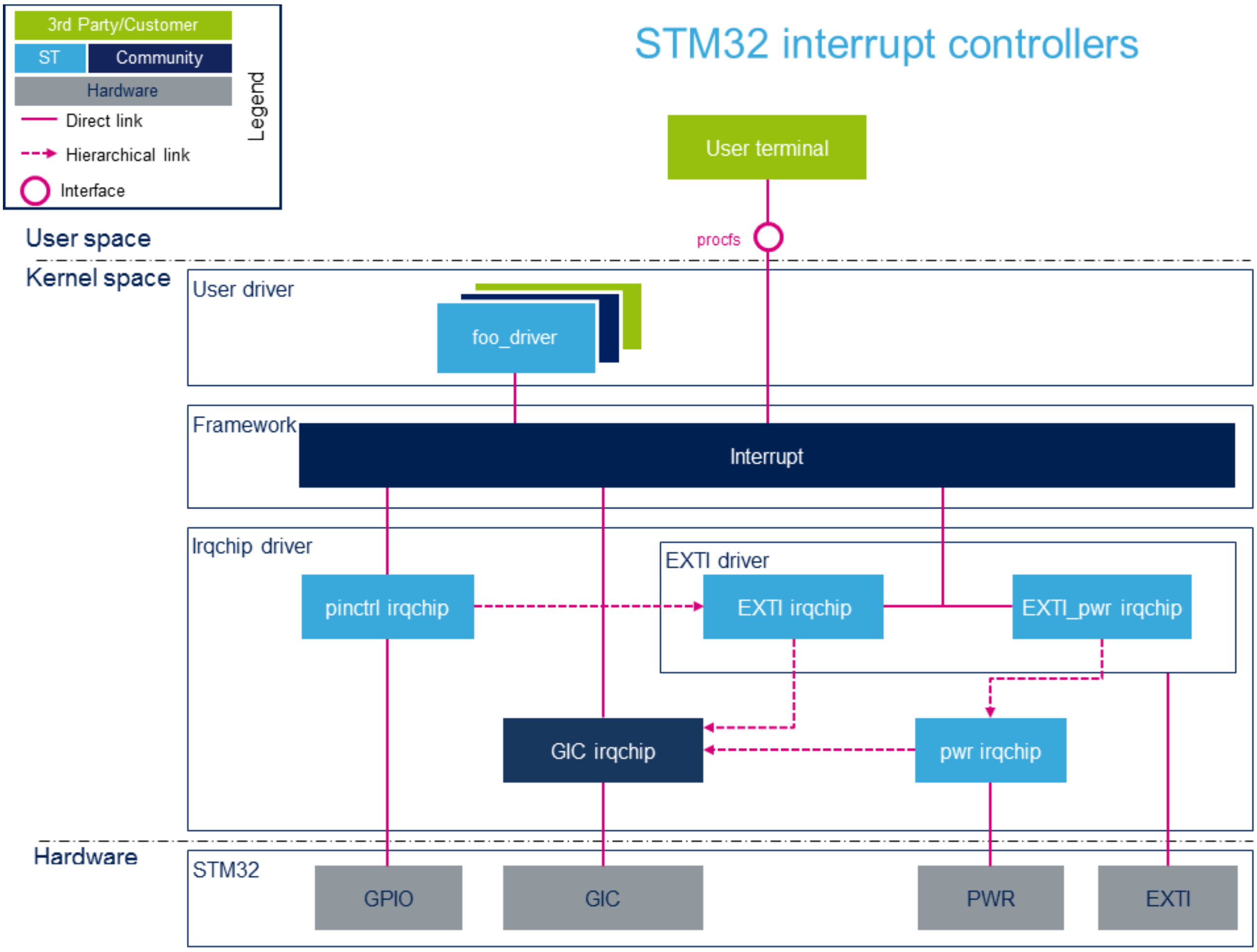
File:Interrupts overview.png

Contents

File:Interrupts overview.png

- File
- File history
- File usage



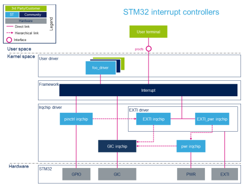
Size of this preview: 800 × 600 pixels. Other resolutions: 320 × 240 pixels | 960 × 720 pixels.
Original file (960 × 720 pixels, file size: 45 KB, MIME type: image/png)
A quality version of this page, approved on *17 June 2020*, was based off this revision.

Source
STMicroelectronics

File history

Click on a date/time to view the file as it appeared at that time.

(newest | oldest) View (newer 250 | older 250) (10 | 20 | 50 | 100 | 250 | 500)

	Date/Time	Thumbnail	Dimensions	User	Comment
	09:58, 3 January 2019		960 × 720 (49 KB)	Frq08678 (talk | contribs)	

(newest | oldest) View (newer 250 | older 250) (10 | 20 | 50 | 100 | 250 | 500)

- You cannot overwrite this file.

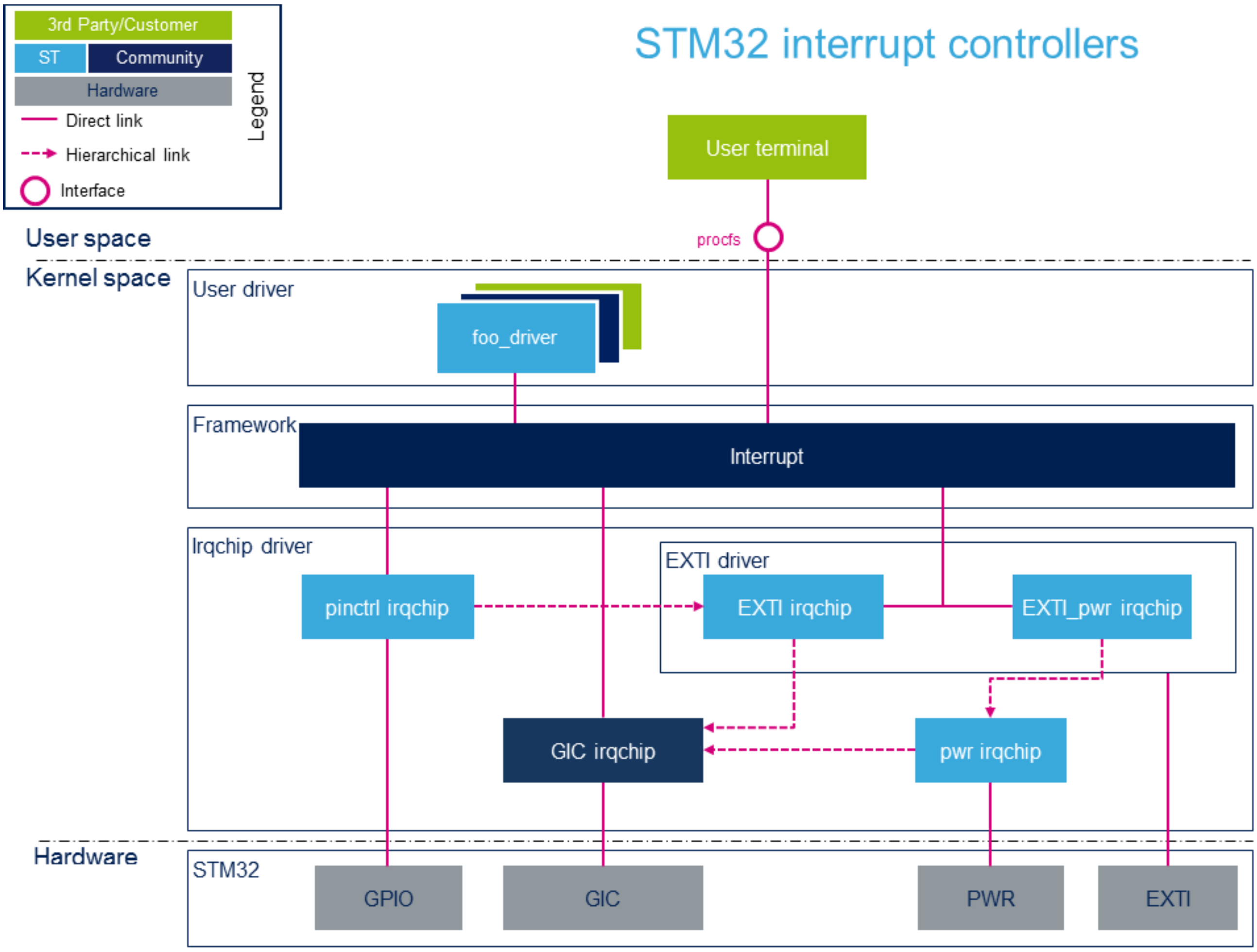
File usage

The following page links to this file:

- Interrupt overview

Stable: 17.06.2020 - 15:30 / Revision: 24.03.2020 - 09:03

- File
- File history
- File usage



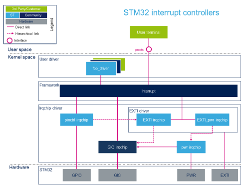
Size of this preview: 800 × 600 pixels. Other resolutions: 320 × 240 pixels | 960 × 720 pixels.
Original file (960 × 720 pixels, file size: 45 KB, MIME type: image/png)
A quality version of this page, approved on *17 June 2020*, was based off this revision.

Source
STMicroelectronics

File history

Click on a date/time to view the file as it appeared at that time.

(newest | oldest) View (newer 250 | older 250) (10 | 20 | 50 | 100 | 250 | 500)

	Date/Time	Thumbnail	Dimensions	User	Comment
	09:58, 3 January 2019		960 × 720 (49 KB)	Frq08678 (talk | contribs)	

(newest | oldest) View (newer 250 | older 250) (10 | 20 | 50 | 100 | 250 | 500)

- You cannot overwrite this file.

File usage

The following page links to this file:

- Interrupt overview

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® environment.

Contents

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.

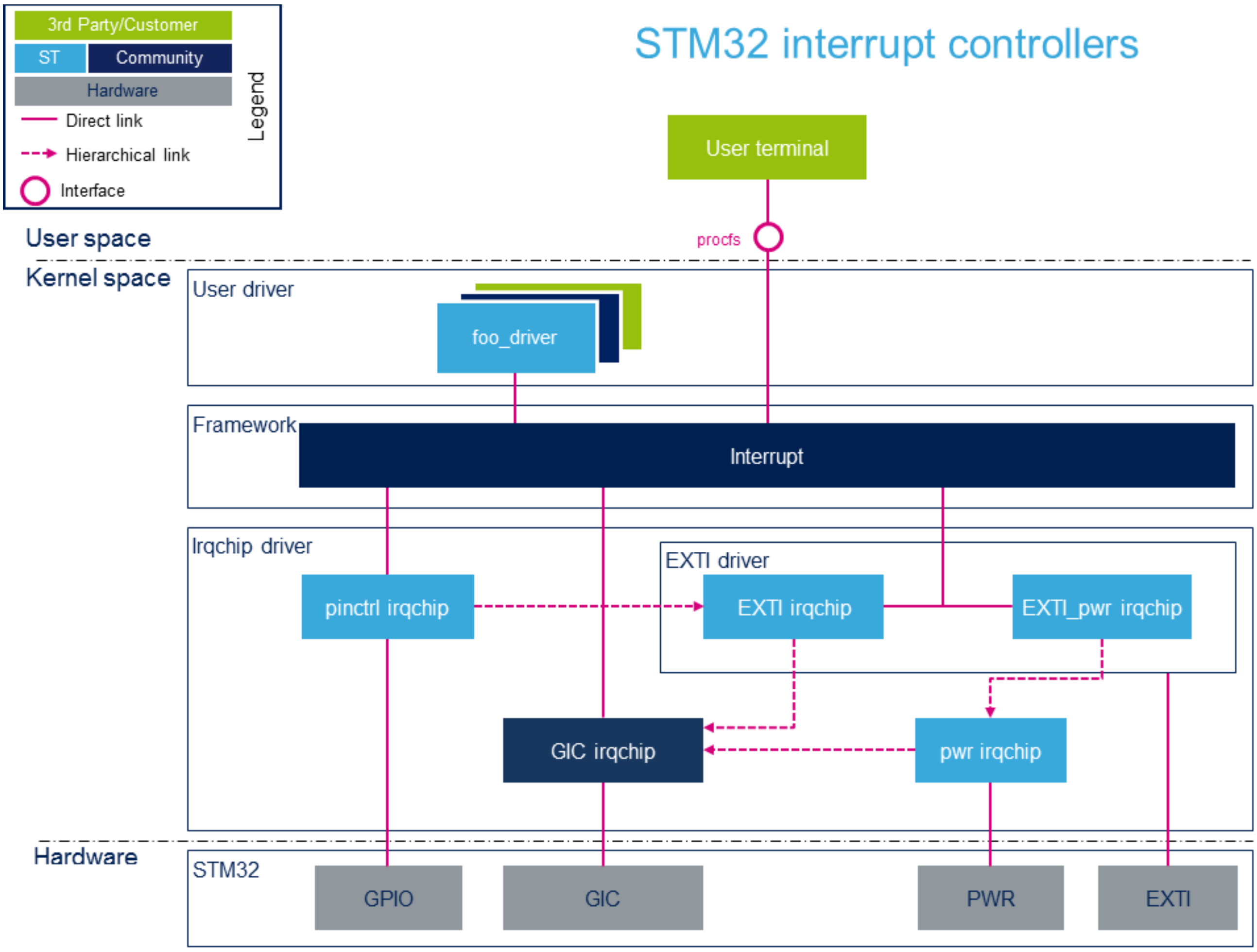
For more information refer to Linux® kernel documentation in *core-api/genericirq.html*[1].

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

- **procfs:** 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 GIG 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[2] . 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[2]. 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[2]. 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[2]. 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

procfs 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[3]).

```
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[4], 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® kernel header file: *include/linux/interrupt.h*[5].

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® 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), IRQ domain documentation
- https://www.kernel.org/doc/Documentation/filesystems/proc.txt(master), User space /proc documentation
- 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), GIC controller binding documentation
- 8.08.1 https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/st,stm32-exti.txt(master), Exti interrupts bindings documentation

Linux® 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