iSite BTS3001C-116V100R001

Pinctrl

Issue **01**

Date **2018-03-16**

Trademarks and Permissions

 , HISILICON, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided “AS IS” without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://www.hisilicon.com/cn/

Email: support@hisilicon.com

About This Document

Purpose

This document describes the networking and protection of SDH, PDH, Ethernet, ATM, SAN and video services. In addition, network management information, orderwire and clock planning is described briefly.

This document provides guides to get the information about how to construct a network.

Intended Audience

This document is intended for:

- Policy planning engineers
- Installation and commissioning engineers
- NM configuration engineers
- Technical support engineers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
DANGER	Indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury.
WARNING	Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury.
CAUTION	Indicates a potentially hazardous situation which, if not avoided, may result in minor or moderate injury.
NOTICE	Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results. NOTICE is used to address practices not related to personal injury.

Symbol	Description
NOTE	Calls attention to important information, best practices and tips. NOTE is used to address information not related to personal injury, equipment damage, and environment deterioration.

Change History

Changes between document issues are cumulative. The latest document issue contains all the changes made in earlier issues.

Issue 01 (2018-03-16)

This issue is used for first office application (FOA).

Contents

Figures

Tables

1 Description

1.1 Pinctrl

1.1.1 General description

In the hardware aspect, the functions and characteristics of configuring one or a group of pins can be realized by the registers of the pinctrl controller in the chip 970.

In terms of software, the kernel pinctrl module is based on Linux 4.9, which implements the unified interface for pin function multiplexing, drive capability configuration, pull-up operation, and pull-down operation, including some interfaces that can operate one pin, and some interfaces that can operate a set of pins at the same time.

1.1.2 API

devm_pinctrl_get	Resource managed pinctrl handle, retrieves the pinctrl handle for a device
devm_pinctrl_put	Resource managed pinctrl handle, decrease use count on a previously claimed pinctrl handle
pinctrl_lookup_state	retrieves a state handle from a pinctrl handle
pinctrl_select_state	select/activate/program a pinctrl state to HW
devm_pinctrl_get_select	get the pinctrl handle, and set state
devm_pinctrl_get_select_default	get the pinctrl handle, and set default state

1.2 Function

1.2.1 devm_pinctrl_get

prototype

struct pinctrl *devm_pinctrl_get(struct device *dev);

description

pinctrl_get() - retrieves the pinctrl handle for a device
devm_pinctrl_get() - Resource managed pinctrl_get()

parameter

dev: the device to obtain the handle for

return

struct pinctrl or null, and error code

1.2.2 devm_pinctrl_put

prototype

void devm_pinctrl_put(struct pinctrl *p);

description

pinctrl_put() - decrease use count on a previously claimed pinctrl handle
devm_pinctrl_put() - Resource managed pinctrl_put()

parameter

p: the pinctrl handle to release

return

none

1.2.3 pinctrl_lookup_state

prototype

struct pinctrl_state *pinctrl_lookup_state(struct pinctrl *p, const char *name);

description

retrieves a state handle from a pinctrl handle

parameter

p: the pinctrl handle to retrieve the state from
name: the state name to retrieve

return

struct pinctrl or null, and error code

1.2.4 pinctrl_select_state

prototype

int pinctrl_select_state(struct pinctrl *p, struct pinctrl_state *state);

description

select/activate/program a pinctrl state to HW

parameter

p: the pinctrl handle for the device that requests configuration
state: the state handle to select/activate/program

return

0 success, or error

1.2.5 devm_pinctrl_get_select

prototype

struct pinctrl * devm_pinctrl_get_select(struct device *dev, const char *name);

description

Called devm_pinctrl_get_select() and set default state

parameter

dev: the device to obtain the handle for
name: the state name to retrieve

return

struct pinctrl or null, and error code

1.2.6 devm_pinctrl_get_select_default

prototype

struct pinctrl * devm_pinctrl_get_select_default(struct device *dev);

description

Called the following api:
devm_pinctrl_get()
pinctrl_lookup_state()
pinctrl_select_state()

parameter

dev: the device to obtain the handle for

return

struct pinctrl or null, and error code

1.3 Reference

1.3.1 DTS Configuration

The steps for configuring pinctrl in DTS are as follows:

① If the developed device mydevice uses uart0, the transceiver takes two GPIO pins as GPIO_035 and GPIO_036, and the pin state is default. In the device tree file, you can configure the device node as follows:

```
mydevice {

    pinctrl-names = "default";

    pinctrl-0 = <&uart0_pmx_func &uart0_cfg_func>;

};
```

Explanation:

• pinctrl-names is pinctrl's state name default;

• uart0_pmx_func is the pin multiplexing register setting for the status default.

• uart0_cfg_func is the pin configuration register setting for the status default.

• If there is more than one state, it can be written as

```
pinctrl-names = "default","idle";
pinctrl-0 = <&uart0_pmx_func &uart0_cfg_func>;
pinctrl-1 = <&uart0_pmx_idle &uart0_cfg_idle>;
```

The default configuration is pinctrl-0, and the idle configuration is pinctrl-1.

② The following is a register description of the default state. The caller only needs to look up their node names and then refer nodes as their own node information without any changes to the gates.

In the DTS file hikey970-pinctrl.dtsi, the uart0 pin multiplexing register is set as follows:

```
pmx0: pinmux@e896c000 {
        compatible = "pinctrl-single";
        reg = <0x0 0xe896c000 0x0 0x72c>;
        #pinctrl-cells = <1>;
        #gpio-range-cells = <0x3>;
        pinctrl-single,register-width = <0x20>;
        pinctrl-single,function-mask = <0x7>;
        /* pin base, nr pins & gpio function */
        pinctrl-single,gpio-range = <&range 0 82 0>;

        uart0_pmx_func: uart0_pmx_func {
                pinctrl-single,pins = <
                        0x054 MUX_M2 /* UART0_RXD */
                        0x058 MUX_M2 /* UART0_TXD */
                >;
        };
};
```

Explanation:

• MUX_M2 is the pin multiplexing register value, where the macro is used to select pin function 2, and the specific macro is defined in the file include/dt-bindings/pinctrl/hisi.h.

• 0x54 and 0x58 are the offset addresses relative to the pin multiplex register base address 0xe896c000, ie, the multiplexed register addresses for GPIO_035 and GPIO_036.

③ Hikey970-pinctrl.dtsi in the DTS file with the uart0 pin configuration register is set as follows:

```
pmx2: pinmux@e896c800 {
        compatible = "pinconf-single";
        reg = <0x0 0xe896c800 0x0 0x72c>;
        #pinctrl-cells = <1>;
        pinctrl-single,register-width = <0x20>;

        uart0_cfg_func: uart0_cfg_func {
                pinctrl-single,pins = <
                        0x058 0x0 /* UART0_RXD */
                        0x05c 0x0 /* UART0_TXD */
                >;
                pinctrl-single,bias-pulldown = <
                        PULL_DIS
                        PULL_DOWN
                        PULL_DIS
                        PULL_DOWN
                >;
                pinctrl-single,bias-pullup = <
                        PULL_DIS
                        PULL_UP
                        PULL_DIS
                        PULL_UP
                >;
                pinctrl-single,drive-strength = <
                        DRIVE7_04MA DRIVE6_MASK
                >;
        };
};
```

Explanation:
• PULL_UP, PULL_DOWN are pin configuration register values, where macros are used to indicate whether the pin is set to pull-up or pull-down. The specific macros are defined in the file include/dt-bindings/pinctrl/hisi.h.

• DRIVE7_04MA indicates that the configuration register drive strength is set to 4mA, refer to include/dt-bindings/pinctrl/hisi.h.
• 0x58 and 0x5c are the offset addresses relative to the pin configuration register base address 0xe896c800, ie, the configuration register addresses for GPIO_035 and GPIO_036.

1.3.2 Driver

The pinctrl operation steps in the device driver are as follows

① Write your own device driver file

```
mydevice.c
```

② Include header files

```
#include <linux/pinctrl/consumer.h>
```

③ Get pinctrl handle

```
struct pinctrl *mydevice_pinctrl;

mydevice_pinctrl = devm_pinctrl_get(mydevice->dev);

    if (IS_ERR(mydevice_pinctrl))

        dev_err(&mydevice->dev, "could not get pinctrl\n");
```

④ Get and set pinctrl to default

```
int status;

struct pinctrl_state *pinctrl_default;

pinctrl_default = pinctrl_lookup_state(mydevice_pinctrl,
PINCTRL_STATE_DEFAULT);

if (!IS_ERR(pinctrl_default)) {

   status = pinctrl_select_state(mydevice_pinctrl, pinctrl_default);

   if (status)

       dev_err(mydevice->dev, "could not set default pins\n");

} else

   dev_err(mydevice->dev, "could not get default pinstate\n");
```

Explanation:
• The above steps are only for reference.
• Steps 3 and 4 can also be implemented using the interface devm_pinctrl_get_select_default.
• The device driver requests the pin resource. The function that starts with devm_xxx() is used first, which can automatically manage the application of the pin resource.