Interrupts

Dr. Farahmand

Updated: 3/14/18
Basic Concepts in Interrupts

- An interrupt is a communication process set up in a microprocessor or microcontroller in which:
  - An internal or external device requests the MPU to stop the processing
  - The MPU acknowledges the request
  - Attends to the request
  - Goes back to processing where it was interrupted
Polling vs Interrupt

Polling: You **pick up the phone every few seconds** to check whether you are getting a call.

Interrupt: Do whatever you should do and **pick up the phone when it rings**.

```c
// Polling method
while (1) {
    read_button_input;
    if (pushed)
        exit;
}

// Interrupt method
interrupt_handler(){
    turn_on_LED;
    exit;
}
```
Types of Interrupts (1 of 2)

- **Hardware interrupts**
  - Maskable: can be masked or disabled
  - Two groups: external and internal interrupts
  - Non-maskable: cannot be disabled

- **Software interrupts**: generally used when the situation requires stop processing and start all over
  - Examples: divide by zero or stack overflow
  - Generally, microcontrollers do not include software interrupts
Example of an Interrupt Circuit

How many INTs? Where are they?

Page 46 PIC18F4X2
When the interrupt process is enabled, the MPU, during execution, checks the interrupt request flag just before the end of each instruction.

If the interrupt request is present, the MPU:

- Completes the execution of the instruction
- Resets the interrupt flag
- Saves the address of the program counter on the stack
- Some interrupt processes also save contents of MPU registers on the stack.
- Stops the execution
To restart the execution, the MPU needs to be redirected to the memory location where the interrupt request can be met.
- “What to do if you receive the interrupt!”
- Accomplished by interrupt vectors

The set of instructions written to meet the request (or to accomplish the task) is called an interrupt service routine (ISR).

Once the request is accomplished, the MPU should find its way back to the instruction, next memory location where it was interrupted.
- Accomplished by a specific return instruction
Interrupt Service Routine (ISR)

- A group of instructions that accomplishes the task requested by the interrupting source
- Similar to a subroutine except that the ISR must be terminated in a Return instruction specially designed for interrupts
  - The Return instruction, when executed, finds the return address on the stack and redirects the program execution where the program was interrupted.
  - Some Return instructions are designed to retrieve the contents of MPU registers if saved as a part of the interrupts.
Memory Map of Cortex-M4

- **System**: 0xFFFFFFFF
  - NVIC, System Timer, SCB, vendor-specific memory

- **External RAM**: 0xE0000000
  - Off-chip memory for data

- **External Device**: 0xA0000000
  - Such as SD card

- **Peripheral**: 0x60000000
  - AHB & APB, such as timers, GPIO

- **SRAM**: 0x40000000
  - On-chip RAM, for heap, stack, & code

- **Code**: 0x20000000
  - On-chip Flash, for code & data

- **Code**: 0x00000000
  - One Byte (8 bits)

- **On-chip Flash**: 0x00000000
  - 0.5 GB

- **On-chip RAM**: 0x20000000
  - 0.5 GB

- **On-chip RAM**: 0x40000000
  - 0.5 GB

- **On-chip RAM**: 0x60000000
  - 0.5 GB

- **Off-chip Memory**: 0xE0000000
  - 1 GB

- **Off-chip Memory**: 0xA0000000
  - 1 GB
Data Memory

- **Code**: 0x00000000 - 0x20000000
  - 0.5 GB
- **SRAM**: 0x20000000 - 0x60000000
  - 0.5 GB
- **External RAM**: 0x60000000 - 0xA0000000
  - 0.5 GB
- **External Device**: 0xA0000000 - 0xE0000000
  - 1 GB
- **System**: 0xE0000000 - 0xFFFFFFFF
  - 0.5 GB
- **Internal SRAM Memory**: 0x20000000 - 0x3FFFFFFF
  - 96 KB
- **Zero-initialized data**: 0x20017FFF
- **Initialized data**: 0x20000000 - 0x20017FFF
- **Stack**: 0x20000000 - 0x3FFFFFFF
- **Heap**: 0x20017FFF - 0x3FFFFFFF

**Notes**:
- One Byte (8 bits)
Instruction Memory
## Interrupt Vector Table

When interrupt x is triggered, jump to the ISR for interrupt x. (1 ≤ x ≤ 255)

### Address of ISR 1

### Interrupt Vector Table

<table>
<thead>
<tr>
<th>Interrupt Number (8 bits)</th>
<th>Memory Address of ISR (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Interrupt Service Routine for interrupt 1</td>
</tr>
<tr>
<td>2</td>
<td>Interrupt Service Routine for interrupt 2</td>
</tr>
<tr>
<td>3</td>
<td>Interrupt Service Routine for interrupt 3</td>
</tr>
<tr>
<td>4</td>
<td>Interrupt Service Routine for interrupt 4</td>
</tr>
<tr>
<td>5</td>
<td>Interrupt Service Routine for interrupt 5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Interrupt Vector Table

The Nested Vectored Interrupt Controller (NVIC), prioritizes and handles all interrupts.

When we press the push button connected to the pin PA 3, the hardware generates an electrical signal, called interrupt request, EXTI3.

When NVIC receives the interrupt request, it forces the processor to jump to, and execute, a special piece of code, called an interrupt service routine, or an interrupt handler.

Then, the processor will execute the function named EXTI3_IRQHandler.
Interrupt Service Routine Vector Table

- Start address for the exception handler for each exception type is fixed and pre-defined
- Processor loads PC with this fixed, pre-defined address
- Exception Vector Table starts at memory address 0
- Program Counter \( pc = 0x00000004 \) initially

<table>
<thead>
<tr>
<th>Address</th>
<th>Priority</th>
<th>Type of priority</th>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000_0000</td>
<td></td>
<td></td>
<td></td>
<td>Stack Pointer</td>
</tr>
<tr>
<td>0x0000_0004</td>
<td>-3</td>
<td>fixed</td>
<td>Reset</td>
<td>Reset Vector</td>
</tr>
<tr>
<td>0x0000_0008</td>
<td>-2</td>
<td>fixed</td>
<td>NMI_Handler</td>
<td>Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.</td>
</tr>
<tr>
<td>0x0000_000C</td>
<td>-1</td>
<td>fixed</td>
<td>HardFault_Handler</td>
<td>All class of fault</td>
</tr>
<tr>
<td>0x0000_0010</td>
<td>0</td>
<td>settable</td>
<td>MemManage_Handler</td>
<td>Memory management</td>
</tr>
<tr>
<td>0x0000_0014</td>
<td>1</td>
<td>settable</td>
<td>BusFault_Handler</td>
<td>Pre-fetch fault, memory access fault</td>
</tr>
<tr>
<td>0x0000_0018</td>
<td>2</td>
<td>settable</td>
<td>UsageFault_Handler</td>
<td>Undefined instruction or illegal state</td>
</tr>
<tr>
<td>0x0000_001C-0x0000_002B</td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0000_002C</td>
<td>3</td>
<td>settable</td>
<td>SVC_Handler</td>
<td>System service call via SWI instruction</td>
</tr>
<tr>
<td>0x0000_0030</td>
<td>4</td>
<td>settable</td>
<td>DebugMon_Handler</td>
<td>Debug Monitor</td>
</tr>
<tr>
<td>0x0000_0034</td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0000_0038</td>
<td>5</td>
<td>settable</td>
<td>PendSV_Handler</td>
<td>Pendable request for system service</td>
</tr>
<tr>
<td>0x0000_003C</td>
<td>6</td>
<td>settable</td>
<td>SysTick_Handler</td>
<td>System tick timer</td>
</tr>
</tbody>
</table>

...
Interrupt Number

- Cortex-M supports up to 256 interrupts.
  - First 16 are system exceptions
    - CMSIS defines their interrupt numbers as negative
    - Defined by ARM core
  - The rest 240 are peripheral interrupts
    - Peripheral interrupt number starts with 0.
    - Defined by chip manufacturers.
CMSIS Interrupt Number

/*****  Cortex-M4 System Exceptions *******************************************/

NonMaskableIntIRQn = -14, /* 2 Cortex-M4 Non Maskable Interrupt */
HardFaultIRQn = -13, /* 3 Cortex-M4 Hard Fault Interrupt */
MemoryManagementIRQn = -12, /* 4 Cortex-M4 Memory Management Interrupt */
BusFaultIRQn = -11, /* 5 Cortex-M4 Bus Fault Interrupt */
UsageFaultIRQn = -10, /* 6 Cortex-M4 Usage Fault Interrupt */
SVCALL_IRQn = -5, /* 11 Cortex-M4 SV Call Interrupt */
DebugMonitorIRQn = -4, /* 12 Cortex-M4 Debug Monitor Interrupt */
PendSVIRQn = -2, /* 14 Cortex-M4 Pend SV Interrupt */
SysTickIRQn = -1, /* 15 Cortex-M4 System Tick Interrupt */

/*****  Peripheral Interrupt Numbers *******************************************/

WWDG_IRQn = 0, /* Window WatchDog Interrupt */
PVD_PVM_IRQn = 1, /* PVD/PVM1,2,3,4 through EXTI Line detection Interrupts */
TAMP_STAMP_IRQn = 2, /* Tamper and TimeStamp interrupts through the EXTI line */
RTC_WKUP_IRQn = 3, /* RTC Wakeup interrupt through the EXTI line */
FLASH_IRQn = 4, /* FLASH global Interrupt */
RCC_IRQn = 5, /* RCC global Interrupt */
EXTI0_IRQn = 6, /* EXTI Line0 Interrupt */
...

stm32l476xx.h
### Vector table for STM32F41

#### Reference Manual
Table 37

<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>-2</td>
<td>fixed</td>
<td>NMI</td>
<td>Non maskable interrupt, Clock Security System</td>
<td>0x0000 0008</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>fixed</td>
<td>HardFault</td>
<td>All class of fault</td>
<td>0x0000 000C</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>settable</td>
<td>MemManage</td>
<td>Memory management</td>
<td>0x0000 0010</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>settable</td>
<td>BusFault</td>
<td>Pre-fetch fault, memory access fault</td>
<td>0x0000 0014</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>settable</td>
<td>UsageFault</td>
<td>Undefined instruction or illegal state</td>
<td>0x0000 0018</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
<td>0x0000 001C - 0x0000 002B</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>settable</td>
<td>SVCall</td>
<td>System Service call via SWI instruction</td>
<td>0x0000 002C</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>settable</td>
<td>Debug Monitor</td>
<td>Debug Monitor</td>
<td>0x0000 0030</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
<td>0x0000 0034</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>settable</td>
<td>PendSV</td>
<td>Pendable request for system service</td>
<td>0x0000 0038</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>settable</td>
<td>Systick</td>
<td>System tick timer</td>
<td>0x0000 003C</td>
<td></td>
</tr>
<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>EXTI16 / PVD</td>
<td>EXTI Line 16 interrupt / PVD through EXTI line detection interrupt</td>
<td>0x0000 0044</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>settable</td>
<td>EXTI21 / TAMP_STAMP</td>
<td>EXTI Line 21 interrupt / 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>EXTI22 / RTC_WKUP</td>
<td>EXTI Line 22 interrupt / 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>
<tr>
<td>12</td>
<td>19</td>
<td>settable</td>
<td>DMA1_Stream1</td>
<td>DMA1 Stream1 global interrupt</td>
<td>0x0000 0070</td>
</tr>
<tr>
<td>13</td>
<td>20</td>
<td>settable</td>
<td>DMA1_Stream2</td>
<td>DMA1 Stream2 global interrupt</td>
<td>0x0000 0074</td>
</tr>
<tr>
<td>14</td>
<td>21</td>
<td>settable</td>
<td>DMA1_Stream3</td>
<td>DMA1 Stream3 global interrupt</td>
<td>0x0000 0078</td>
</tr>
</tbody>
</table>

**Interrupt Number**

`EXTI3_IRQn = 9`
**Vector table for STM32F41**

<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>-2</td>
<td>fixed</td>
<td>NMI</td>
<td>Non maskable interrupt, Clock Security System</td>
<td>0x0000 0008</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>fixed</td>
<td>HardFault</td>
<td>All class of fault</td>
<td>0x0000 000C</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>settable</td>
<td>MemManage</td>
<td>Memory management</td>
<td>0x0000 0010</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>settable</td>
<td>BusFault</td>
<td>Pre-fetch fault, memory access fault</td>
<td>0x0000 0014</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>settable</td>
<td>UsageFault</td>
<td>Undefined instruction or illegal state</td>
<td>0x0000 0018</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
<td></td>
<td>0x0000 001C - 0x0000 002B</td>
</tr>
<tr>
<td>3</td>
<td>settable</td>
<td>SVCall</td>
<td>System Service call via SWI instruction</td>
<td>0x0000 002C</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>settable</td>
<td>Debug Monitor</td>
<td>Debug Monitor</td>
<td>0x0000 0030</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
<td>0x0000 0034</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>settable</td>
<td>PendSV</td>
<td>Pendable request for system service</td>
<td>0x0000 0038</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>settable</td>
<td>Systick</td>
<td>System tick timer</td>
<td>0x0000 003C</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>settable</td>
<td>WWDG</td>
<td>Window Watchdog interrupt</td>
<td>0x0000 0040</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Address of pointer to EXTI3 ISR = 64 + 4 × 9 = 100 = 0x64

Interrupt Number EXTI3_IRQn = 9
Vector table for STM32F41

Example 2:
SysTick_IRQHandler = -1

Address of pointer to SysTick ISR = 64 + 4 × (-1) = 60 = 0x3C
Interrupt Service Routine (ISR)

We need to include it in the program

```c
void EXTI3_IRQHandler{
  Initialize SP
  Initialize PC
...
  state.
}
```

Memory Address of ISR

<table>
<thead>
<tr>
<th>Interrupt Number (8 bits)</th>
<th>Memory Address of ISR (32 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>EXTI0_IRQn = 6</td>
<td>EXTI0_IRQHandler</td>
</tr>
<tr>
<td>EXTI1_IRQn = 7</td>
<td>EXTI1_IRQHandler</td>
</tr>
<tr>
<td>EXTI2_IRQn = 8</td>
<td>EXTI2_IRQHandler</td>
</tr>
<tr>
<td>EXTI3_IRQn = 9</td>
<td>EXTI3_IRQHandler</td>
</tr>
<tr>
<td>EXTI4_IRQn = 10</td>
<td>EXTI4_IRQHandler</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>EXTI9_5_IRQn = 23</td>
<td>EXTI9_5_IRQHandler</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>EXTI15_10_IRQn = 40</td>
<td>EXTI15_10_IRQHandler</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Interrupt numbers on Vector table

- Bit 0 is 1, indicating Thumb state.
- Initialize PC
- Initialize SP
How Interrupts are enabled:

1. Start Main()
2. Process starts main.
3. Auto stacking: PUSH {R0-r3, r12, LR, PC, PSR}
4. Execute ISR
5. 1. Interrupt returns. Active bits will be cleared.
   2. Auto unstacking: POP {R0-r3, r12, LR, PC, PSR}
6. Continue to the execution of main program
Stacking & Unstacking

Interrupt Handler

User Program

Thread Mode

Handler Mode

Thread Mode

Interrupt Signal

Stacking

Unstacking

Interrupt Exit

User Program

Time
The External Interrupt controller (EXTI) supports two types of external interrupts:

- Configurable external interrupts
  - GPIO, RTC, comparators, power voltage detector (PVD), and peripheral voltage monitoring (PVM)
  - For these interrupts, the controller has a programmable edge detector, and software can select which active edge generate an interrupt request.

- Direct external interrupts
  - only rising edge can generate an interrupt request
  - Direct external interrupts are mostly used for communication peripherals, low-power timer, and LCD.
External Interrupt (EXTI) Controller

There are many registers involved in setting the interrupts

Find the address of the following registers:
EXTI_FTSR, EXTI_RTSR, EXTI_SWIER, EXTI_EMR, EXTI_IMR, EXTI_PR

What do these names stand for? What are they used for?
## Register Addresses

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Type</th>
<th>Reset Value</th>
<th>Offset</th>
<th>Register</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>EXTI_IMR</td>
<td>Reserved</td>
<td></td>
<td>0x00</td>
<td>MR[22:21]</td>
<td>Reserved</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>MR[18:0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x04</td>
<td>EXTI_EMR</td>
<td>Reserved</td>
<td></td>
<td>0x04</td>
<td>TR[22:21]</td>
<td>Reserved</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>TR[18:0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td>EXTI_RTSR</td>
<td>Reserved</td>
<td></td>
<td>0x08</td>
<td>SWIER[22:21]</td>
<td>Reserved</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>SWIER[18:0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td>EXTI_SWIER</td>
<td>Reserved</td>
<td></td>
<td>0x10</td>
<td>PR[22:21]</td>
<td>Reserved</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>PR[18:0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x14</td>
<td>EXTI_PR</td>
<td>Reserved</td>
<td></td>
<td>0x14</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Line 13**
External Interrupt (EXTI) Sources

SYSCFG external interrupt configuration register (SYSCFG_EXTICR)

PA \_x \ 000
PB \_x \ 001
PC \_x \ 010
PD \_x \ 011
PE \_x \ 100
PF \_x \ 101
PG \_x \ 110
PH \_x \ 111

source selection

x = 0, 1, 2, ..., 15

One mux for each of EXTI0, EXTI1, ..., EXTI15

STM32

Nested-Vectored Interrupt Controller

Cortex-M4

EXTI

NVIC
SYSCFG registers

Eight Registers – Identifying the source of the EXTERNAL Interrupt

See Reference Manual Section 7 (See Table 22)
SYSCFG registers

SYSCFG external interrupt configuration register 2 (SYSCFG_EXTICR2)

See Reference Manual Section 7
Steps to Configure PA.3 Acting on the Rising Edge

- Clock Source
- Enable PA.3 as Digital Input
- Enable Interrupt
- Rising Edge
Steps to Configure PA.3: Setup the Clock – Turn it on!

```c
// Pull PA.3 down internally; Trigger on rising edge
int main(void) {
    // Enable GPIO Clock
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    // GPIO Mode: Input(00), Output (01),
    //            AF(10),    Analog (11)
    GPIOA->MODER &= ~3U << 6;
    // GPIO Push-Pull: No pull
    //  Pull-up (01), Pull-down (10), Reserved (11)
    GPIOA->PUPDR &= ~3U << 6;
    GPIOA->PUPDR |= 2U << 6;    // Pull down
    NVIC_EnableIRQ(EXTI3_IRQn); // Enable Interrupt
    // Connect External Line to the GPIO
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI3;
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI3_PA;
    // Interrupt Mask Register
    // 0 = marked, 1 = not masked (enabled)
    EXTI->IMR1  |= EXTI_IMR1_IM3;
    // Rising trigger selection
    // 0 = trigger disabled, 1 = trigger enabled
    EXTI->RTSR1 |= EXTI_RTSR1_RT3;
    while(1);
}
```

```c
//stm32l476xx.h
#define RCC_AHB2ENR_GPIOAEN ((uint32_t)0x00000001U)
```

AHB2 peripheral clock enable register (AHB2ENR)

![AHB2 peripheral clock enable register](attachment:image)

Steps to Configure PA.3: Setup the Clock – Turn it on!
Steps to Configure PA.3: Setup PA.3 as digital input port

```c
int main(void) {
    // Enable GPIO Clock
    RCC_AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    // GPIO Mode: Input(00), Output (01),
    //            AF(10), Analog (11, default)
    GPIOA_MODER &= ~3U << 6;
    // Pull PA.3 down internally; Trigger on rising edge
    GPIOA_PUPDR &= ~3U << 6;    // Pull down
    GPIOA_PUPDR |= 2U << 6;    // Pull down
    NVIC_EnableIRQ(EXTI3_IRQn); // Enable Interrupt
    RCC_APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    SYSCFG_EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI3;
    SYSCFG_EXTICR[0] |=  SYSCFG_EXTICR1_EXTI3_PA;
    // Interrupt Mask Register
    // 0 = marked, 1 = not masked (enabled)
    EXTI_IMR1  |= EXTI_IMR1_IM3;
    // Rising trigger selection
    // 0 = trigger disabled, 1 = trigger enabled
    EXTI_RTSR1 |= EXTI_RTSR1_RT3;
    while(1);
}
```

Mode bits

- 00: Digital Input
- 01: Digital Output
- 10: Alternative Function
- 11: Analog

GPIO Pin 11
Steps to Configure PA.3: Enable the Pulldown on PA.3

```c
int main(void) {
    // Enable GPIO Clock
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    // GPIO Mode: Input(00), Output (01),
    //            AF(10),  Analog (11)
    GPIOA->MODER &= ~3U << 6;
    // GPIO Push-Pull: No pull-up, pull-down (00),
    // Pull-up (01), Pull-down (10), Reserved (11)
    GPIOA->PUPDR &= ~3U << 6;
    GPIOA->PUPDR |= 2U << 6;  // Pull down
    
    // Connect External Line to the GPIO
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI3;
    SYSCFG->EXTICR[0] |=  SYSCFG_EXTICR1_EXTI3_PA;
    
    // Interrupt Mask Register
    // 0 = marked, 1 = not masked (enabled)
    EXTI->IMR1  |= EXTI_IMR1_IM3;
    // Rising trigger selection
    // 0 = trigger disabled, 1 = trigger enabled
    EXTI->RTSR1 |= EXTI_RTSR1_RT3;
    
    while(1);
}
```
Steps to Configure PA.3: Enable Interrupts By NVIC – What to do when Interrupt occurs

```c
// Pull PA.3 down internally; Trigger on rising edge
int main(void) {
    // Enable GPIO Clock
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    // GPIO Mode: Input(00), Output (01),
    //            AF(10),    Analog (11)
    GPIOA->MODER &= ~3U << 6;
    // GPIO Push-Pull: No pull-up, pull-down (00),
    //                  Pull-up (01), Pull-down (10), Reserved (11)
    GPIOA->PUPDR &= ~3U << 6;
    GPIOA->PUPDR |= 2U << 6;    // Pull down
    NVIC_EnableIRQ(EXTI3_IRQn); // Enable Interrupt

    while(1);
}
```

Embedded microcontroller architecture and component labels such as Nested-Vectored Interrupt Controller (NVIC), Cortex-M4, EXTI3_IRQHandler, Interrupt Vector Table, and EXTI3_IRQn are included in the diagram for clarity.
Steps to Configure PA.3: Setup the correct source of the Interrupt (External Line to the MPU)

```c
int main(void) {
    // Enable GPIO Clock
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    // GPIO Mode: Input(00), Output (01),
    //            AF(10), Analog (11)
    GPIOA->MODER &= ~3U << 6;
    // GPIO Push-Pull: No pull-up, pull-down (00),
    //              Pull-up (01), Pull-down (10), Reserved (11)
    GPIOA->PUPDR &= ~3U << 6;
    GPIOA->PUPDR |= 2U << 6;    // Pull down
    NVIC_EnableIRQ(EXTI3_IRQn); // Enable Interrupt

    // Connect External Line to the GPI
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI3;
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI3_PA;
}
```
Steps to Configure PA.3: Enable Edge Trigger

// Pull PA.3 down internally; Trigger on rising edge
int main(void) {
    // Enable GPIO Clock
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    // GPIO Mode: Input(00), Output (01),
    //            AF(10),    Analog (11)
    GPIOA->MODER &= ~3U << 6;
    // GPIO Push-Pull: No pull-up, pull-down (00),
    //    Pull-up (01), Pull-down (10), Reserved (11)
    GPIOA->PUPDR &= ~3U << 6;
    GPIOA->PUPDR |= 2U << 6; // Pull down
    NVIC_EnableIRQ(EXTI3_IRQn); // Enable Interrupt
    // Connect External Line to the GPI
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
    SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI3;
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI3_PA;
    // Rising trigger selection
    // 0 = trigger disabled, 1 = trigger enabled
    EXTI->RTSR1 |= EXTI_RTSR1_RT3;
    while(1);
}

Rising edge triggers EXTI interrupt.
// Pull PA.3 down internally; Trigger on rising edge
int main(void) {
    // Enable GPIO Clock
    RCC->AHB2ENR | = RCC_AHB2ENR_GPIOAEN;
    // GPIO Mode: Input(00), Output (01),
    //            AF(10),    Analog (11)
    GPIOA->MODER & = ~3U << 6;
    // GPIO Push-Pull: No pull-up, pull-down (00),
    // Pull-up (01), Pull-down (10), Reserved (11)
    GPIOA->PUPDR & = ~3U << 6;
    GPIOA->PUPDR | = 2U << 6;    // Pull down
    NVIC_EnableIRQ(EXTI3_IRQn); // Enable Interrupt

    // Connect External Line to the GPI
    RCC->APB2ENR | = RCC_APB2ENR_SYSCFGEN;
    SYSCFG->EXTICR[0] & = ~SYSCFG_EXTICR1_EXTI3;
    SYSCFG->EXTICR[0] | = SYSCFG_EXTICR1_EXTI3_PA;

    // Rising trigger selection
    // 0 = trigger disabled, 1 = trigger enabled
    EXTI->RTSR1 | = EXTI_RTSR1_RT3;

    // Interrupt Mask Register
    // 0 = marked, 1 = not masked (enabled)
    EXTI->IMR1 | = EXTI_IMR1_IM3;
}

Steps to Configure PA.3: Enable the Interrupt Register
Steps to Configure PA.3: Wait until an Interrupt Occurs!

```c
// Pull PA.3 down internally; Trigger on rising edge
int main(void) {
    // Enable GPIO Clock
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
    // GPIO Mode: Input(00), Output (01),
    //            AF(10), Analog (11)
    GPIOA->MODER &= ~3U << 6;
    // GPIO Push-Pull: No pull-up, pull-down (00),
    // Pull-up (01), Pull-down (10), Reserved (11)
    GPIOA->PUPDR &= ~3U << 6;
    GPIOA->PUPDR |= 2U << 6;  // Pull down

    NVIC_EnableIRQ(EXTI3_IRQn); // Enable Interrupt

    // Connect External Line to the GPI
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGGEN;
    SYSCFG->EXTICR[0] &= ~SYSCFG_EXTICR1_EXTI3;
    SYSCFG->EXTICR[0] |=  SYSCFG_EXTICR1_EXTI3_PA;

    // Interrupt Mask Register
    // 0 = marked, 1 = not masked (enabled)
    EXTI->IMR1 |= EXTI_IMR1_IM3;

    // Rising trigger selection
    // 0 = trigger disabled, 1 = trigger enabled
    EXTI->RTSR1 |= EXTI_RTSR1_RT3;

    while(1);
}
```

Processor runs a dead loop forever!
// Clock Registers
#define RCC_AHB1ENR 0x40023830 // enables/disables clocks for peripherals
#define RCC_APB2ENR 0x40023844 // enable peripheral clock register for SYSCFG

// GPIO Registers
#define GPIOA_MODER 0x40020000 // select the I/O direction
#define GPIOA_ODR 0x40020014 // stores the data to be output, it is read/write accessible
#define GPIOC_IDR 0x40020810 // stores the data on the input register
#define GPIOC_MODER 0x40020800 // sets a pin on port C as input or output

// Interrupt Registers:
#define SYSCFG_EXTICR4 0x40013814 // Set PC13 as interrupt source
#define EXTI_IMR 0x40013C00 // Interrupt Mask Register
#define EXTI_FTSR 0x40013C0C // Falling Edge Trigger
#define NVIC 0xE000E104 // Nested Vector Interrupt Controller
#define EXTI_PR 0x40013C14 // Pending IRQ
Another Example (Set PC.13 - Falling Edge)

// Clock Registers
#define RCC_AHB1ENR 0x40023830 // enables/disables clocks for peripherals
#define RCC_APB2ENR 0x40023844 // enable peripheral clock register for SYSCFG

// GPIO Registers
#define GPIOA_MODER 0x40020000 // select the I/O direction
#define GPIOA_ODR 0x40020014 // stores the data to be output, it is read/write accessible
#define GPIOC_IDR 0x40020810 // stores the data on the input register
#define GPIOC_MODER 0x40020800 // sets a pin on port C as input or output

// Interrupt Registers:
#define SYSCFG_EXTICR4 0x40013814 // Set PC13 as interrupt source
#define EXTI_IMR 0x40013C00 // Interrupt Mask Register
#define EXTI_FTSR 0x40013C0C // Falling Edge Trigger
#define NVIC 0xE000E104 // Nested Vector Interrupt Controller
#define EXTI_PR 0x40013C14 // Pending IRQ

// Interrupt registers
volatile uint32_t *exticr4 = (volatile uint32_t *)SYSCFG_EXTICR4;
volatile uint32_t *exti_imr = (volatile uint32_t *)EXTI_IMR;
volatile uint32_t *exti_ftsr = (volatile uint32_t *)EXTI_FTSR;
volatile uint32_t *exti_pr = (volatile uint32_t *)EXTI_PR;
volatile uint32_t *nvic_enable = (volatile uint32_t *)NVIC;

// Turn on Clocks
*clk_reg |= 0x5; // Turn on GPIOA and GPIOC
*sys_cfg_clk_reg |= 0x4000 ; // Clock SYSCFG used to set PORTC to Line13

// Configure GPIO
*dir_reg |= 0x400; // Set GPIOA PA05 as an OUTPUT
uint32_t timer = LED_DELAY_ON; // Load initial timer

// Configure Interrupts
*exticr4 |= 0x20; // Sets Line13 to be from Port C[13]
*exti_ftsr |= 0x2000; // Generate interrupt on FALLING edge for Line13
*exti_imr |= 0x2000; // Unmask interrupt for Line13
*nvic_enable |= 0x100; // Enable Interrupt 40 - EXTI15_10: From Vector Table 199
Interrupt Handler

- What should we do when interrupt occurs? → Erase it & wait for another interrupt!
- Where is the code to do so? → This is in the interrupt handler # 40
- But how do we “erase” the interrupt? → Use EXTI_PR register

```c
// ISR for line 15-10 interrupt
void EXTI15_10_IRQHandler(void){
    if((*exti_pr & 0x2000) != 0) {
        toggle ^= 1;
        *exti_pr |= 0x2000;
    }
    // Test if Line 13 interrupt has occurred
    // Clear Line 13 by writing to it (cleared by HW)
}
```

Interrupt PC.13 → # 40
→ We use Handler EXTI15-10