          ;bit 6 enables EXTI0
NVIC_IPR1    EQU  0xE000E404 ;3rd byte = EXTI0 priority

ldr  r0,=NVIC_ISER0
mov  r1,#0x0040 ;Set bit 6 of ISER0 for EXTI0
str  r1,[r0]
ldr  r0,=NVIC_IPR1
    ;IRQ6 priority in IPR1[23:16]
ldr  r1,[r0] ;Read IPR1
bic  r1,#0x00ff0000 ;Clear [23:16] for IRQ6
orr  r1,#0x00500000 ;Bits [23:20] = 5
str  r1,[r0] ;Upper 4 bits of byte = priority
CMSIS\textsuperscript{1} functions

- NVIC_Enable(IRQn\_Type IRQn)
- NVIC_Disable(IRQn\_Type IRQn)
- NVIC_SetPending(IRQn\_Type IRQn)
- NVIC_ClearPending(IRQn\_Type IRQn)
- NVIC_GetPending(IRQn\_Type IRQn)
- NVIC_SetPriority(IRQn\_Type IRQn,unit32\_t priority)
- NVIC_GetPriority(IRQn\_Type IRQn)

\textsuperscript{1}CMSIS = Cortex Microcontroller Software Interface Standard
- Vendor-independent hardware abstraction layer for Cortex-M
- Facilitates software reuse
- Other CMSIS functions: System tick timer, Debug interface, etc.
STM32F4xx external interrupt/event controller

- External devices can interrupt CPU via GPIO pins
  (Some microcontrollers have dedicated interrupt pins)
- Up to 16 external interrupts (EXTI0-EXTI15), plus 7 internal events

Diagram:

- AMBAAPB bus
- PCLK2
- Peripheral interface
- Pending request register
- Interrupt mask register
- Software interrupt event register
- Rising trigger selection register
- Falling trigger selection register
- Edge detect circuit
- External interrupt signal (GPIO pin)
STM32F4xx external interrupt sources
(select in System Configuration Module – SYSCFG)

- 16 multiplexers select GPIO pins as external interrupts EXTI0..EXTI15
- Mux inputs selected via 4-bit fields of EXTICR[k] registers (k=0..3)
  - EXTIx = 0 selects PAx, 1 selects PBx, 2 selects PCx, etc.
  - EXTICR[0] selects EXTI3-EXTI0; EXTICR[1] selects EXTI7-EXTI4, etc.

Example: Select pin PC2 as external interrupt EXTI2

```c
SYSCFG->EXTICR[0] &= 0xF0FF; //clear EXTI2 bit field
SYSCFG->EXTICR[0] |= 0x0200; //set EXTI2 = 2 to select PC2
```
STM32L1xx EXTI configuration registers

- Register bits 15-0 control EXTI15-EXTI0, respectively
- **EXTI_IMR** – interrupt mask register
  - 1 unmasks (enables) the corresponding interrupt
  - 0 masks (disables) the interrupt
- **EXTI_RTSR/FTSR** – rising/falling trigger selection register
  - 1 to enable rising/falling edge to trigger the interrupt/event
  - 0 to ignore the rising/falling edge
- **EXTI_PR** – interrupt pending register
  - bit set to 1 by hardware if interrupt/event occurred *(bit is readable)*
  - clear bit by writing 1 *(writing 0 has no effect)*
  - **interrupt handler** must write 1 to this bit to clear the pending state of the interrupt (to cancel the IRQn request)

**Example:** Configure EXTI2 as rising-edge triggered

```c
EXTI->RTSR |= 0x0004;  // Bit2=1 to make EXTI2 rising-edge trig.
EXTI->IMR     = 0x0004;  // Bit2=1 to enable EXTI2
EXTI->PR     |= 0x0004;  // Bit2=1 to clear EXTI2 pending status
```

Clearing pending status needs to be done in the interrupt handler after every interrupt.
Example: Enable EXTI0 as rising-edge triggered, selecting PC0 as EXTI0.

;System Configuration Registers
SYSCFG EQU 0x40013800
EXTICR1 EQU 0x08

;External Interrupt Registers
EXTI EQU 0x40013C00
IMR EQU 0x00 ;Interrupt Mask Register
RTSR EQU 0x08 ;Rising Trigger Select
FTSR EQU 0x0C ;Falling Trigger Select
PR EQU 0x14 ;Pending Register

;select PC0 as EXTI0
ldr r1,=SYSCFG ;SYSCFG selects EXTI sources
ldrh r2,[r1,#EXTICR1] ;EXTICR1 = sources for EXTI0 - EXTI3
bic r2,#0x000f ;Clear EXTICR1[3-0] for EXTI0 source
orr r2,#0x0002 ;EXTICR1[3-0] = 2 to select PC0 as EXTI0 source
strh r2,[r1,#EXTICR1] ;Write to select PC0 as EXTI0

;configure and enable EXTI0 as rising-edge triggered
ldr r1,=EXTI ;EXTI register block
mov r2,#1 ;bit #0 for EXTI0 in each of the following registers
str r2,[r1,#RTSR] ;Select rising-edge trigger for EXTI0
str r2,[r1,#PR] ;Clear any pending event on EXTI0
str r2,[r1,#IMR] ;Enable EXTI0
Interrupt Rituals

- Things you must do in every ritual
  - Initialize data structures (counters, pointers)
  - Arm interrupt in the peripheral device
    - Enable a flag to trigger an interrupt (ex. IMR of the EXTI module)
    - Clear the flag (to ignore any previous events) (ex. PR of the EXTI module)
  - Configure NVIC
    - Enable interrupt (set bit of NVIC_ISERx)
    - Set priority (set bits of NVIC_IPRx)
  - Enable CPU Interrupts
    - Assembly code: CPSIE I
    - C code: EnableInterrupts()
Project setup for interrupt-driven applications

- Write the interrupt handler for each peripheral
  - Clear the flag that requested the interrupt (acknowledge the intr. request)
  - Perform the desired actions, communicating with other functions via shared global variables
  - Use function names from the vector table
    Example: `void EXTI4_IRQHandler () { statements }`

- Perform all initialization for each peripheral device:
  - Initialize the device, “arm” its interrupt, and clear its “flag”
    Example: External interrupt EXTI
    - Configure GPIO pin as a digital input
    - Select the pin as the EXTI source (in SYSCFG module)
    - Enable interrupt to be requested when a flag is set by the desired event (rising/falling edge)
    - Clear the pending flag (to ignore any previous events)

- NVIC
  - Enable interrupt: `NVIC_EnableIRQ (IRQn);`
  - Set priority: `NVIC_SetPriority (IRQn, priority);`
  - Clear pending status: `NVIC_ClearPendingIRQ (IRQn);`

- Initialize counters, pointers, global variables, etc.
- Enable CPU Interrupts: `__enable_irq ();`

(diagram on next slide)
Signal flow/setup for External Interrupt EXTI\(x\), \(x = 0\ldots15\)

**SYSCFG module**
- Select source for EXTI\(x\)
- 4 bits each
- 0000 = P\(Ax\)
- 0001 = P\(Bx\)
- SYSCFG \(\rightarrow\) EXTI\(x\)

**GPIO module**
- Pin P\(Ax\)
- Configure P\(Ax\) as input (00)
- GPIOA \(\rightarrow\) MODER

**EXTI module**
- For all EXTI\(x\) registers: Bit \(n\) controls EXTI\(n\)
- EXTI \(\rightarrow\) IMR
- EXTI \(\rightarrow\) FTSR
- EXTI \(\rightarrow\) RTSR
- EXTI \(\rightarrow\) PR
- EXTI \(\rightarrow\) edge

**CPU**
- Enable
- NVIC
- NVIC_EnableIRQ(n); interrupt mask (enable)
- __enable_irq();
- __disable_irq());
- 1 of 45, passed by NVIC to CPU

**NVIC**
- Set when EXTI\(x\) activates
- **Clear** when intr. handler exits
- NVIC_ClearPendingIRQ(n); clear pending flag
- NVIC_SetPriority(intr. #, value); value = priority of intr. #
- set this interrupt
- select falling and/or rising edge
Interrupt Service Routine (ISR)

- Interrupt service routine (interrupt handler) name must be in the interrupt vector table (ex. EXTI0_IRQHandler)

- Things you must do in every interrupt service routine
  - Acknowledge
    - clear the flag that requested the interrupt (ex. PR of EXTI module)
    - SysTick is exception; automatic acknowledge
  - Save/restore contents of R4-R11, if used. (AAPCS)
  - Perform the requested service
  - Communicate via shared global variables
Sources of interrupt “overhead”

- Handler execution time.
- Interrupt mechanism overhead.
- Register save/restore.
- Pipeline-related penalties (advanced processors).
- Cache-related penalties (advanced processors).
- **Interrupt “latency”** = time from activation of interrupt signal until event serviced.
- ARM worst-case latency to respond to interrupt is 27 cycles:
  - 2 cycles to synchronize external request.
  - Up to 20 cycles to complete current instruction (worst are LDM/STM).
  - 3 cycles for data abort.
  - 2 cycles to enter interrupt handling state.