GPIOLib overview

A quality version of this page, approved on *11 February 2019*, was based off this revision.

Template:ArticleMainWriter Template:ArticleApprovedVersion

SUMMARY

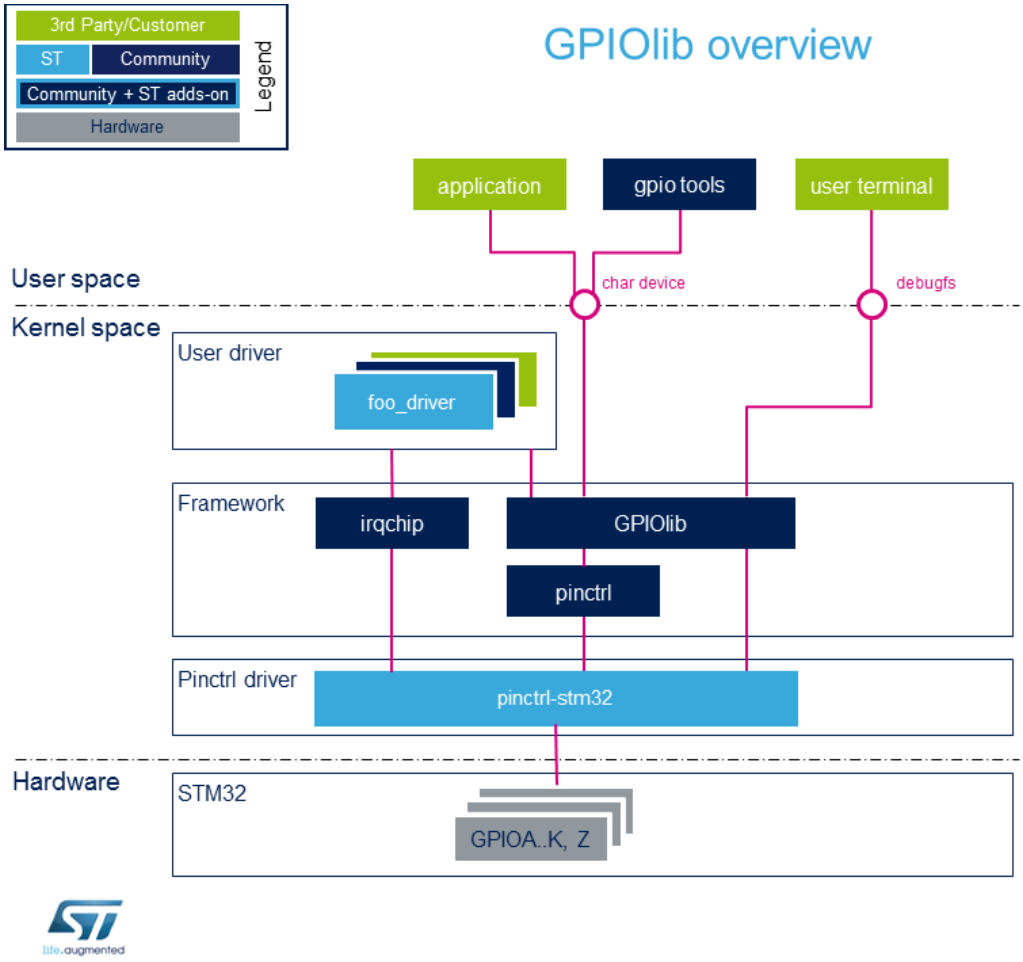
The purpose of this article is to explain how the GPIOlib framework manages IOs/pins, how to configure it, and how to use it.

Contents

1 Framework purpose

As explain in Overview of GPIO pins, the pins are multiplexed between "Alternate function", "ANALOG" and "GPIO" modes. The current article deals with the framework to use in order to manipulate GPIO: the GPIOLib framework. The GPIOLib framework is used when you want to control a pin directly in software, i.e. to set pin as output as a certain level (0/1), or to set pin in input configuration, or to configure the GPIO as an interrupt line. More details about GPIO in the kernel documentation are available in gpio.txt [1]

2 System overview



2.1 Component description

- **GPIOLib:**
  - provides the GPIOLib API to be used by a driver who wants to control a GPIO. See API for more details.
  - calls the pinctrl framework to manage/configure pins.
- **Pinctrl:** the pinctrl framework **core**. Its role is:
  - to call specific vendor callback for the pin configuration (muxing end setting).
  - to create the logical pin mapping and guarantee pin exclusivity for a device. See Pinctrl overview
- **Pinctrl-stm32:** microprocessor **specific** pinctrl driver, its role is:
  - to register the vendor specific function (callback) in the pinctrl framework.
  - to access to hardware registers to configure pins (muxing and all pin capabilities).
- **IRQ chip**: provides the API[2] to request an IRQ for a GPIO Line.
- **foo_driver:** foo_driver could be any driver that needs to control a GPIO.
- **debugfs:** provides a debug interface available from the user terminal. See How to monitor with debugfs.
- **GPIO tools:** tools offered by the kernel to manipulate GPIOs. See GPIO tools.

2.2 API description

Depending on needs and the caller location (kernel space or user space), several APIs are available to control a pin:

- **User space API:** the Linux® standard character device driver interface[3] allows applications to manipulate all "system objects" with the standard file API (open, read, write, close, IOctl...). This is used to manipulate GPIO through user space, as gpio devices are seen as files.
  - Refer to GNU documentation for file manipulation.
  - See example How to control a GPIO in userspace

- **Kernel space API**: the GPIOLib API provides API to the user driver.
  - See in the Kernel documentation "consumer.txt" [4] to know how to obtain and use GPIO in a driver.
  - See in the Kernel documentation "board.txt" [5] to know how to assign GPIOs to a consumer device.
  - The GPIO descriptor acquisition is linked to a function label declared in the device tree. So there is a direct relationship between the driver and the device tree about this label, and the driver name (see sample code "dummy.c" below (ie "st, dummy" and "redled")).
  - See example How to control a GPIO in kernel space

- **Debugfs API**: provides only information about GPIO configuration. See How to monitor with debugfs.

3 Configuration

3.1 Kernel configuration

GPIOLib is activated by default in ST deliveries.

3.2 Device tree configuration

A detailed device tree configuration is described in GPIO device tree configuration.

4 How to use framework

4.1 How to configure a new board

If you have to declare a GPIO, please refer first to the hardware block GPIO page: GPIO internal peripheral and your board scematics. Then either you declare your GPIO directly in device tree or you declare it using the STM32CubeMX.

4.2 How to toggle a GPIO in user space

How to control a GPIO in userspace

4.3 How to toggle a GPIO in kernel space

How_to_control_a_GPIO_in_kernel_space

4.4 Tools

Control GPIO through libgpiod

5 How to trace and debug the framework

5.1 How to monitor

5.1.1 How to monitor with debugfs

Some information about gpio is available in Debugfs interface.

```
Board $> cat /sys/kernel/debug/gpio
```

Output:

```
gpiochip0: GPIOs 0-15, parent: platform/soc:pin-controller, GPIOA:
...
gpiochip1: GPIOs 16-31, parent: platform/soc:pin-controller, GPIOB:
...
gpiochip5: GPIOs 80-95, parent: platform/soc:pin-controller, GPIOF:
 gpio-94  (                    |?                   ) out hi
 gpio-95  (                    |reset               ) out hi
...
```

It provides per gpiochip: requested GPIOs, the state of the GPIO and the gpio-controller parent.

5.1.2 Other ways to monitor

You can also get GPIO information by using gpiodetect - gpioinfo tools

5.2 How to trace

By default there is no kernel trace to see the activity on GPIOs. However you could enable dynamic debug for the gpiolib and the pinctrl.

```
Board $> dmesg -n8
Board $> echo "file drivers/pinctrl/* +p" > /sys/kernel/debug/dynamic_debug/control
Board $> echo "file drivers/gio/* +p" > /sys/kernel/debug/dynamic_debug/control
```

6 Source code location

The source files are located inside the Linux kernel.

- **GPIOLib**: gpiolib.c[6] and gpiolib-of.c[7]
- **GPIO tools**: folder tools/gpio in Kernel[8]
- **Pinctrl core part**: generic core[9], generic pinconf[10] and generic pinmux[11]
- **STM32 pinctrl vendor part**: folder to STM32 dedicated pinctrl functions[12]

7 To go further

GPIOLib is the main framework to handle a GPIO at run time. However, dedicated GPIO drivers already exist to control basic features (Leds, regulator, buttons, ...). You can find the list (and more details) inside the Kernel documentation [13]

8 References

- Kernel documentation - gpio.txt, GPIO Kernel documentation
- Interrupt Management training document, *Linux Kernel and Driver Development*
- character device interface, *Linux Kernel and Driver Development* training document, see **Kernel frameworks for device drivers** chapter
- Kernel gpio documentation - consumer.txt,to know how to obtain and use GPIO in a driver
- Kernel gpio documentation - board.txt, to know how to assign GPIOs to a consumer device and a function
- GPIOLib source - gpiolib.c Provides GPIOLib API
- GPIOLib source - gpiolib-of.c Provides OF-GPIOLib API
- gpio tools folder Provides all sources of GPIO tools
- Pinctrl framework source - core.c Sources of generic pinctrl framework
- Pinctrl framework source - pinconf.c Sources of generic pin configuration
- Pinctrl framework source - pinmux.c Sources of generic pin muxing
- STM32 vendor specific folder Provides all vendor specifics functions
- Kernel documentation - drivers-on-gpio.txt List and explain standard kernel drivers used for common GPIO task

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.)

Application programming interface

foo_driver could be any driver that needs to control a GPIO

Debug File System (See https://en.wikipedia.org/wiki/Debugfs for more details)