



---

## Interrupt overview



---

## Contents

---

1. Interrupt overview .....	34
2. EXTI internal peripheral .....	10
3. GIC internal peripheral .....	18
4. GPIO internal peripheral .....	26
5. PWR internal peripheral .....	42
6. Power overview .....	50
7. Pseudo filesystem .....	58



A quality version of this page, approved on 17 March 2021, was based off this revision.

This article explains stm32mp157 interrupt topology and its management on Linux<sup>®</sup> environment.

## Contents

1 Framework purpose .....	36
2 STM32 interrupt topology .....	37
2.1 Overview .....	37
2.2 Component description .....	37
3 API description .....	39
3.1 User space API .....	39
3.2 Kernel space API .....	39
4 Configuration .....	40
4.1 Kernel configuration .....	40
4.2 Device tree configuration .....	40
4.2.1 GIC irqchip .....	40
4.2.2 EXTI irqchip .....	40
4.2.3 EXTI_PWR irqchip .....	40
4.2.4 pinctrl irqchip .....	41
4.2.5 pwr irqchip .....	41
5 References .....	42



---

## 1 Framework purpose

---

The Linux<sup>®</sup> kernel software layer that handles the interrupts is splitted into two parts:"

- A generic part:
  - providing a common API to request and configure an interrupt line.
  - creating a virtual mapping for all interrupts in order to have only one ID per interrupt.
  - providing callback for irqchip registering.
- An irqchip driver part:
  - handling hardware accesses and managing specific features.

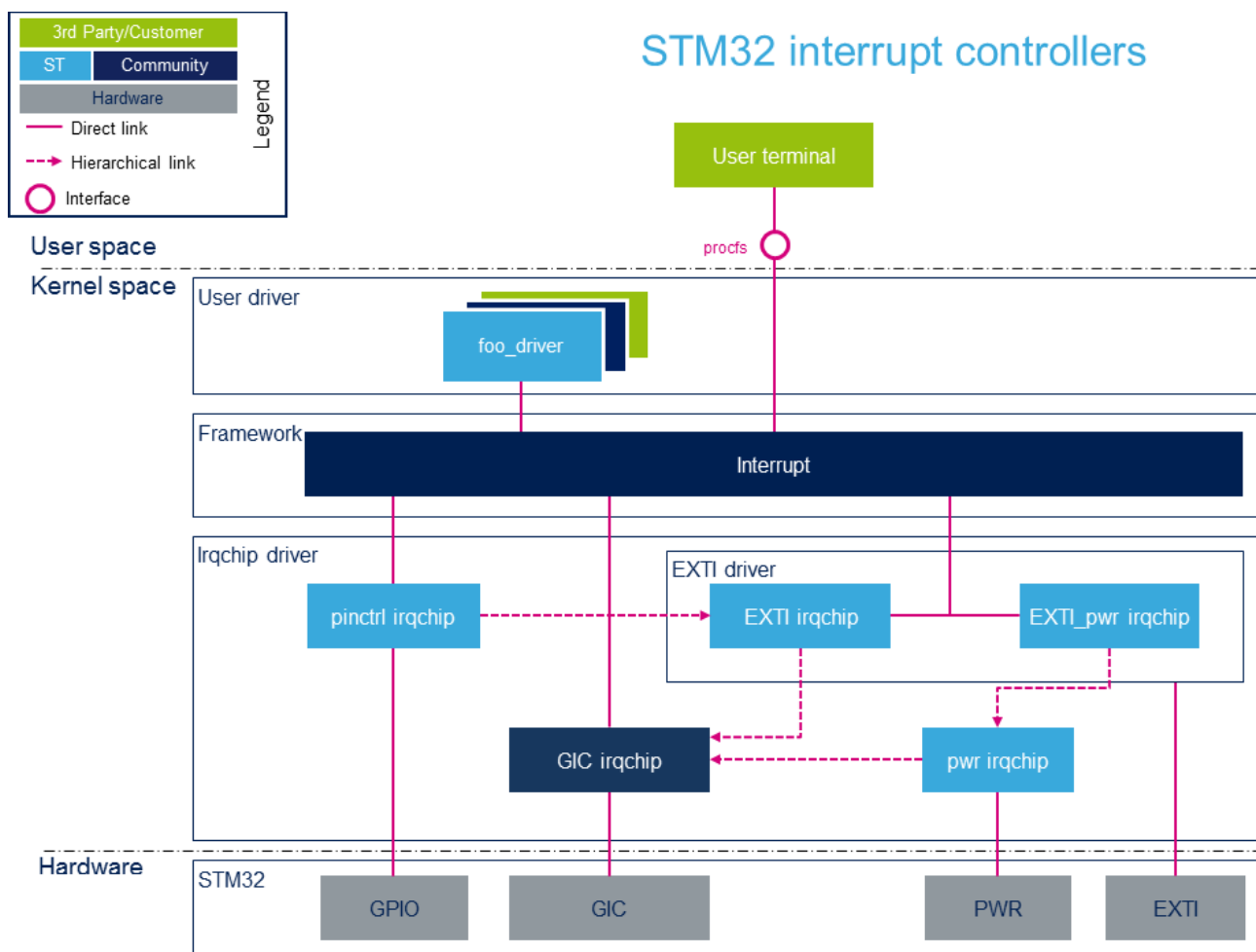
For more information refer to Linux<sup>®</sup> kernel documentation in [core-api/genericirq.html<sup>\[1\]</sup>](#).

## 2 STM32 interrupt topology

As explain in [Framework purpose](#), the irqchip driver makes the interface with the hardware to configure and manage an interrupt. On STM32MP1 devices, a hardware interrupt can be generated by GIC, EXTI, PWR or GPIO. Several irqchip drivers are consequently required, one per hardware block.

The next section provides topology information for each kind of interrupt source.

### 2.1 Overview



### 2.2 Component description

- **procfcs:** provides interrupt information to the user space.
- **foo\_driver:** device driver requesting an interrupt line.
- **interrupt framework:** generic part described in the [Framework purpose](#) section.



- 
- **Irqchips:**
    - **GIC irqchip:** GIC irqchip driver part. This irqchip driver is used when a GIC interrupt is directly requested by a device. This is the case for all the peripheral interrupts that are not wakeup sources. This irqchip is in charge of controlling the GIC internal peripheral (hardware). An example of GIC irqchip usage is available [here](#).
    - **EXTI irqchip:** EXTI irqchip driver part. This irqchip driver is used when an EXTI interrupt (EXTernal Interrupt) is requested by a device. This kind of interrupts is used to wake up the system from **low-power mode**. This irqchip directly controls the EXTI internal peripheral but it is also linked to the **gic irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
    - **EXTI\_pwr irqchip:** EXTI\_pwr irqchip driver part. This irqchip driver is used when an EXTI interrupt mapped to a wakeup pin is requested by a device. This kind of interrupts is used to wake up the system from the deepest low-power mode (more details about low-power modes are available [here](#)). This irqchip directly controls the EXTI internal peripheral but it is also linked to the **pwr irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. The **pwr irqchip** in turn controls the PWR internal peripheral and it is also linked to the **gic irqchip** through the same hierarchical irq domain mechanism<sup>[2]</sup>. These hierarchical links are transparent for the requester.
    - **Pinctrl irqchip:** Pinctrl irqchip driver part. This irqchip driver is used when a device wants to configure/request a GPIO as an interrupt. It directly controls the GPIO internal peripheral but it is also linked to the **EXTI irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
  - **STM32 hardware peripherals:** GIC, EXTI, PWR, GPIO



## 3 API description

The kernel space API is the interface for declaring and managing interrupts. The user space interface is used to monitor interrupt information or set interrupt affinity.

### 3.1 User space API

procs performs the following tasks:

- It provides information about interrupts such as the virtual number, the hardware ID and the irqchip used (see chapter 1.2 *Kernel data of /proc kernel documentation*<sup>[3]</sup>).

```

root@stm32mp1:~# cat /proc/interrupts
          CPU0           CPU1
17:         0             0      GIC-0  37 Level      rcc irq
20:    7509664       7509640      GIC-0  27 Level      arch_timer
22:         0             0      GIC-0 232 Level      arm-pmu
23:         0             0      GIC-0 233 Level      arm-pmu
24:         0             0      GIC-0  68 Level      4000b000.audio-controller
26:         0             0  stm32-exti-h 27 Edge      4000e000.serial:wakeup
27:         8915          0      GIC-0  84 Level      40010000.serial
28:         0             0  stm32-exti-h 30 Edge      40010000.serial:wakeup
29:         654          0      GIC-0  63 Level      40012000.i2c
30:         0             0      GIC-0  64 Level      40012000.i2c
31:         0             0  stm32-exti-h 21 Edge      40012000.i2c:wakeup
33:         0             0      GIC-0 123 Level      4400b004.audio-controller, 4400b024.
audio-controller
...

```

- It configures interrupt affinity<sup>[4]</sup>, that is assigns an interrupt to a dedicated CPU.

### 3.2 Kernel space API

The main kernel API drivers for users are the following:

- **devm\_request\_irq**: requests an interrupt.
- **devm\_free\_irq**: frees an interrupt.
- **enable\_irq**: enables a requested interrupt.
- **disable\_irq**: disables a requested interrupt.
- **enable\_irq\_wake**: enables a requested interrupt that could wake up the system.
- **disable\_irq\_wake**: disables a requested interrupt that could wake up the system.

... The available routines can be found in Linux<sup>®</sup> kernel header file: *include/linux/interrupt.h*<sup>[5]</sup>.



## 4 Configuration

### 4.1 Kernel configuration

The interrupt framework and irqchip drivers are enabled by default.

### 4.2 Device tree configuration

The generic way to declare an interrupt in the device tree is declared in Linux<sup>®</sup> kernel documentation in: *Documentation/devicetree/bindings/interrupt-controller/interrupts.txt* <sup>[6]</sup>.

However each irqchip driver has his own bindings description. The below chapters provide the link to the bindings documentation for each interrupt controller as well as a simple example of interrupt declaration.

#### 4.2.1 GIC irqchip

- *Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt* <sup>[7]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts = <&intc GIC_SPI 72 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.2 EXTI irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti 26 IRQ_TYPE_EDGE_RISING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.3 EXTI\_PWR irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti_pwr 55 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```





---

#### 4.2.4 pinctrl irqchip

- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&gpioa 14 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.5 pwr irqchip

Pwr irqchip has not to be used directly. User has to use *exti\_pwr* to invoke pwr irqchip thanks to the hierarchical implementation.



## 5 References

- Generic IRQ documentation
- 2.02.12.22.3 [https://www.kernel.org/doc/Documentation/IRQ-domain.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-domain.txt(master)), IRQ domain documentation
- [https://www.kernel.org/doc/Documentation/filesystems/proc.txt\(master\)](https://www.kernel.org/doc/Documentation/filesystems/proc.txt(master)), User space /proc documentation
- [https://www.kernel.org/doc/Documentation/IRQ-affinity.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-affinity.txt(master)), IRQ affinity documentation
- [include/linux/interrupt.h ], Kernel interrupt API
- Generic interrupts bindings documentation, Generic interrupts bindings documentation
- [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt(master)), GIC controller binding documentation
- 8.08.1 [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt(master)), Exti interrupts bindings documentation

Linux<sup>®</sup> is a registered trademark of Linus Torvalds.

Application programming interface

Generic Interrupt Controller

External Interrupt

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Process File System (See <https://en.wikipedia.org/wiki/Procsfs> for more details)

foo\_driver could be any driver that needs to control a GPIO

Central processing unit

Serial Peripheral Interface

Stable: 26.03.2021 - 13:10 / Revision: 18.03.2021 - 14:45

This article explains stm32mp157 interrupt topology and its management on Linux<sup>®</sup> environment.

### Contents

1 Framework purpose .....	12
2 STM32 interrupt topology .....	13
2.1 Overview .....	13
2.2 Component description .....	13
3 API description .....	15
3.1 User space API .....	15
3.2 Kernel space API .....	15
4 Configuration .....	16
4.1 Kernel configuration .....	16
4.2 Device tree configuration .....	16
4.2.1 GIC irqchip .....	16
4.2.2 EXTI irqchip .....	16
4.2.3 EXTI_PWR irqchip .....	16



---

4.2.4 pinctrl irqchip .....	17
4.2.5 pwr irqchip .....	17
5 References .....	18



---

## 1 Framework purpose

---

The Linux<sup>®</sup> kernel software layer that handles the interrupts is splitted into two parts:"

- A generic part:
  - providing a common API to request and configure an interrupt line.
  - creating a virtual mapping for all interrupts in order to have only one ID per interrupt.
  - providing callback for irqchip registering.
- An irqchip driver part:
  - handling hardware accesses and managing specific features.

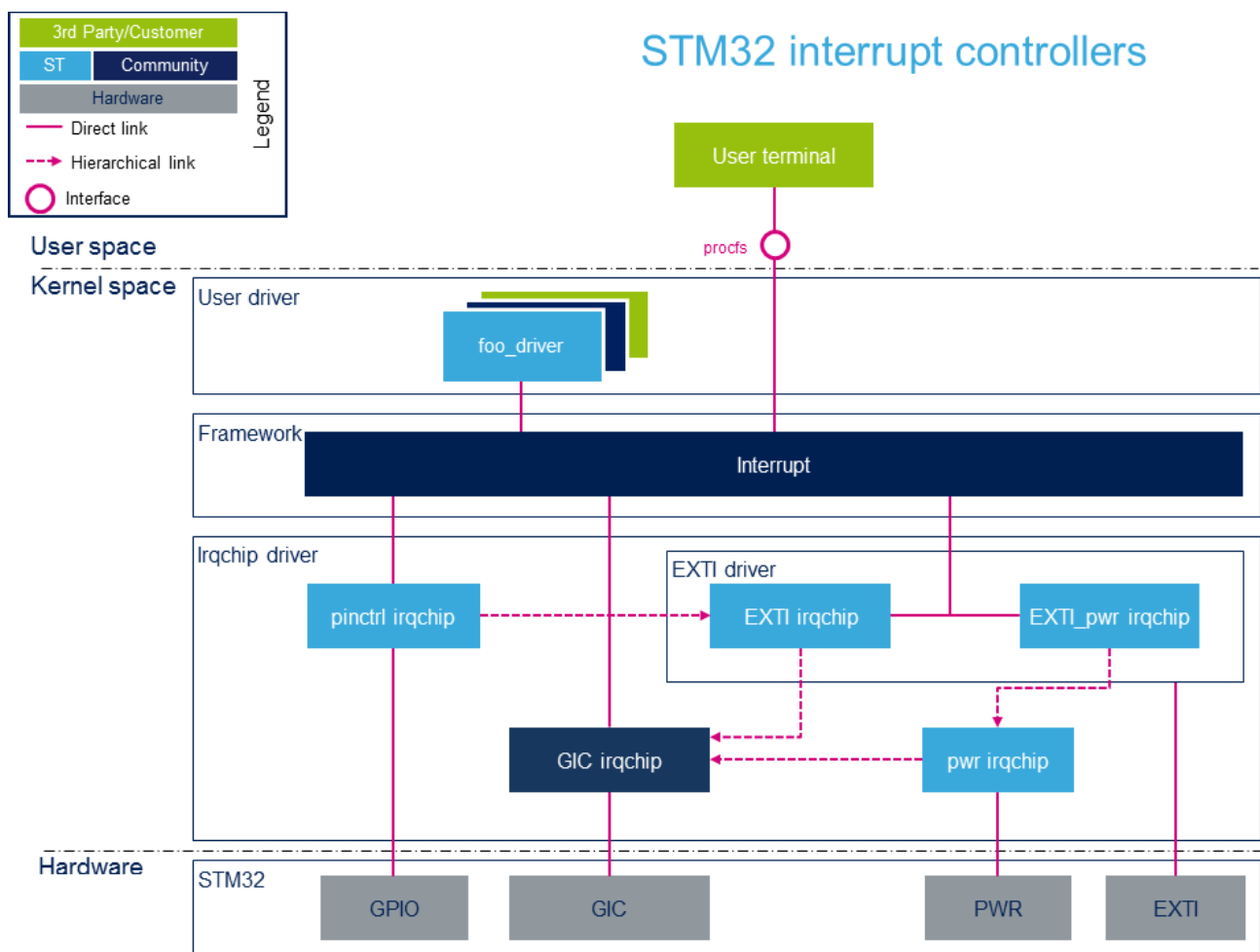
For more information refer to Linux<sup>®</sup> kernel documentation in [core-api/genericirq.html<sup>\[1\]</sup>](#).

## 2 STM32 interrupt topology

As explain in [Framework purpose](#), the irqchip driver makes the interface with the hardware to configure and manage an interrupt. On STM32MP1 devices, a hardware interrupt can be generated by GIC, EXTI, PWR or GPIO. Several irqchip drivers are consequently required, one per hardware block.

The next section provides topology information for each kind of interrupt source.

### 2.1 Overview



### 2.2 Component description

- **procf**: provides interrupt information to the user space.
- **foo\_driver**: device driver requesting an interrupt line.
- **interrupt framework**: generic part described in the [Framework purpose](#) section.



- 
- **Irqchips:**
    - **GIC irqchip:** GIC irqchip driver part. This irqchip driver is used when a GIC interrupt is directly requested by a device. This is the case for all the peripheral interrupts that are not wakeup sources. This irqchip is in charge of controlling the GIC internal peripheral (hardware). An example of GIC irqchip usage is available [here](#).
    - **EXTI irqchip:** EXTI irqchip driver part. This irqchip driver is used when an EXTI interrupt (EXTERNAL Interrupt) is requested by a device. This kind of interrupts is used to wake up the system from **low-power mode**. This irqchip directly controls the EXTI internal peripheral but it is also linked to the **gic irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
    - **EXTI\_pwr irqchip:** EXTI\_pwr irqchip driver part. This irqchip driver is used when an EXTI interrupt mapped to a wakeup pin is requested by a device. This kind of interrupts is used to wake up the system from the deepest low-power mode (more details about low-power modes are available [here](#)). This irqchip directly controls the EXTI internal peripheral but it is also linked to the **pwr irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. The **pwr irqchip** in turn controls the PWR internal peripheral and it is also linked to the **gic irqchip** through the same hierarchical irq domain mechanism<sup>[2]</sup>. These hierarchical links are transparent for the requester.
    - **Pinctrl irqchip:** Pinctrl irqchip driver part. This irqchip driver is used when a device wants to configure/request a GPIO as an interrupt. It directly controls the GPIO internal peripheral but it is also linked to the **EXTI irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
  - **STM32 hardware peripherals:** GIC, EXTI, PWR, GPIO



## 3 API description

The kernel space API is the interface for declaring and managing interrupts. The user space interface is used to monitor interrupt information or set interrupt affinity.

### 3.1 User space API

procs performs the following tasks:

- It provides information about interrupts such as the virtual number, the hardware ID and the irqchip used (see chapter 1.2 *Kernel data of /proc kernel documentation*<sup>[3]</sup>).

```

root@stm32mp1:~# cat /proc/interrupts
          CPU0           CPU1
17:         0             0      GIC-0  37 Level      rcc irq
20:    7509664    7509640      GIC-0  27 Level      arch_timer
22:         0             0      GIC-0 232 Level      arm-pmu
23:         0             0      GIC-0 233 Level      arm-pmu
24:         0             0      GIC-0  68 Level      4000b000.audio-controller
26:         0             0  stm32-exti-h 27 Edge      4000e000.serial:wakeup
27:     8915             0      GIC-0  84 Level      40010000.serial
28:         0             0  stm32-exti-h 30 Edge      40010000.serial:wakeup
29:         654             0      GIC-0  63 Level      40012000.i2c
30:         0             0      GIC-0  64 Level      40012000.i2c
31:         0             0  stm32-exti-h 21 Edge      40012000.i2c:wakeup
33:         0             0      GIC-0 123 Level      4400b004.audio-controller, 4400b024.
audio-controller
...

```

- It configures interrupt affinity<sup>[4]</sup>, that is assigns an interrupt to a dedicated CPU.

### 3.2 Kernel space API

The main kernel API drivers for users are the following:

- **devm\_request\_irq**: requests an interrupt.
- **devm\_free\_irq**: frees an interrupt.
- **enable\_irq**: enables a requested interrupt.
- **disable\_irq**: disables a requested interrupt.
- **enable\_irq\_wake**: enables a requested interrupt that could wake up the system.
- **disable\_irq\_wake**: disables a requested interrupt that could wake up the system.

... The available routines can be found in Linux<sup>®</sup> kernel header file: *include/linux/interrupt.h*<sup>[5]</sup>.



## 4 Configuration

### 4.1 Kernel configuration

The interrupt framework and irqchip drivers are enabled by default.

### 4.2 Device tree configuration

The generic way to declare an interrupt in the device tree is declared in Linux<sup>®</sup> kernel documentation in: *Documentation/devicetree/bindings/interrupt-controller/interrupts.txt* <sup>[6]</sup>.

However each irqchip driver has his own bindings description. The below chapters provide the link to the bindings documentation for each interrupt controller as well as a simple example of interrupt declaration.

#### 4.2.1 GIC irqchip

- *Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt* <sup>[7]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts = <&intc GIC_SPI 72 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.2 EXTI irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti 26 IRQ_TYPE_EDGE_RISING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.3 EXTI\_PWR irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti_pwr 55 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```





---

#### 4.2.4 pinctrl irqchip

- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&gpioa 14 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.5 pwr irqchip

Pwr irqchip has not to be used directly. User has to use *exti\_pwr* to invoke pwr irqchip thanks to the hierarchical implementation.



## 5 References

- Generic IRQ documentation
- 2.02.12.22.3 [https://www.kernel.org/doc/Documentation/IRQ-domain.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-domain.txt(master)), IRQ domain documentation
- [https://www.kernel.org/doc/Documentation/filesystems/proc.txt\(master\)](https://www.kernel.org/doc/Documentation/filesystems/proc.txt(master)), User space /proc documentation
- [https://www.kernel.org/doc/Documentation/IRQ-affinity.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-affinity.txt(master)), IRQ affinity documentation
- [include/linux/interrupt.h ], Kernel interrupt API
- Generic interrupts bindings documentation, Generic interrupts bindings documentation
- [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt(master)), GIC controller binding documentation
- 8.08.1 [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt(master)), Exti interrupts bindings documentation

Linux<sup>®</sup> is a registered trademark of Linus Torvalds.

Application programming interface

Generic Interrupt Controller

External Interrupt

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Process File System (See <https://en.wikipedia.org/wiki/Procsfs> for more details)

foo\_driver could be any driver that needs to control a GPIO

Central processing unit

Serial Peripheral Interface

Stable: 26.03.2021 - 13:17 / Revision: 18.03.2021 - 14:47

This article explains stm32mp157 interrupt topology and its management on Linux<sup>®</sup> environment.

### Contents

1 Framework purpose .....	20
2 STM32 interrupt topology .....	21
2.1 Overview .....	21
2.2 Component description .....	21
3 API description .....	23
3.1 User space API .....	23
3.2 Kernel space API .....	23
4 Configuration .....	24
4.1 Kernel configuration .....	24
4.2 Device tree configuration .....	24
4.2.1 GIC irqchip .....	24
4.2.2 EXTI irqchip .....	24
4.2.3 EXTI_PWR irqchip .....	24



---

4.2.4 pinctrl irqchip .....	25
4.2.5 pwr irqchip .....	25
5 References .....	26



---

## 1 Framework purpose

---

The Linux<sup>®</sup> kernel software layer that handles the interrupts is splitted into two parts:"

- A generic part:
  - providing a common API to request and configure an interrupt line.
  - creating a virtual mapping for all interrupts in order to have only one ID per interrupt.
  - providing callback for irqchip registering.
- An irqchip driver part:
  - handling hardware accesses and managing specific features.

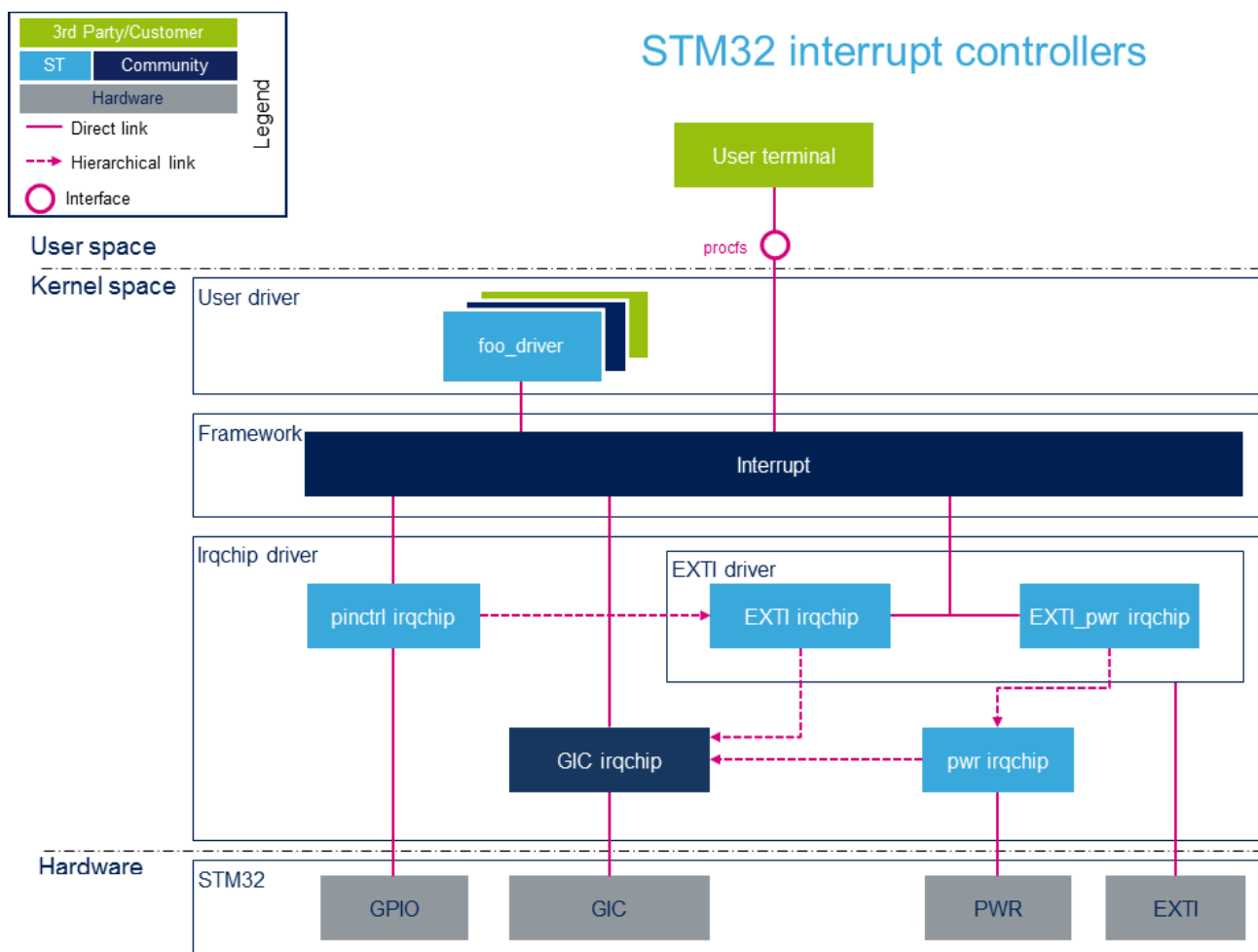
For more information refer to Linux<sup>®</sup> kernel documentation in [core-api/genericirq.htm<sup>\[1\]</sup>](#).

## 2 STM32 interrupt topology

As explain in [Framework purpose](#), the irqchip driver makes the interface with the hardware to configure and manage an interrupt. On STM32MP1 devices, a hardware interrupt can be generated by GIC, EXTI, PWR or GPIO. Several irqchip drivers are consequently required, one per hardware block.

The next section provides topology information for each kind of interrupt source.

### 2.1 Overview



### 2.2 Component description

- **procfcs:** provides interrupt information to the user space.
- **foo\_driver:** device driver requesting an interrupt line.
- **interrupt framework:** generic part described in the [Framework purpose](#) section.



- 
- **Irqchips:**
    - **GIC irqchip:** GIC irqchip driver part. This irqchip driver is used when a GIC interrupt is directly requested by a device. This is the case for all the peripheral interrupts that are not wakeup sources. This irqchip is in charge of controlling the GIC internal peripheral (hardware). An example of GIC irqchip usage is available [here](#).
    - **EXTI irqchip:** EXTI irqchip driver part. This irqchip driver is used when an EXTI interrupt (EXTernal Interrupt) is requested by a device. This kind of interrupts is used to wake up the system from **low-power mode**. This irqchip directly controls the EXTI internal peripheral but it is also linked to the **gic irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
    - **EXTI\_pwr irqchip:** EXTI\_pwr irqchip driver part. This irqchip driver is used when an EXTI interrupt mapped to a wakeup pin is requested by a device. This kind of interrupts is used to wake up the system from the deepest low-power mode (more details about low-power modes are available [here](#)). This irqchip directly controls the EXTI internal peripheral but it is also linked to the **pwr irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. The **pwr irqchip** in turn controls the PWR internal peripheral and it is also linked to the **gic irqchip** through the same hierarchical irq domain mechanism<sup>[2]</sup>. These hierarchical links are transparent for the requester.
    - **Pinctrl irqchip:** Pinctrl irqchip driver part. This irqchip driver is used when a device wants to configure/request a GPIO as an interrupt. It directly controls the GPIO internal peripheral but it is also linked to the **EXTI irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
  - **STM32 hardware peripherals:** GIC, EXTI, PWR, GPIO



## 3 API description

The kernel space API is the interface for declaring and managing interrupts. The user space interface is used to monitor interrupt information or set interrupt affinity.

### 3.1 User space API

`procds` performs the following tasks:

- It provides information about interrupts such as the virtual number, the hardware ID and the irqchip used (see chapter 1.2 *Kernel data of /proc kernel documentation*<sup>[3]</sup>).

```

root@stm32mp1:~# cat /proc/interrupts
          CPU0           CPU1
17:         0             0      GIC-0  37 Level      rcc irq
20:    7509664    7509640      GIC-0  27 Level      arch_timer
22:         0             0      GIC-0 232 Level      arm-pmu
23:         0             0      GIC-0 233 Level      arm-pmu
24:         0             0      GIC-0  68 Level      4000b000.audio-controller
26:         0             0  stm32-exti-h 27 Edge      4000e000.serial:wakeup
27:     8915             0      GIC-0  84 Level      40010000.serial
28:         0             0  stm32-exti-h 30 Edge      40010000.serial:wakeup
29:         654             0      GIC-0  63 Level      40012000.i2c
30:         0             0      GIC-0  64 Level      40012000.i2c
31:         0             0  stm32-exti-h 21 Edge      40012000.i2c:wakeup
33:         0             0      GIC-0 123 Level      4400b004.audio-controller, 4400b024.
audio-controller
...

```

- It configures interrupt affinity<sup>[4]</sup>, that is assigns an interrupt to a dedicated CPU.

### 3.2 Kernel space API

The main kernel API drivers for users are the following:

- **`devm_request_irq`**: requests an interrupt.
- **`devm_free_irq`**: frees an interrupt.
- **`enable_irq`**: enables a requested interrupt.
- **`disable_irq`**: disables a requested interrupt.
- **`enable_irq_wake`**: enables a requested interrupt that could wake up the system.
- **`disable_irq_wake`**: disables a requested interrupt that could wake up the system.

... The available routines can be found in Linux<sup>®</sup> kernel header file: *include/linux/interrupt.h*<sup>[5]</sup>.



## 4 Configuration

### 4.1 Kernel configuration

The interrupt framework and irqchip drivers are enabled by default.

### 4.2 Device tree configuration

The generic way to declare an interrupt in the device tree is declared in Linux<sup>®</sup> kernel documentation in: *Documentation/devicetree/bindings/interrupt-controller/interrupts.txt* <sup>[6]</sup>.

However each irqchip driver has his own bindings description. The below chapters provide the link to the bindings documentation for each interrupt controller as well as a simple example of interrupt declaration.

#### 4.2.1 GIC irqchip

- *Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt* <sup>[7]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts = <&intc GIC_SPI 72 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.2 EXTI irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti 26 IRQ_TYPE_EDGE_RISING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.3 EXTI\_PWR irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti_pwr 55 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```





---

#### 4.2.4 pinctrl irqchip

- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&gpioa 14 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.5 pwr irqchip

Pwr irqchip has not to be used directly. User has to use *exti\_pwr* to invoke pwr irqchip thanks to the hierarchical implementation.



## 5 References

- Generic IRQ documentation
- 2.02.12.22.3 [https://www.kernel.org/doc/Documentation/IRQ-domain.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-domain.txt(master)), IRQ domain documentation
- [https://www.kernel.org/doc/Documentation/filesystems/proc.txt\(master\)](https://www.kernel.org/doc/Documentation/filesystems/proc.txt(master)), User space /proc documentation
- [https://www.kernel.org/doc/Documentation/IRQ-affinity.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-affinity.txt(master)), IRQ affinity documentation
- [include/linux/interrupt.h ], Kernel interrupt API
- Generic interrupts bindings documentation, Generic interrupts bindings documentation
- [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt(master)), GIC controller binding documentation
- 8.08.1 [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt(master)), Exti interrupts bindings documentation

Linux<sup>®</sup> is a registered trademark of Linus Torvalds.

Application programming interface

Generic Interrupt Controller

External Interrupt

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Process File System (See <https://en.wikipedia.org/wiki/Procsfs> for more details)

foo\_driver could be any driver that needs to control a GPIO

Central processing unit

Serial Peripheral Interface

Stable: 12.08.2021 - 15:30 / Revision: 12.08.2021 - 15:30

This article explains stm32mp157 interrupt topology and its management on Linux<sup>®</sup> environment.

### Contents

1 Framework purpose .....	28
2 STM32 interrupt topology .....	29
2.1 Overview .....	29
2.2 Component description .....	29
3 API description .....	31
3.1 User space API .....	31
3.2 Kernel space API .....	31
4 Configuration .....	32
4.1 Kernel configuration .....	32
4.2 Device tree configuration .....	32
4.2.1 GIC irqchip .....	32
4.2.2 EXTI irqchip .....	32
4.2.3 EXTI_PWR irqchip .....	32



---

4.2.4 pinctrl irqchip .....	33
4.2.5 pwr irqchip .....	33
5 References .....	34



---

## 1 Framework purpose

---

The Linux® kernel software layer that handles the interrupts is splitted into two parts:"

- A generic part:
  - providing a common API to request and configure an interrupt line.
  - creating a virtual mapping for all interrupts in order to have only one ID per interrupt.
  - providing callback for irqchip registering.
- An irqchip driver part:
  - handling hardware accesses and managing specific features.

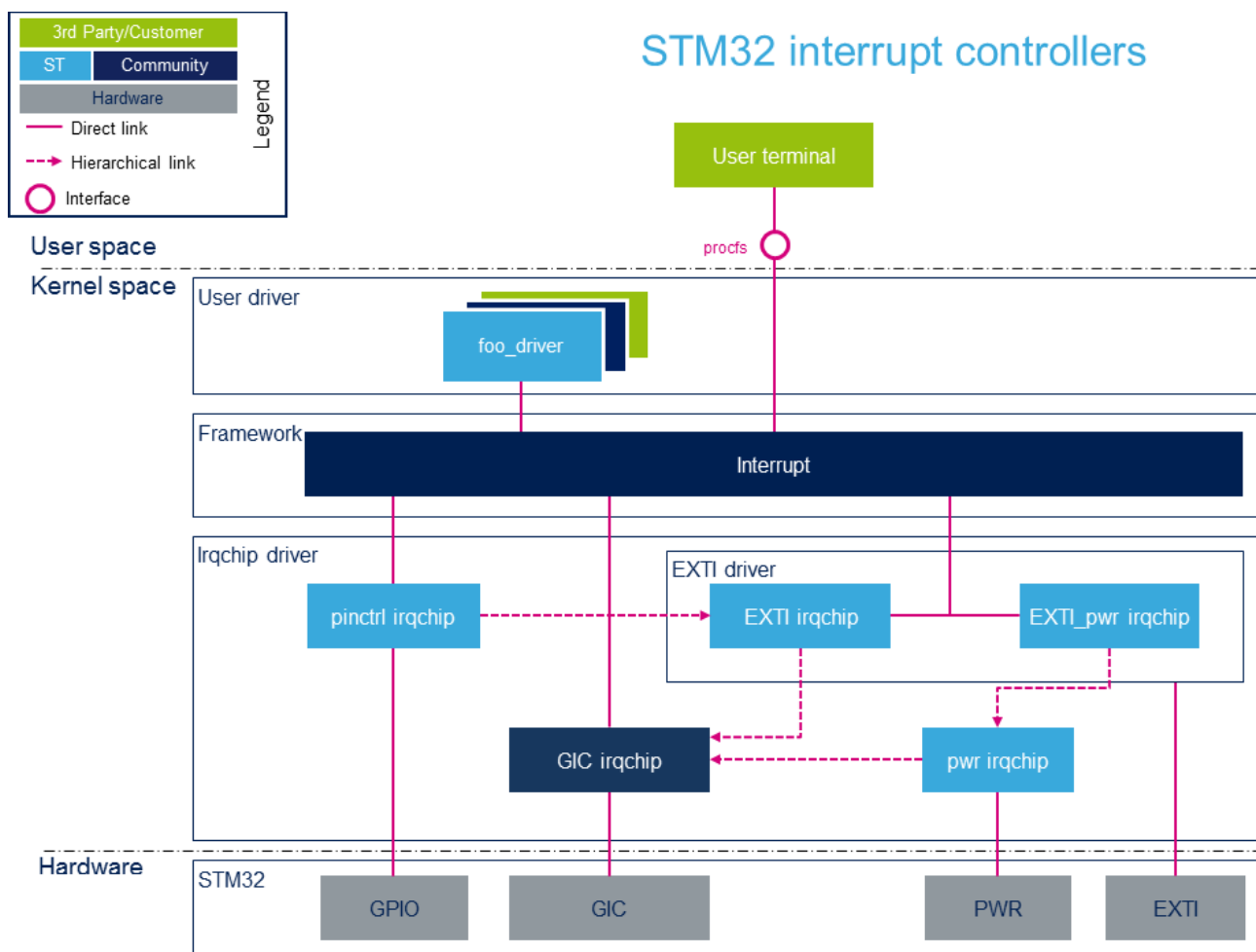
For more information refer to Linux® kernel documentation in [core-api/genericirq.html<sup>\[1\]</sup>](#).

## 2 STM32 interrupt topology

As explain in [Framework purpose](#), the irqchip driver makes the interface with the hardware to configure and manage an interrupt. On STM32MP1 devices, a hardware interrupt can be generated by GIC, EXTI, PWR or GPIO. Several irqchip drivers are consequently required, one per hardware block.

The next section provides topology information for each kind of interrupt source.

### 2.1 Overview



### 2.2 Component description

- **procs:** provides interrupt information to the user space.
- **foo\_driver:** device driver requesting an interrupt line.
- **interrupt framework:** generic part described in the [Framework purpose](#) section.



- 
- **Irqchips:**
    - **GIC irqchip:** GIC irqchip driver part. This irqchip driver is used when a GIC interrupt is directly requested by a device. This is the case for all the peripheral interrupts that are not wakeup sources. This irqchip is in charge of controlling the GIC internal peripheral (hardware). An example of GIC irqchip usage is available [here](#).
    - **EXTI irqchip:** EXTI irqchip driver part. This irqchip driver is used when an EXTI interrupt (EXTernal Interrupt) is requested by a device. This kind of interrupts is used to wake up the system from **low-power mode**. This irqchip directly controls the EXTI internal peripheral but it is also linked to the **gic irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
    - **EXTI\_pwr irqchip:** EXTI\_pwr irqchip driver part. This irqchip driver is used when an EXTI interrupt mapped to a wakeup pin is requested by a device. This kind of interrupts is used to wake up the system from the deepest low-power mode (more details about low-power modes are available [here](#)). This irqchip directly controls the EXTI internal peripheral but it is also linked to the **pwr irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. The **pwr irqchip** in turn controls the PWR internal peripheral and it is also linked to the **gic irqchip** through the same hierarchical irq domain mechanism<sup>[2]</sup>. These hierarchical links are transparent for the requester.
    - **Pinctrl irqchip:** Pinctrl irqchip driver part. This irqchip driver is used when a device wants to configure/request a GPIO as an interrupt. It directly controls the GPIO internal peripheral but it is also linked to the **EXTI irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
  - **STM32 hardware peripherals:** GIC, EXTI, PWR, GPIO



## 3 API description

The kernel space API is the interface for declaring and managing interrupts. The user space interface is used to monitor interrupt information or set interrupt affinity.

### 3.1 User space API

`procds` performs the following tasks:

- It provides information about interrupts such as the virtual number, the hardware ID and the irqchip used (see chapter 1.2 *Kernel data of /proc kernel documentation*<sup>[3]</sup>).

```

root@stm32mp1:~# cat /proc/interrupts
          CPU0           CPU1
17:         0             0      GIC-0  37 Level      rcc irq
20:    7509664    7509640      GIC-0  27 Level      arch_timer
22:         0             0      GIC-0 232 Level      arm-pmu
23:         0             0      GIC-0 233 Level      arm-pmu
24:         0             0      GIC-0  68 Level      4000b000.audio-controller
26:         0             0  stm32-exti-h 27 Edge      4000e000.serial:wakeup
27:     8915             0      GIC-0  84 Level      40010000.serial
28:         0             0  stm32-exti-h 30 Edge      40010000.serial:wakeup
29:         654             0      GIC-0  63 Level      40012000.i2c
30:         0             0      GIC-0  64 Level      40012000.i2c
31:         0             0  stm32-exti-h 21 Edge      40012000.i2c:wakeup
33:         0             0      GIC-0 123 Level      4400b004.audio-controller, 4400b024.
audio-controller
...

```

- It configures interrupt affinity<sup>[4]</sup>, that is assigns an interrupt to a dedicated CPU.

### 3.2 Kernel space API

The main kernel API drivers for users are the following:

- **`devm_request_irq`**: requests an interrupt.
- **`devm_free_irq`**: frees an interrupt.
- **`enable_irq`**: enables a requested interrupt.
- **`disable_irq`**: disables a requested interrupt.
- **`enable_irq_wake`**: enables a requested interrupt that could wake up the system.
- **`disable_irq_wake`**: disables a requested interrupt that could wake up the system.

... The available routines can be found in Linux<sup>®</sup> kernel header file: *include/linux/interrupt.h*<sup>[5]</sup>.



## 4 Configuration

### 4.1 Kernel configuration

The interrupt framework and irqchip drivers are enabled by default.

### 4.2 Device tree configuration

The generic way to declare an interrupt in the device tree is declared in Linux<sup>®</sup> kernel documentation in: *Documentation/devicetree/bindings/interrupt-controller/interrupts.txt* <sup>[6]</sup>.

However each irqchip driver has his own bindings description. The below chapters provide the link to the bindings documentation for each interrupt controller as well as a simple example of interrupt declaration.

#### 4.2.1 GIC irqchip

- *Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt* <sup>[7]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts = <&intc GIC_SPI 72 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.2 EXTI irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti 26 IRQ_TYPE_EDGE_RISING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.3 EXTI\_PWR irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti_pwr 55 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```





---

#### 4.2.4 pinctrl irqchip

- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&gpioa 14 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.5 pwr irqchip

Pwr irqchip has not to be used directly. User has to use *exti\_pwr* to invoke pwr irqchip thanks to the hierarchical implementation.



## 5 References

- Generic IRQ documentation
- 2.02.12.22.3 [https://www.kernel.org/doc/Documentation/IRQ-domain.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-domain.txt(master)), IRQ domain documentation
- [https://www.kernel.org/doc/Documentation/filesystems/proc.txt\(master\)](https://www.kernel.org/doc/Documentation/filesystems/proc.txt(master)), User space /proc documentation
- [https://www.kernel.org/doc/Documentation/IRQ-affinity.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-affinity.txt(master)), IRQ affinity documentation
- [include/linux/interrupt.h ], Kernel interrupt API
- Generic interrupts bindings documentation, Generic interrupts bindings documentation
- [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt(master)), GIC controller binding documentation
- 8.08.1 [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt(master)), Exti interrupts bindings documentation

Linux<sup>®</sup> is a registered trademark of Linus Torvalds.

Application programming interface

Generic Interrupt Controller

External Interrupt

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Process File System (See <https://en.wikipedia.org/wiki/Procsfs> for more details)

foo\_driver could be any driver that needs to control a GPIO

Central processing unit

Serial Peripheral Interface

Stable: 17.03.2021 - 09:41 / Revision: 17.03.2021 - 09:41

A quality version of this page, approved on 17 March 2021, was based off this revision.

This article explains stm32mp157 interrupt topology and its management on Linux<sup>®</sup> environment.

### Contents

1 Framework purpose .....	36
2 STM32 interrupt topology .....	37
2.1 Overview .....	37
2.2 Component description .....	37
3 API description .....	39
3.1 User space API .....	39
3.2 Kernel space API .....	39
4 Configuration .....	40
4.1 Kernel configuration .....	40
4.2 Device tree configuration .....	40
4.2.1 GIC irqchip .....	40
4.2.2 EXTI irqchip .....	40



---

4.2.3 EXTI_PWR irqchip .....	40
4.2.4 pinctrl irqchip .....	41
4.2.5 pwr irqchip .....	41
5 References .....	42



---

## 1 Framework purpose

---

The Linux® kernel software layer that handles the interrupts is splitted into two parts:"

- A generic part:
  - providing a common API to request and configure an interrupt line.
  - creating a virtual mapping for all interrupts in order to have only one ID per interrupt.
  - providing callback for irqchip registering.
- An irqchip driver part:
  - handling hardware accesses and managing specific features.

For more information refer to Linux® kernel documentation in [core-api/genericirq.html<sup>\[1\]</sup>](#).

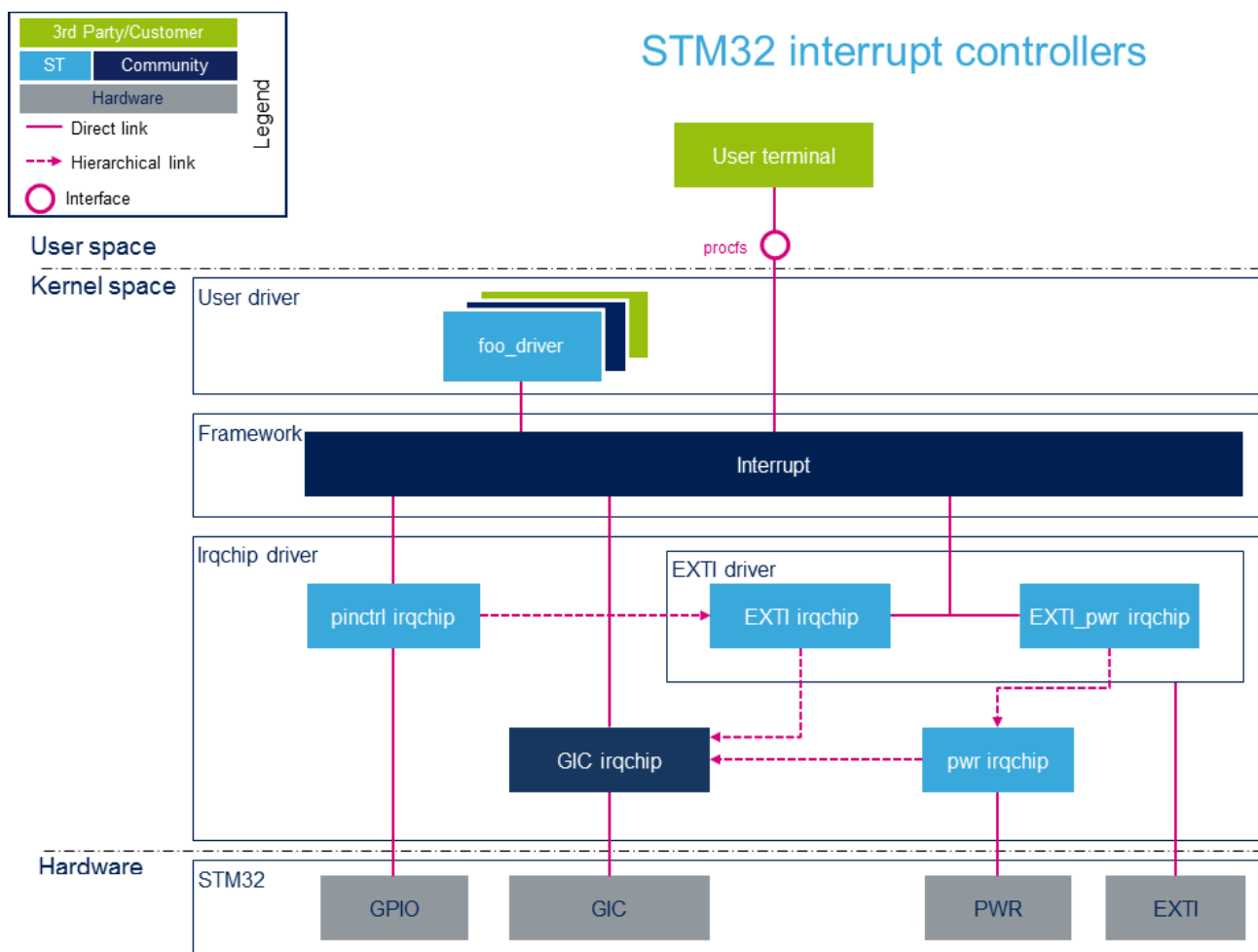


## 2 STM32 interrupt topology

As explain in [Framework purpose](#), the irqchip driver makes the interface with the hardware to configure and manage an interrupt. On STM32MP1 devices, a hardware interrupt can be generated by GIC, EXTI, PWR or GPIO. Several irqchip drivers are consequently required, one per hardware block.

The next section provides topology information for each kind of interrupt source.

### 2.1 Overview



### 2.2 Component description

- **procs:** provides interrupt information to the user space.
- **foo\_driver:** device driver requesting an interrupt line.
- **interrupt framework:** generic part described in the [Framework purpose](#) section.



- 
- **Irqchips:**
    - **GIC irqchip:** GIC irqchip driver part. This irqchip driver is used when a GIC interrupt is directly requested by a device. This is the case for all the peripheral interrupts that are not wakeup sources. This irqchip is in charge of controlling the GIC internal peripheral (hardware). An example of GIC irqchip usage is available [here](#).
    - **EXTI irqchip:** EXTI irqchip driver part. This irqchip driver is used when an EXTI interrupt (EXTernal Interrupt) is requested by a device. This kind of interrupts is used to wake up the system from [low-power mode](#). This irqchip directly controls the EXTI internal peripheral but it is also linked to the **gic irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
    - **EXTI\_pwr irqchip:** EXTI\_pwr irqchip driver part. This irqchip driver is used when an EXTI interrupt mapped to a wakeup pin is requested by a device. This kind of interrupts is used to wake up the system from the deepest low-power mode (more details about low-power modes are available [here](#)). This irqchip directly controls the EXTI internal peripheral but it is also linked to the **pwr irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. The **pwr irqchip** in turn controls the PWR internal peripheral and it is also linked to the **gic irqchip** through the same hierarchical irq domain mechanism<sup>[2]</sup>. These hierarchical links are transparent for the requester.
    - **Pinctrl irqchip:** Pinctrl irqchip driver part. This irqchip driver is used when a device wants to configure/request a GPIO as an interrupt. It directly controls the GPIO internal peripheral but it is also linked to the **EXTI irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
  - **STM32 hardware peripherals:** GIC, EXTI, PWR, GPIO



## 3 API description

The kernel space API is the interface for declaring and managing interrupts. The user space interface is used to monitor interrupt information or set interrupt affinity.

### 3.1 User space API

`procds` performs the following tasks:

- It provides information about interrupts such as the virtual number, the hardware ID and the irqchip used (see chapter 1.2 *Kernel data of /proc* kernel documentation<sup>[3]</sup>).

```

root@stm32mp1:~# cat /proc/interrupts
          CPU0           CPU1
17:         0             0      GIC-0  37 Level      rcc irq
20:    7509664       7509640      GIC-0  27 Level      arch_timer
22:         0             0      GIC-0 232 Level      arm-pmu
23:         0             0      GIC-0 233 Level      arm-pmu
24:         0             0      GIC-0  68 Level      4000b000.audio-controller
26:         0             0  stm32-exti-h 27 Edge      4000e000.serial:wakeup
27:         8915           0      GIC-0  84 Level      40010000.serial
28:         0             0  stm32-exti-h 30 Edge      40010000.serial:wakeup
29:         654           0      GIC-0  63 Level      40012000.i2c
30:         0             0      GIC-0  64 Level      40012000.i2c
31:         0             0  stm32-exti-h 21 Edge      40012000.i2c:wakeup
33:         0             0      GIC-0 123 Level      4400b004.audio-controller, 4400b024.
audio-controller
...

```

- It configures interrupt affinity<sup>[4]</sup>, that is assigns an interrupt to a dedicated CPU.

### 3.2 Kernel space API

The main kernel API drivers for users are the following:

- **`devm_request_irq`**: requests an interrupt.
- **`devm_free_irq`**: frees an interrupt.
- **`enable_irq`**: enables a requested interrupt.
- **`disable_irq`**: disables a requested interrupt.
- **`enable_irq_wake`**: enables a requested interrupt that could wake up the system.
- **`disable_irq_wake`**: disables a requested interrupt that could wake up the system.

... The available routines can be found in Linux<sup>®</sup> kernel header file: `include/linux/interrupt.h`<sup>[5]</sup>.



## 4 Configuration

### 4.1 Kernel configuration

The interrupt framework and irqchip drivers are enabled by default.

### 4.2 Device tree configuration

The generic way to declare an interrupt in the device tree is declared in Linux<sup>®</sup> kernel documentation in: *Documentation/devicetree/bindings/interrupt-controller/interrupts.txt* [6].

However each irqchip driver has his own bindings description. The below chapters provide the link to the bindings documentation for each interrupt controller as well as a simple example of interrupt declaration.

#### 4.2.1 GIC irqchip

- *Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt* [7]
- Device tree usage:

```
&foo_node {
    ...
    interrupts = <&intc GIC_SPI 72 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.2 EXTI irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* [8]
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti 26 IRQ_TYPE_EDGE_RISING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.3 EXTI\_PWR irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* [8]
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti_pwr 55 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```





#### 4.2.4 pinctrl irqchip

- Device tree usage:

```
&foo_node {  
    ...  
    interrupts-extended = <&gpioa 14 IRQ_TYPE_EDGE_FALLING>;  
    interrupt-names = "foo_name";  
    ...  
};
```

#### 4.2.5 pwr irqchip

Pwr irqchip has not to be used directly. User has to use *exti\_pwr* to invoke pwr irqchip thanks to the hierarchical implementation.



## 5 References

- Generic IRQ documentation
- 2.02.12.22.3 [https://www.kernel.org/doc/Documentation/IRQ-domain.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-domain.txt(master)), IRQ domain documentation
- [https://www.kernel.org/doc/Documentation/filesystems/proc.txt\(master\)](https://www.kernel.org/doc/Documentation/filesystems/proc.txt(master)), User space /proc documentation
- [https://www.kernel.org/doc/Documentation/IRQ-affinity.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-affinity.txt(master)), IRQ affinity documentation
- [include/linux/interrupt.h ], Kernel interrupt API
- Generic interrupts bindings documentation, Generic interrupts bindings documentation
- [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt(master)), GIC controller binding documentation
- 8.08.1 [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt(master)), Exti interrupts bindings documentation

Linux<sup>®</sup> is a registered trademark of Linus Torvalds.

Application programming interface

Generic Interrupt Controller

External Interrupt

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Process File System (See <https://en.wikipedia.org/wiki/Procsfs> for more details)

foo\_driver could be any driver that needs to control a GPIO

Central processing unit

Serial Peripheral Interface

Stable: 25.09.2020 - 09:16 / Revision: 25.09.2020 - 09:15

This article explains stm32mp157 interrupt topology and its management on Linux<sup>®</sup> environment.

### Contents

1 Framework purpose .....	44
2 STM32 interrupt topology .....	45
2.1 Overview .....	45
2.2 Component description .....	45
3 API description .....	47
3.1 User space API .....	47
3.2 Kernel space API .....	47
4 Configuration .....	48
4.1 Kernel configuration .....	48
4.2 Device tree configuration .....	48
4.2.1 GIC irqchip .....	48
4.2.2 EXTI irqchip .....	48
4.2.3 EXTI_PWR irqchip .....	48



---

4.2.4 pinctrl irqchip .....	49
4.2.5 pwr irqchip .....	49
5 References .....	50



---

## 1 Framework purpose

---

The Linux® kernel software layer that handles the interrupts is splitted into two parts:"

- A generic part:
  - providing a common API to request and configure an interrupt line.
  - creating a virtual mapping for all interrupts in order to have only one ID per interrupt.
  - providing callback for irqchip registering.
- An irqchip driver part:
  - handling hardware accesses and managing specific features.

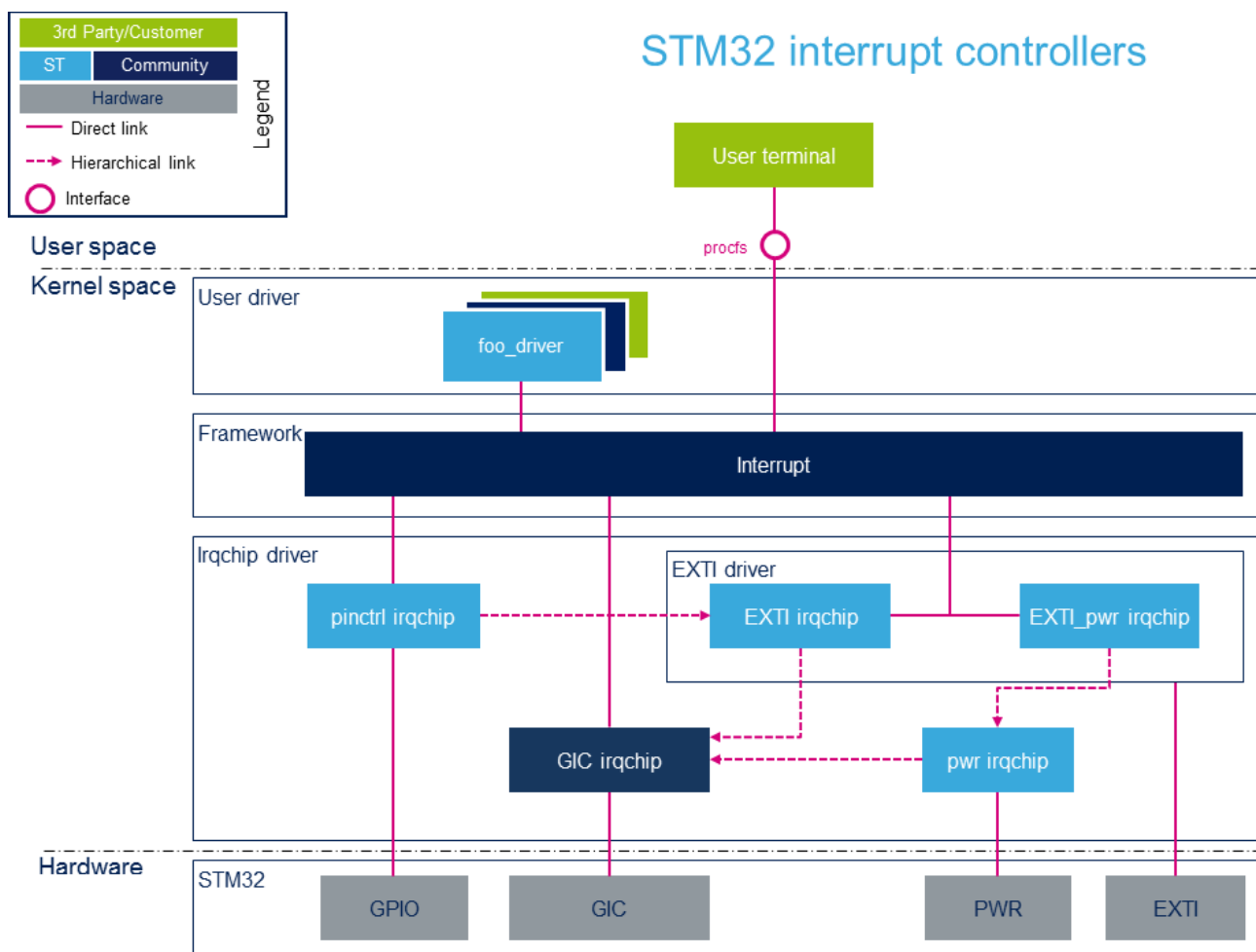
For more information refer to Linux® kernel documentation in [core-api/genericirq.html<sup>\[1\]</sup>](#).

## 2 STM32 interrupt topology

As explain in [Framework purpose](#), the irqchip driver makes the interface with the hardware to configure and manage an interrupt. On STM32MP1 devices, a hardware interrupt can be generated by GIC, EXTI, PWR or GPIO. Several irqchip drivers are consequently required, one per hardware block.

The next section provides topology information for each kind of interrupt source.

### 2.1 Overview



### 2.2 Component description

- **procf**: provides interrupt information to the user space.
- **foo\_driver**: device driver requesting an interrupt line.
- **interrupt framework**: generic part described in the [Framework purpose](#) section.



- 
- **Irqchips:**
    - **GIC irqchip:** GIC irqchip driver part. This irqchip driver is used when a GIC interrupt is directly requested by a device. This is the case for all the peripheral interrupts that are not wakeup sources. This irqchip is in charge of controlling the GIC internal peripheral (hardware). An example of GIC irqchip usage is available [here](#).
    - **EXTI irqchip:** EXTI irqchip driver part. This irqchip driver is used when an EXTI interrupt (EXTernal Interrupt) is requested by a device. This kind of interrupts is used to wake up the system from **low-power mode**. This irqchip directly controls the EXTI internal peripheral but it is also linked to the **gic irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
    - **EXTI\_pwr irqchip:** EXTI\_pwr irqchip driver part. This irqchip driver is used when an EXTI interrupt mapped to a wakeup pin is requested by a device. This kind of interrupts is used to wake up the system from the deepest low-power mode (more details about low-power modes are available [here](#)). This irqchip directly controls the EXTI internal peripheral but it is also linked to the **pwr irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. The **pwr irqchip** in turn controls the PWR internal peripheral and it is also linked to the **gic irqchip** through the same hierarchical irq domain mechanism<sup>[2]</sup>. These hierarchical links are transparent for the requester.
    - **Pinctrl irqchip:** Pinctrl irqchip driver part. This irqchip driver is used when a device wants to configure/request a GPIO as an interrupt. It directly controls the GPIO internal peripheral but it is also linked to the **EXTI irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
  - **STM32 hardware peripherals:** GIC, EXTI, PWR, GPIO



## 3 API description

The kernel space API is the interface for declaring and managing interrupts. The user space interface is used to monitor interrupt information or set interrupt affinity.

### 3.1 User space API

procs performs the following tasks:

- It provides information about interrupts such as the virtual number, the hardware ID and the irqchip used (see chapter 1.2 *Kernel data of /proc kernel documentation*<sup>[3]</sup>).

```

root@stm32mp1:~# cat /proc/interrupts
          CPU0           CPU1
17:         0             0      GIC-0  37 Level      rcc irq
20:    7509664    7509640      GIC-0  27 Level      arch_timer
22:         0             0      GIC-0 232 Level      arm-pmu
23:         0             0      GIC-0 233 Level      arm-pmu
24:         0             0      GIC-0  68 Level      4000b000.audio-controller
26:         0             0  stm32-exti-h 27 Edge      4000e000.serial:wakeup
27:     8915             0      GIC-0  84 Level      40010000.serial
28:         0             0  stm32-exti-h 30 Edge      40010000.serial:wakeup
29:         654             0      GIC-0  63 Level      40012000.i2c
30:         0             0      GIC-0  64 Level      40012000.i2c
31:         0             0  stm32-exti-h 21 Edge      40012000.i2c:wakeup
33:         0             0      GIC-0 123 Level      4400b004.audio-controller, 4400b024.
audio-controller
...

```

- It configures interrupt affinity<sup>[4]</sup>, that is assigns an interrupt to a dedicated CPU.

### 3.2 Kernel space API

The main kernel API drivers for users are the following:

- **devm\_request\_irq**: requests an interrupt.
- **devm\_free\_irq**: frees an interrupt.
- **enable\_irq**: enables a requested interrupt.
- **disable\_irq**: disables a requested interrupt.
- **enable\_irq\_wake**: enables a requested interrupt that could wake up the system.
- **disable\_irq\_wake**: disables a requested interrupt that could wake up the system.

... The available routines can be found in Linux<sup>®</sup> kernel header file: *include/linux/interrupt.h*<sup>[5]</sup>.



## 4 Configuration

### 4.1 Kernel configuration

The interrupt framework and irqchip drivers are enabled by default.

### 4.2 Device tree configuration

The generic way to declare an interrupt in the device tree is declared in Linux<sup>®</sup> kernel documentation in: *Documentation/devicetree/bindings/interrupt-controller/interrupts.txt* <sup>[6]</sup>.

However each irqchip driver has his own bindings description. The below chapters provide the link to the bindings documentation for each interrupt controller as well as a simple example of interrupt declaration.

#### 4.2.1 GIC irqchip

- *Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt* <sup>[7]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts = <&intc GIC_SPI 72 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.2 EXTI irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti 26 IRQ_TYPE_EDGE_RISING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.3 EXTI\_PWR irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti_pwr 55 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```





#### 4.2.4 pinctrl irqchip

- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&gpioa 14 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.5 pwr irqchip

Pwr irqchip has not to be used directly. User has to use *exti\_pwr* to invoke pwr irqchip thanks to the hierarchical implementation.



## 5 References

- Generic IRQ documentation
- 2.02.12.22.3 [https://www.kernel.org/doc/Documentation/IRQ-domain.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-domain.txt(master)), IRQ domain documentation
- [https://www.kernel.org/doc/Documentation/filesystems/proc.txt\(master\)](https://www.kernel.org/doc/Documentation/filesystems/proc.txt(master)), User space /proc documentation
- [https://www.kernel.org/doc/Documentation/IRQ-affinity.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-affinity.txt(master)), IRQ affinity documentation
- [include/linux/interrupt.h ], Kernel interrupt API
- Generic interrupts bindings documentation, Generic interrupts bindings documentation
- [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt(master)), GIC controller binding documentation
- 8.08.1 [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt(master)), Exti interrupts bindings documentation

Linux<sup>®</sup> is a registered trademark of Linus Torvalds.

Application programming interface

Generic Interrupt Controller

External Interrupt

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Process File System (See <https://en.wikipedia.org/wiki/Procsfs> for more details)

foo\_driver could be any driver that needs to control a GPIO

Central processing unit

Serial Peripheral Interface

Stable: 01.12.2020 - 10:35 / Revision: 01.12.2020 - 09:20

This article explains stm32mp157 interrupt topology and its management on Linux<sup>®</sup> environment.

### Contents

1 Framework purpose .....	52
2 STM32 interrupt topology .....	53
2.1 Overview .....	53
2.2 Component description .....	53
3 API description .....	55
3.1 User space API .....	55
3.2 Kernel space API .....	55
4 Configuration .....	56
4.1 Kernel configuration .....	56
4.2 Device tree configuration .....	56
4.2.1 GIC irqchip .....	56
4.2.2 EXTI irqchip .....	56
4.2.3 EXTI_PWR irqchip .....	56



---

4.2.4 pinctrl irqchip .....	57
4.2.5 pwr irqchip .....	57
5 References .....	58



---

## 1 Framework purpose

---

The Linux<sup>®</sup> kernel software layer that handles the interrupts is splitted into two parts:"

- A generic part:
  - providing a common API to request and configure an interrupt line.
  - creating a virtual mapping for all interrupts in order to have only one ID per interrupt.
  - providing callback for irqchip registering.
- An irqchip driver part:
  - handling hardware accesses and managing specific features.

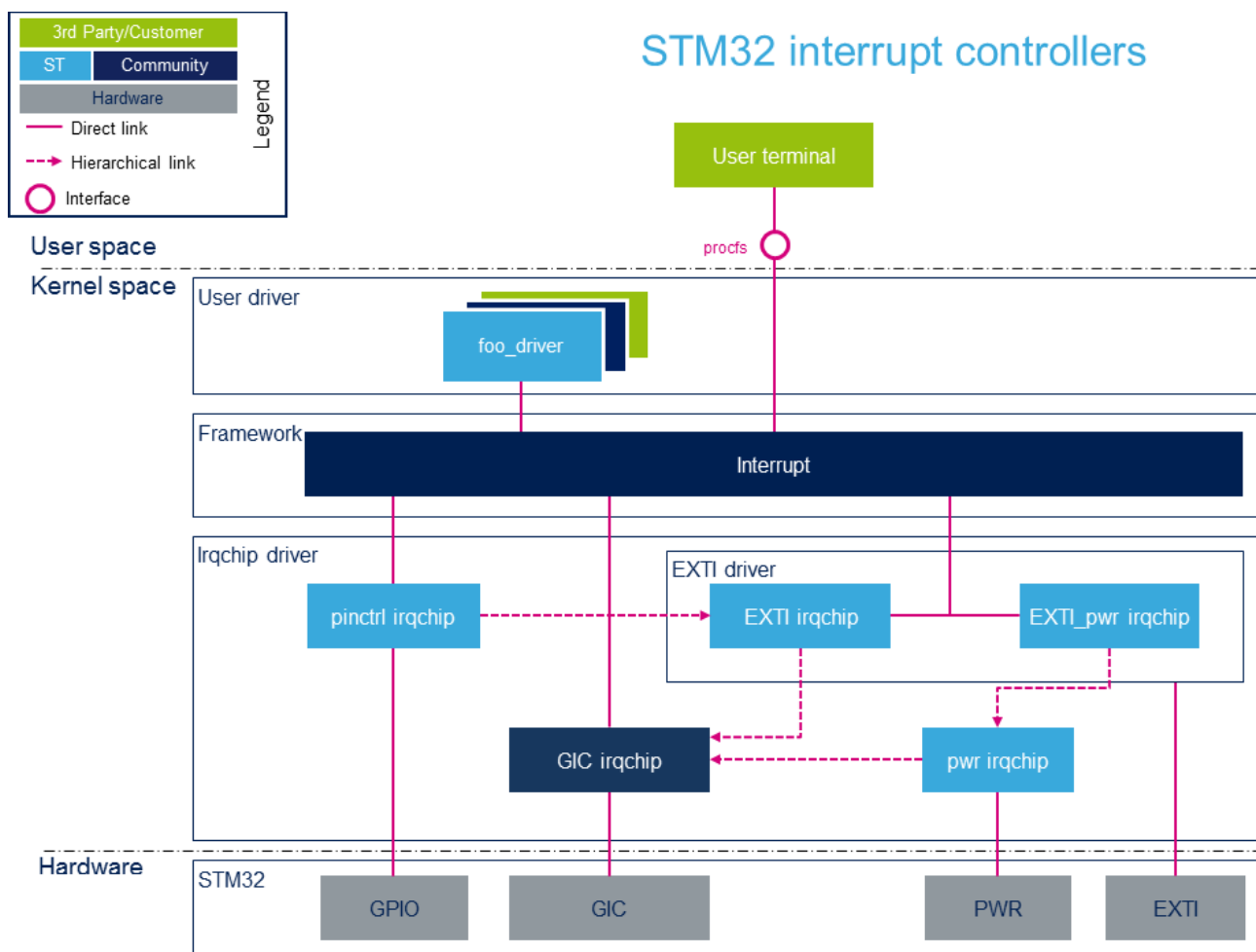
For more information refer to Linux<sup>®</sup> kernel documentation in [core-api/genericirq.htm<sup>\[1\]</sup>](#).

## 2 STM32 interrupt topology

As explain in [Framework purpose](#), the irqchip driver makes the interface with the hardware to configure and manage an interrupt. On STM32MP1 devices, a hardware interrupt can be generated by GIC, EXTI, PWR or GPIO. Several irqchip drivers are consequently required, one per hardware block.

The next section provides topology information for each kind of interrupt source.

### 2.1 Overview



### 2.2 Component description

- **procfcs:** provides interrupt information to the user space.
- **foo\_driver:** device driver requesting an interrupt line.
- **interrupt framework:** generic part described in the [Framework purpose](#) section.



- 
- **Irqchips:**
    - **GIC irqchip:** GIC irqchip driver part. This irqchip driver is used when a GIC interrupt is directly requested by a device. This is the case for all the peripheral interrupts that are not wakeup sources. This irqchip is in charge of controlling the GIC internal peripheral (hardware). An example of GIC irqchip usage is available [here](#).
    - **EXTI irqchip:** EXTI irqchip driver part. This irqchip driver is used when an EXTI interrupt (EXTernal Interrupt) is requested by a device. This kind of interrupts is used to wake up the system from **low-power mode**. This irqchip directly controls the EXTI internal peripheral but it is also linked to the **gic irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
    - **EXTI\_pwr irqchip:** EXTI\_pwr irqchip driver part. This irqchip driver is used when an EXTI interrupt mapped to a wakeup pin is requested by a device. This kind of interrupts is used to wake up the system from the deepest low-power mode (more details about low-power modes are available [here](#)). This irqchip directly controls the EXTI internal peripheral but it is also linked to the **pwr irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. The **pwr irqchip** in turn controls the PWR internal peripheral and it is also linked to the **gic irqchip** through the same hierarchical irq domain mechanism<sup>[2]</sup>. These hierarchical links are transparent for the requester.
    - **Pinctrl irqchip:** Pinctrl irqchip driver part. This irqchip driver is used when a device wants to configure/request a GPIO as an interrupt. It directly controls the GPIO internal peripheral but it is also linked to the **EXTI irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
  - **STM32 hardware peripherals:** GIC, EXTI, PWR, GPIO



## 3 API description

The kernel space API is the interface for declaring and managing interrupts. The user space interface is used to monitor interrupt information or set interrupt affinity.

### 3.1 User space API

procs performs the following tasks:

- It provides information about interrupts such as the virtual number, the hardware ID and the irqchip used (see chapter 1.2 *Kernel data of /proc kernel documentation*<sup>[3]</sup>).

```

root@stm32mp1:~# cat /proc/interrupts
          CPU0           CPU1
17:         0             0      GIC-0  37 Level      rcc irq
20:    7509664    7509640      GIC-0  27 Level      arch_timer
22:         0             0      GIC-0 232 Level      arm-pmu
23:         0             0      GIC-0 233 Level      arm-pmu
24:         0             0      GIC-0  68 Level      4000b000.audio-controller
26:         0             0  stm32-exti-h 27 Edge      4000e000.serial:wakeup
27:     8915             0      GIC-0  84 Level      40010000.serial
28:         0             0  stm32-exti-h 30 Edge      40010000.serial:wakeup
29:         654             0      GIC-0  63 Level      40012000.i2c
30:         0             0      GIC-0  64 Level      40012000.i2c
31:         0             0  stm32-exti-h 21 Edge      40012000.i2c:wakeup
33:         0             0      GIC-0 123 Level      4400b004.audio-controller, 4400b024.
audio-controller
...

```

- It configures interrupt affinity<sup>[4]</sup>, that is assigns an interrupt to a dedicated CPU.

### 3.2 Kernel space API

The main kernel API drivers for users are the following:

- **devm\_request\_irq**: requests an interrupt.
- **devm\_free\_irq**: frees an interrupt.
- **enable\_irq**: enables a requested interrupt.
- **disable\_irq**: disables a requested interrupt.
- **enable\_irq\_wake**: enables a requested interrupt that could wake up the system.
- **disable\_irq\_wake**: disables a requested interrupt that could wake up the system.

... The available routines can be found in Linux<sup>®</sup> kernel header file: *include/linux/interrupt.h*<sup>[5]</sup>.



## 4 Configuration

### 4.1 Kernel configuration

The interrupt framework and irqchip drivers are enabled by default.

### 4.2 Device tree configuration

The generic way to declare an interrupt in the device tree is declared in Linux<sup>®</sup> kernel documentation in: *Documentation/devicetree/bindings/interrupt-controller/interrupts.txt* <sup>[6]</sup>.

However each irqchip driver has his own bindings description. The below chapters provide the link to the bindings documentation for each interrupt controller as well as a simple example of interrupt declaration.

#### 4.2.1 GIC irqchip

- *Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt* <sup>[7]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts = <&intc GIC_SPI 72 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.2 EXTI irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti 26 IRQ_TYPE_EDGE_RISING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.3 EXTI\_PWR irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti_pwr 55 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```





---

#### 4.2.4 pinctrl irqchip

- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&gpioa 14 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.5 pwr irqchip

Pwr irqchip has not to be used directly. User has to use *exti\_pwr* to invoke pwr irqchip thanks to the hierarchical implementation.



## 5 References

- Generic IRQ documentation
- 2.02.12.22.3 [https://www.kernel.org/doc/Documentation/IRQ-domain.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-domain.txt(master)), IRQ domain documentation
- [https://www.kernel.org/doc/Documentation/filesystems/proc.txt\(master\)](https://www.kernel.org/doc/Documentation/filesystems/proc.txt(master)), User space /proc documentation
- [https://www.kernel.org/doc/Documentation/IRQ-affinity.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-affinity.txt(master)), IRQ affinity documentation
- [include/linux/interrupt.h ], Kernel interrupt API
- Generic interrupts bindings documentation, Generic interrupts bindings documentation
- [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt(master)), GIC controller binding documentation
- 8.08.1 [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt(master)), Exti interrupts bindings documentation

Linux<sup>®</sup> is a registered trademark of Linus Torvalds.

Application programming interface

Generic Interrupt Controller

External Interrupt

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Process File System (See <https://en.wikipedia.org/wiki/Procsfs> for more details)

foo\_driver could be any driver that needs to control a GPIO

Central processing unit

Serial Peripheral Interface

Stable: 31.01.2020 - 13:23 / Revision: 31.01.2020 - 13:16

This article explains stm32mp157 interrupt topology and its management on Linux<sup>®</sup> environment.

### Contents

1 Framework purpose .....	60
2 STM32 interrupt topology .....	61
2.1 Overview .....	61
2.2 Component description .....	61
3 API description .....	63
3.1 User space API .....	63
3.2 Kernel space API .....	63
4 Configuration .....	64
4.1 Kernel configuration .....	64
4.2 Device tree configuration .....	64
4.2.1 GIC irqchip .....	64
4.2.2 EXTI irqchip .....	64
4.2.3 EXTI_PWR irqchip .....	64



---

4.2.4 pinctrl irqchip .....	65
4.2.5 pwr irqchip .....	65
5 References .....	66



---

## 1 Framework purpose

---

The Linux<sup>®</sup> kernel software layer that handles the interrupts is splitted into two parts:"

- A generic part:
  - providing a common API to request and configure an interrupt line.
  - creating a virtual mapping for all interrupts in order to have only one ID per interrupt.
  - providing callback for irqchip registering.
- An irqchip driver part:
  - handling hardware accesses and managing specific features.

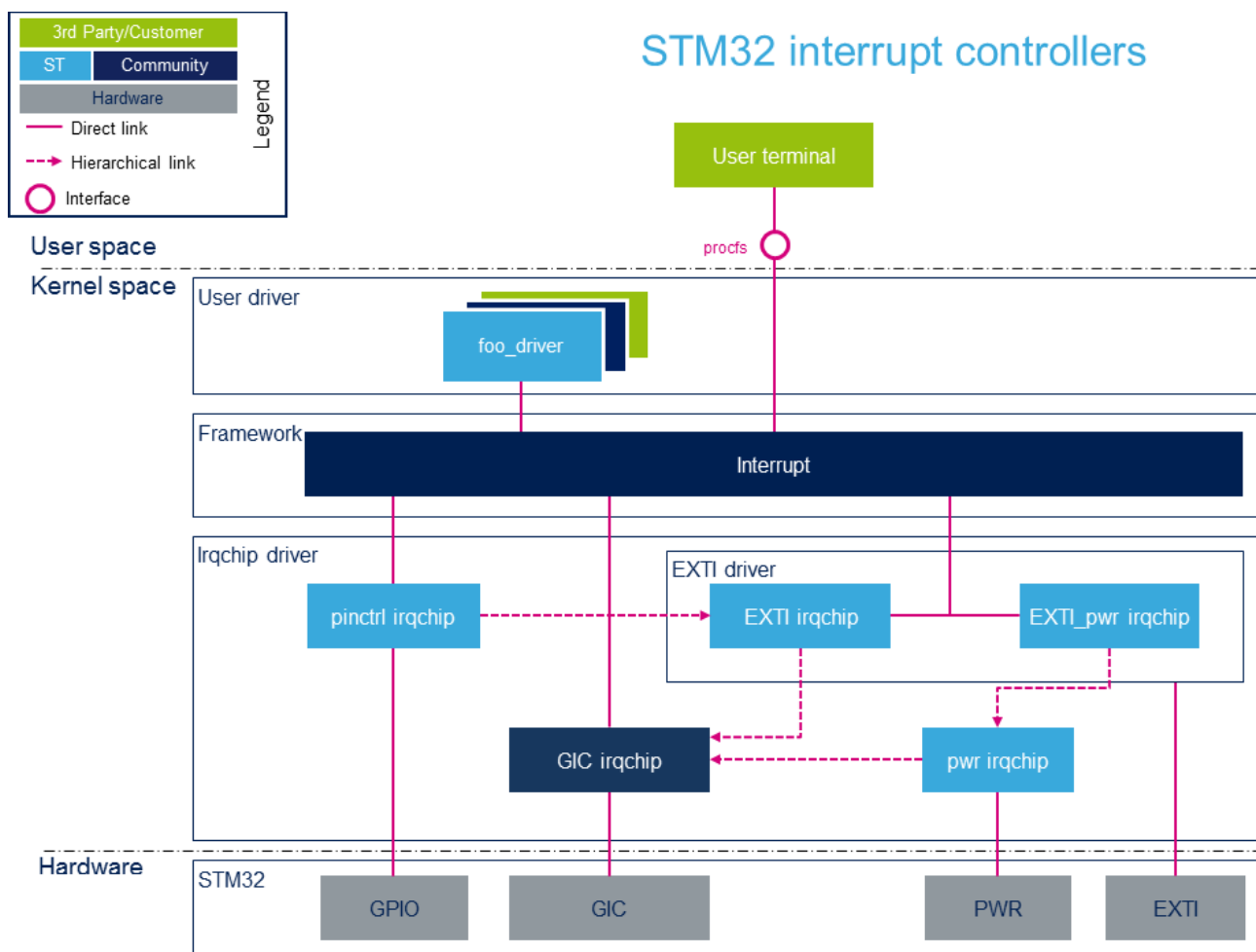
For more information refer to Linux<sup>®</sup> kernel documentation in [core-api/genericirq.htm<sup>\[1\]</sup>](#).

## 2 STM32 interrupt topology

As explain in [Framework purpose](#), the irqchip driver makes the interface with the hardware to configure and manage an interrupt. On STM32MP1 devices, a hardware interrupt can be generated by GIC, EXTI, PWR or GPIO. Several irqchip drivers are consequently required, one per hardware block.

The next section provides topology information for each kind of interrupt source.

### 2.1 Overview



### 2.2 Component description

- **procs:** provides interrupt information to the user space.
- **foo\_driver:** device driver requesting an interrupt line.
- **interrupt framework:** generic part described in the [Framework purpose](#) section.



- 
- **Irqchips:**
    - **GIC irqchip:** GIC irqchip driver part. This irqchip driver is used when a GIC interrupt is directly requested by a device. This is the case for all the peripheral interrupts that are not wakeup sources. This irqchip is in charge of controlling the GIC internal peripheral (hardware). An example of GIC irqchip usage is available [here](#).
    - **EXTI irqchip:** EXTI irqchip driver part. This irqchip driver is used when an EXTI interrupt (EXTeRnal Interrupt) is requested by a device. This kind of interrupts is used to wake up the system from **low-power mode**. This irqchip directly controls the EXTI internal peripheral but it is also linked to the **gic irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
    - **EXTI\_pwr irqchip:** EXTI\_pwr irqchip driver part. This irqchip driver is used when an EXTI interrupt mapped to a wakeup pin is requested by a device. This kind of interrupts is used to wake up the system from the deepest low-power mode (more details about low-power modes are available [here](#)). This irqchip directly controls the EXTI internal peripheral but it is also linked to the **pwr irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. The **pwr irqchip** in turn controls the PWR internal peripheral and it is also linked to the **gic irqchip** through the same hierarchical irq domain mechanism<sup>[2]</sup>. These hierarchical links are transparent for the requester.
    - **Pinctrl irqchip:** Pinctrl irqchip driver part. This irqchip driver is used when a device wants to configure/request a GPIO as an interrupt. It directly controls the GPIO internal peripheral but it is also linked to the **EXTI irqchip** through the hierarchical irq domain mechanism<sup>[2]</sup>. This link is transparent for the requester.
  - **STM32 hardware peripherals:** GIC, EXTI, PWR, GPIO



## 3 API description

The kernel space API is the interface for declaring and managing interrupts. The user space interface is used to monitor interrupt information or set interrupt affinity.

### 3.1 User space API

`procds` performs the following tasks:

- It provides information about interrupts such as the virtual number, the hardware ID and the irqchip used (see chapter 1.2 *Kernel data of /proc kernel documentation*<sup>[3]</sup>).

```

root@stm32mp1:~# cat /proc/interrupts
          CPU0           CPU1
17:         0             0      GIC-0  37 Level      rcc irq
20:    7509664    7509640      GIC-0  27 Level      arch_timer
22:         0             0      GIC-0 232 Level      arm-pmu
23:         0             0      GIC-0 233 Level      arm-pmu
24:         0             0      GIC-0  68 Level      4000b000.audio-controller
26:         0             0  stm32-exti-h 27 Edge      4000e000.serial:wakeup
27:     8915             0      GIC-0  84 Level      40010000.serial
28:         0             0  stm32-exti-h 30 Edge      40010000.serial:wakeup
29:         654             0      GIC-0  63 Level      40012000.i2c
30:         0             0      GIC-0  64 Level      40012000.i2c
31:         0             0  stm32-exti-h 21 Edge      40012000.i2c:wakeup
33:         0             0      GIC-0 123 Level      4400b004.audio-controller, 4400b024.
audio-controller
...

```

- It configures interrupt affinity<sup>[4]</sup>, that is assigns an interrupt to a dedicated CPU.

### 3.2 Kernel space API

The main kernel API drivers for users are the following:

- **`devm_request_irq`**: requests an interrupt.
- **`devm_free_irq`**: frees an interrupt.
- **`enable_irq`**: enables a requested interrupt.
- **`disable_irq`**: disables a requested interrupt.
- **`enable_irq_wake`**: enables a requested interrupt that could wake up the system.
- **`disable_irq_wake`**: disables a requested interrupt that could wake up the system.

... The available routines can be found in Linux<sup>®</sup> kernel header file: *include/linux/interrupt.h*<sup>[5]</sup>.



## 4 Configuration

### 4.1 Kernel configuration

The interrupt framework and irqchip drivers are enabled by default.

### 4.2 Device tree configuration

The generic way to declare an interrupt in the device tree is declared in Linux<sup>®</sup> kernel documentation in: *Documentation/devicetree/bindings/interrupt-controller/interrupts.txt* <sup>[6]</sup>.

However each irqchip driver has his own bindings description. The below chapters provide the link to the bindings documentation for each interrupt controller as well as a simple example of interrupt declaration.

#### 4.2.1 GIC irqchip

- *Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt* <sup>[7]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts = <&intc GIC_SPI 72 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.2 EXTI irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti 26 IRQ_TYPE_EDGE_RISING>;
    interrupt-names = "foo_name";
    ...
};
```

#### 4.2.3 EXTI\_PWR irqchip

- *Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt* <sup>[8]</sup>
- Device tree usage:

```
&foo_node {
    ...
    interrupts-extended = <&exti_pwr 55 IRQ_TYPE_EDGE_FALLING>;
    interrupt-names = "foo_name";
    ...
};
```





---

#### 4.2.4 pinctrl irqchip

- Device tree usage:

```
&foo_node {  
    ...  
    interrupts-extended = <&gpioa 14 IRQ_TYPE_EDGE_FALLING>;  
    interrupt-names = "foo_name";  
    ...  
};
```

#### 4.2.5 pwr irqchip

Pwr irqchip has not to be used directly. User has to use *exti\_pwr* to invoke pwr irqchip thanks to the hierarchical implementation.



## 5 References

- Generic IRQ documentation
- 2.02.12.22.3 [https://www.kernel.org/doc/Documentation/IRQ-domain.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-domain.txt(master)), IRQ domain documentation
- [https://www.kernel.org/doc/Documentation/filesystems/proc.txt\(master\)](https://www.kernel.org/doc/Documentation/filesystems/proc.txt(master)), User space /proc documentation
- [https://www.kernel.org/doc/Documentation/IRQ-affinity.txt\(master\)](https://www.kernel.org/doc/Documentation/IRQ-affinity.txt(master)), IRQ affinity documentation
- [include/linux/interrupt.h ], Kernel interrupt API
- Generic interrupts bindings documentation, Generic interrupts bindings documentation
- [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm,gic.txt(master)), GIC controller binding documentation
- 8.08.1 [https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt\(master\)](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt(master)), Exti interrupts bindings documentation

Linux<sup>®</sup> is a registered trademark of Linus Torvalds.

Application programming interface

Generic Interrupt Controller

External Interrupt

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Process File System (See <https://en.wikipedia.org/wiki/Procfs> for more details)

foo\_driver could be any driver that needs to control a GPIO

Central processing unit

Serial Peripheral Interface