# Open BMC Development Series (VI) Adding support for GPIO

原创 dayuli Posted on 2021-11-29 18:52:25 Read 3.8k Collection 19 Likes 10



Category Column: BMC Article Tags: server linux bmc



摘要 This blog introduces how to operate GPIO in Linux system, including methods of accessing registers through sysfs interface, libgpiod library and directly. First, it describes how to check and configure GPIO driver and check the status of GPIO port, then compares the advantages and disadvantages of using GPIOsysfs interface and libgpiod library in detail, and gives examples of use. Finally, it mentions the method of directly operating GPIO registers through physical addresses, and the popular RPi.GPIO library. The article emphasizes the high efficiency and concurrency advantages of the new interface libgpiod.

The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>



The logic of adding BMC support for GPIO is the same as that of adding GPIO to embedded systems.

## 1. Define the device tree and install the GPIO driver

By default, Linux includes GPIO drivers. To check whether GPIO is installed, check whether /sys/class/gpio exists. If it exists, it means that the gpio driver has been installed. If not, you need to turn on the GPIO driver switch of the Linux kernel.

## 2 Check the usage of GPIO port

```
root@s2600wf:/sys/class/gpio# gpioinfo
gpiochip0 - 232 lines:
        line   0:      unnamed       unused   input  active-high
        line   1:      unnamed       unused   input  active-high
        line   2:      unnamed       unused   input  active-high
        line   3:      unnamed       unused   input  active-high
        line   4:      unnamed       unused   input  active-high
        line   5:      unnamed       unused   input  active-high
        line   6:      unnamed       kernel   input  active-high [used]
        line   7:      unnamed       kernel   input  active-high [used]
        line   8:      unnamed       unused   input  active-high
        line   9:      unnamed       unused   input  active-high
        line  10:      unnamed       unused   input  active-high
        line  11:      unnamed       unused   input  active-high
        line  12:      unnamed       unused   input  active-high
        line  13:      unnamed       unused   input  active-high
        line  14:      unnamed       unused   input  active-high
        line  15:      unnamed       unused   input  active-high
        line  16:      unnamed       kernel   input  active-high [used]
        line  17:      unnamed       kernel   input  active-high [used]
        line  18:      unnamed       unused   input  active-high
        line  19:      unnamed       unused   input  active-high
        line  20:      unnamed       unused   input  active-high
        line  21:      unnamed       unused   input  active-high
        line  22:      unnamed       unused   input  active-high
        line  23:      unnamed       unused   input  active-high
        line  24:      unnamed       unused   input  active-high
        line  25:      unnamed       unused   input  active-high
        line  26:      unnamed       unused   input  active-high
        line  27:      unnamed       unused   input  active-high
        line  28:      unnamed       unused   input  active-high
        line  29:      unnamed       unused   input  active-high
        line  30:      unnamed       unused   input  active-high
        line  31:      unnamed       unused   input  active-high
        line  32:      unnamed       unused   input  active-high
        line  33:      unnamed       unused   input  active-high
        line  34:      unnamed       unused   input  active-high
        line  35:      unnamed       unused   input  active-high
        line  36:      unnamed       unused   input  active-high
        line  37:      unnamed       unused   input  active-high
        line  38:      unnamed       unused   input  active-high
        line  39:      unnamed       unused   input  active-high
        line  40:      unnamed       unused   input  active-high
        line  41:      unnamed       unused   input  active-high
        line  42:      unnamed       unused   input  active-high
        line  43:      unnamed       unused   input  active-high
        line  44:      unnamed       unused   input  active-high
        line  45:      unnamed       unused   input  active-high
```



Sometimes, when we configure the IO port, it will be occupied, or after configuring the IO port, we need to use the command: gpioinfo

View the base address of GPIO:

bash



```
root@s2600wf:/sys/class/gpio# cat /sys/devices/platform/ahb/ahb:apb/1e780000.gpio/gpio/*/base
792
```

## 3. Three methods of reading and writing IO

### 3.1 Read and write IO using GPIO sysfs

In Linux, the most common way to read and write GPIO is to use the GPIO sysfs interface, which is implemented by operating files such `/sys/class/gpio`as `export`, `unexport`, `gpio{N}/direction`, `gpio{N}/value`(replace {N} with the actual pin number) in the directory, and often appears in shell scripts. For example, to control GPIO12 of Raspberry Pi 3B in the shell:



```
sudo su
cd /sys/class/gpio
echo 12 > export                  # IO的地址 base + offset
echo out > gpio12/direction       # io used for output
echo 1 > gpio12/value             # output logic 1 level
echo 0 > gpio12/value             # output logic 0 level
echo 12 > unexport
```

There are many libraries based on GPIO sysfs interface encapsulation. Here we recommend python-periphery , c-periphery and lua-periphery written by vsergeev . You can choose python, lua and c, which are very versatile.

The GPIO sysfs interface is currently widely used, but it has some problems, such as the inability to concurrently obtain sysfs attributes and is basically designed for shell interfaces. Therefore, gpiod has replaced it since Linux 4.8.

> Since linux 4.8 the GPIO sysfs interface is deprecated. User space should use
> the character device instead. This library encapsulates the ioctl calls and
> data structures behind a straightforward API.

### 3.2 Reading and Writing IO with libgpiod

The new design gpiod, GPIO access control is `/dev/gpiodchip0`implemented by operating character device files (such as), and provides some command tools, C libraries and Python encapsulation through libgpiod.

> The new character device interface guarantees all allocated resources are
> freed after closing the device file descriptor and adds several new features
> that are not present in the obsolete sysfs interface (like event polling,
> setting/reading multiple values at once or open-source and open-drain GPIOs).
>
> Unfortunately interacting with the linux device file can no longer be done
> using only standard command-line tools. This is the reason for creating a
> library encapsulating the cumbersome, ioctl-based kernel-userspace interaction
> in a set of convenient functions and opaque data structures.
>
> Additionally this project contains a set of command-line tools that should
> allow an easy conversion of user scripts to using the character device.

`gpioset`Through the , `gpioget`, and provided by libgpiod , `gpiomon`you can quickly read and write GPIO and detect input events.



```
sudo apt install -y gpiod
sudo gpioset 0 12=0
sudo gpiomon 0 12   # detect event on gpio12
```

Since gpiod is relatively new, few people use it. Although there is a python package in libgpiod, it has not been packaged into the debian stretch repository, so I used python ctypes to package it in voice-engine/gpio-next . The python code to control an LED is as follows:



```
import time
from gpio_next import GPIO

LED = GPIO(12, direction=1)
for i in range(10):
    LED.write(i & 1)
    time.sleep(1)
```

Using sysfs or gpiod is the interface of GPIO encapsulation by Linux. After encapsulation, the versatility is better, but the performance will be weaker. If you have learned 51 and ARM microcontrollers, the most familiar way to read and write GPIO is probably to directly operate the GPIO register. This is the fastest way to control the IO port, and you can also directly operate the GPIO register in Linux.

### 3.3. Register direct read and write IO

Compared with single-chip microcomputers, Raspberry Pi uses ARM SoC BCM2837 with MMU. The address space is slightly more complicated, with different address space mappings. From the BCM2837 data sheet , the GPIO register bus address starts at 0x7E200000 (see the figure below), and the mapped physical address is 0x3F200000 (see page 6 of the manual).

To directly read and write physical addresses under Linux, you must first open the device file `/dev/mem`, then `mmap`map the file, and finally read and write the corresponding register according to the register address.

It can be `devmem2`used to read and write physical memory addresses. If not `apt install devmem2`, you can download and compile a copy at https://github.com/VCTLabs/devmem2 . For example, read and write the IO status of the Raspberry Pi:



```
sudo devmem2 0x3F200034 w
```

| Address | Field Name | Description | Size | Read/ Write |
|---|---|---|---|---|
| 0x 7E20 0000 | GPFSEL0 | GPIO Function Select 0 | 32 | R/W |
| 0x 7E20 0000 | GPFSEL0 | GPIO Function Select 0 | 32 | R/W |
| 0x 7E20 0004 | GPFSEL1 | GPIO Function Select 1 | 32 | R/W |
| 0x 7E20 0008 | GPFSEL2 | GPIO Function Select 2 | 32 | R/W |
| 0x 7E20 000C | GPFSEL3 | GPIO Function Select 3 | 32 | R/W |
| 0x 7E20 0010 | GPFSEL4 | GPIO Function Select 4 | 32 | R/W |
| 0x 7E20 0014 | GPFSEL5 | GPIO Function Select 5 | 32 | R/W |
| 0x 7E20 0018 | - | Reserved | - | - |
| 0x 7E20 001C | GPSET0 | GPIO Pin Output Set 0 | 32 | W |
| 0x 7E20 0020 | GPSET1 | GPIO Pin Output Set 1 | 32 | W |
| 0x 7E20 0024 | - | Reserved | - | - |
| 0x 7E20 0028 | GPCLR0 | GPIO Pin Output Clear 0 | 32 | W |
| 0x 7E20 002C | GPCLR1 | GPIO Pin Output Clear 1 | 32 | W |
| 0x 7E20 0030 | - | Reserved | - | - |
| 0x 7E20 0034 | GPLEV0 | GPIO Pin Level 0 | 32 | R |
| 0x 7E20 0038 | GPLEV1 | GPIO Pin Level 1 | 32 | R |

> Physical addresses range from 0x3F000000 to 0x3FFFFFFF for peripherals.

The Python library RPi.GPIO controls GPIO by directly accessing registers. Since the GPIO register addresses of different chips are mostly different, there is no universality.

***Finally: Liking is a virtue, following is fate, collecting is affirmation, you can reward me as you like, your encouragement is part of the goodness in my world, I love you!***