EC2x&AG35-Quecopen GPIO Configuration Method

LTE Standard/ Automotive Module Series

Rev. EC2x&AG35-Quecopen_GPIO_Configuration_Method_V1.1

Date: 2018-09-29

Status: Preliminary

Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.
7th Floor, Hongye Building, No.1801 Hongmei Road, Xuhui District, Shanghai 200233, China
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local office. For more information, please visit:
http://www.quectel.com/support/sales.htm

For technical support, or to report documentation errors, please visit:
http://www.quectel.com/support/technical.htm
Or email to: support@quectel.com

GENERAL NOTES

COPYRIGHT

About the Document

History

Revision	Date	Author	Description
1.0	2018-01-10	Gale GAO	Initial
1.1	2018-09-29	Gale GAO	Added GPIO/EINT API of User Space

Contents

About the Document **2**
Contents **3**

1 Introduction **4**

2 Bootloader Configuring GPIO **5**

3 Kernel Space Configuring GPIO **6**
3.1. Using Kernel Interface 6
3.2. Using pinctrl Subsystem 6

4 User Space Configuring GPIO **9**
4.1. Error Types of Return Values 9
4.2. GPIO 9
4.3. EINT 11

1 Introduction

This document mainly applies for global market. Currently LTE Standard/Automotive module that supports this includes:

- EC2x: EC20 R2.1/EC25/EC21
- AG35

2 Bootloader Configuring GPIO

Please refer to header file: *ql-ol-sdk/ql-ol-bootloader/platform/mdm9607/include/platform/gpio.h* for related definitions.

void gpio_tlmm_config(uint32_t gpio, uint8_t func, uint8_t dir,
uint8_t pull, uint8_t drvstr, uint32_t enable);
Function features: configure the pin functions, directions, pulling up/down, drive strength and whether to enable the functions.
Parameter:

gpio: chip GPIO number
func: 0 means GPIO
dir: direction GPIO_INPUT or GPIO_OUTPUT
pull: pull up/down. GPIO_NO_PULL, GPIO_PULL_DOWN, GPIO_KEEPER, GPIO_PULL_UP
enable: GPIO_ENABLE, GPIO_DISABLE

void gpio_set_val(uint32_t gpio, uint32_t val);
Function features: configure GPIO level
Parameter:

gpio: chip GPIO number
val: 0, 1

uint32_t gpio_get_state(uint32_t gpio);
Function features: obtain GPIO level
Parameter:
gpio: chip GPIO number
Return value: 0, 1

3 Kernel Space Configuring GPIO

3.1. Using Kernel Interface

Apply GPIO
*int gpio_request(unsigned gpio, const char *label);*

Mark GPIO direction including input and output
int gpio_direction_input(unsigned gpio);
int gpio_direction_output(unsigned gpio, int value);

Obtain and set the pin level of GPIO (For output)
int gpio_get_value(unsigned gpio);
void gpio_set_value(unsigned gpio, int value);

Convert GPIO to corresponding IRQ value. It will return the interrupt number.
int gpio_to_irq(unsigned gpio);

Configure the interruption
request_irq(unsigned int irq, irq_handler_t handler,
unsigned long flags, const char *name, void *dev);

Log out the interruption
*void free_irq(unsigned int irq, void *dev_id);*

Release GPIO
void gpio_free(unsigned gpio);

3.2. Using pinctrl Subsystem

Configure the device tree:

```
ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607-pinctrl.dtsi
                gpio_default:   gpio_default{
                    mux {
                        pins = "gpioX", "gpioY";      //Pins needed to be configured
```

```
                function = "gpio";          // Configure as GPIO function
            };
            config {
                pins = "gpioX", "gpioY";
                drive-strength = <2>;       //  Drive strength
                bias-disable;               //  Available values to pull up and down: bias-disable;
                                                bias-pull-down; bias-pull-up
                output-low;                 //  Available values for output level: output-low;
                                                    output-high
            };
        };

ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607.dtsi
        xxx{
            compatible = "xxx";             /* Match user driver */
            pinctrl-names = "default";      /* Define pinctrl name, using pinctrl_lookup_state()
                                               interface to parse in the driver */
            pinctrl-0 = <&gpio_xxx>;        /* Select the GPIO configuration defined above */
            status = "ok";                  /* Enable this device node */
        };
```

After the device tree is configured, write the driver to parse and enable the configuration above. Related Kernel API is as follows:

A. Obtain a handle of pinctrl. The parameter dev contains the device structure of the pin.

```
/**
 * struct devm_pinctrl_get() - Resource managed pinctrl_get()
 * @dev: the device to obtain the handle for
 *
 * If there is a need to explicitly destroy the returned struct pinctrl,
 * devm_pinctrl_put() should be used, rather than plain pinctrl_put().
 */
struct pinctrl *devm_pinctrl_get(struct device *dev)
```

B. Obtain the corresponding pin_state (pin state) of this pin.

```
/**
 * pinctrl_lookup_state() - retrieves a state handle from a pinctrl handle
 * @p: the pinctrl handle to retrieve the state from
 * @name: the state name to retrieve
 */
struct pinctrl_state *pinctrl_lookup_state(struct pinctrl *p, const char *name)
```

C. Set the pin to a certain state.

```
/**
```

```
        * pinctrl_select_state() - select/activate/program a pinctrl state to HW
        * @p: the pinctrl handle for the device that requests configuration
        * @state: the state handle to select/activate/program
        */
int pinctrl_select_state(struct pinctrl *p, struct pinctrl_state *state)
```

4 User Space Configuring GPIO

The user layer controls GPIO via QuecOpen API.

4.1. Error Types of Return Values

```
enum {
        RES_OK = 0,
        RES_BAD_PARAMETER    = -1,
        RES_IO_NOT_SUPPORT   = -2,
        RES_IO_ERROR         = -3,
        RES_NOT_IMPLEMENTED = -4
    };
```

4.2. GPIO

Enumerate:

```
typedef   enum {
        PINDIRECTION_IN   = 0,
        PINDIRECTION_OUT = 1
    }Enum_PinDirection;

    typedef   enum{
        PINLEVEL_LOW   = 0,
        PINLEVEL_HIGH   = 1
    }Enum_PinLevel;

    typedef   enum{
        PINPULLSEL_DISABLE      = 0,
        PINPULLSEL_PULLDOWN = 1,
        PINPULLSEL_PULLUP      = 3
    }Enum_PinPullSel;
```

int QI_GPIO_Base_Init(Enum_PinName pinName);

Function features: initialize GPIO. No other settings are conducted except initialize GPIO. Select one

between GPIO and Ql_GPIO_Init.
Parameter: pinName: see enumeration values of ql_gpio.h Enum_PinName for the pin number.
Return value: See ***Chapter 4.1*** for error types.

```
int Ql_GPIO_Init( Enum_PinName    pinName,
                  Enum_PinDirection    dir,
                  Enum_PinLevel         level,
                  Enum_PinPullSel       pullSel
                );
```

Function features:initialize the specified pins. The function will configure the directions, level and pulling up/down; Return the error type.
Parameter:

pinName: pin name. See enumeration values of ql_gpio.h Enum_PinName;
dir: directions. Enumerate Enum_PinDirection;
level: level. Enumerate Enum_PinLevel;
pullSel: pull up and down. Enumerate Enum_PinPullSel;

Return value: see ***Chapter 4.1*** for error types.

int Ql_GPIO_SetDirection(Enum_PinName pinName, Enum_PinDirection dir);
Function features: the direction configuration of specified pin; Return the error type.
Parameter:

pinName: pin name. See enumeration values of ql_gpio.h Enum_PinName;
dir: directions. Enumerate Enum_PinDirection;

Return value: see ***Chapter 4.1*** for error types.

int Ql_GPIO_GetDirection(Enum_PinName pinName);
Parameter: pinName: pin name. See enumeration values of ql_gpio.h Enum_PinName;
Return value: return the specified pin direction.

int Ql_GPIO_SetLevel(Enum_PinName pinName, Enum_PinLevel level);
When the direction of GPIO is out, the output level from this interface can be directly called; Return the error type.
Parameter:

pinName: pin name. See enumeration values of ql_gpio.h Enum_PinName;
level: pin level. Enumerate Enum_PinLevel;

Return value: see ***Chapter 4.1*** for error types.

int Ql_GPIO_GetLevel(Enum_PinName pinName);
Parameter:

pinName: pin name. See enumeration values of ql_gpio.h Enum_PinName;
pullSel: pull up and down. Enumerate Enum_PinPullSel;

Return value: return the specified pin level.

int Ql_GPIO_SetPullSelection(Enum_PinName pinName, Enum_PinPullSel pullSel);
Function features: pin pulling up/down configuration of specified pins; Return the error type.
Parameter:
pinName: pin name. See enumeration values of ql_gpio.h Enum_PinName;
pullSel: pull up and down. Enumerate Enum_PinPullSel;
Return value: see ***Chapter 4.1*** for error types.

int Ql_GPIO_GetPullSelection(Enum_PinName pinName);
Parameter: pinName: pin name. See enumeration values of ql_gpio.h Enum_PinName;
Return value: return the state of pulling up/down of the specified pin.

int Ql_GPIO_Uninit(Enum_PinName pinName);
Release GPIO configuration of specified pin; Return the error type.
Parameter: pinName: pin name. See enumeration values of ql_gpio.h Enum_PinName;
Return value: see ***Chapter 4.1*** for error types.

Reference: *ql-ol-extsdk/example/gpio*

4.3. EINT

Enumerate:
```
typedef enum {
    EINT_SENSE_NONE,
    EINT_SENSE_RISING,
    EINT_SENSE_FALLING,
    EINT_SENSE_BOTH
}Enum_EintType;

int Ql_EINT_Enable(   Enum_PinName    eintPinName,
Enum_EintType    eintType,
                      Ql_EINT_Callback      eint_callback
);
```
Enable pin interruption and register user callback function; Return the error type.
Parameter:
eintPinName: pin name. See enumeration values of ql_gpio.h Enum_PinName;
eintType: edge triggered types. Enumerate Enum_EintType'
eint_callback: user callback function; Interrupt to trigger callback;
Return value: see ***Chapter 4.1*** for error types.

int Ql_EINT_Disable(Enum_PinName eintPinName);
Log out the function of pin interruption; Return the error type.
Parameter: eintPinName: pin name. See enumeration values of ql_gpio.h Enum_PinName;

Return value: see ***Chapter 4.1*** for error types.

Reference: *ql-ol-extsdk/example/eint*