HiKey970

UART Development Guide

Issue	**01**
Date	**2018-03-11**

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

Email: support@hisilicon.com

Change History

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

Issue 01 (2018-03-11)

The first version.

Contents

1 Description

1.1 UART

1.1.1 General description

The UART is an Advanced Microcontroller Bus Architecture (AMBA) compliant System-on-Chip (SoC) peripheral that is developed, tested, and licensed by ARM.

The UART is an AMBA slave module that connects to the Advanced Peripheral Bus(APB).

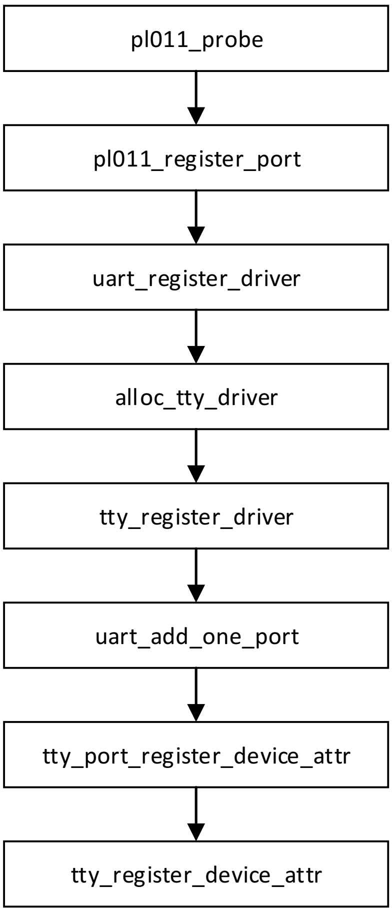
1.1.2 Features

The UART has the following features:

- Compliance to the *AMBA Specification (Rev 2.0)* onwards for easy integration into SoC implementation
- Programmable use of UART or IrDA SIR input/output
- Programmable FIFO disabling for 1-byte depth.
- Standard asynchronous communication bits (start, stop and parity). These are added prior to transmission and removed on reception
- Independent masking of transmit FIFO, receive FIFO, receive timeout, modem status, and error condition interrupts.
- Support for *Direct Memory Access* (DMA).
- False start bit detection.
- Line break generation and detection.
- Support of the modem control functions CTS, DCD, DSR, RTS, DTR, and RI.
- Programmable hardware flow control
- Fully-programmable serial interface characteristics
- Identification registers that uniquely identify the UART. These can be used by an operating system to automatically configure itself

1.2 UART Workflow

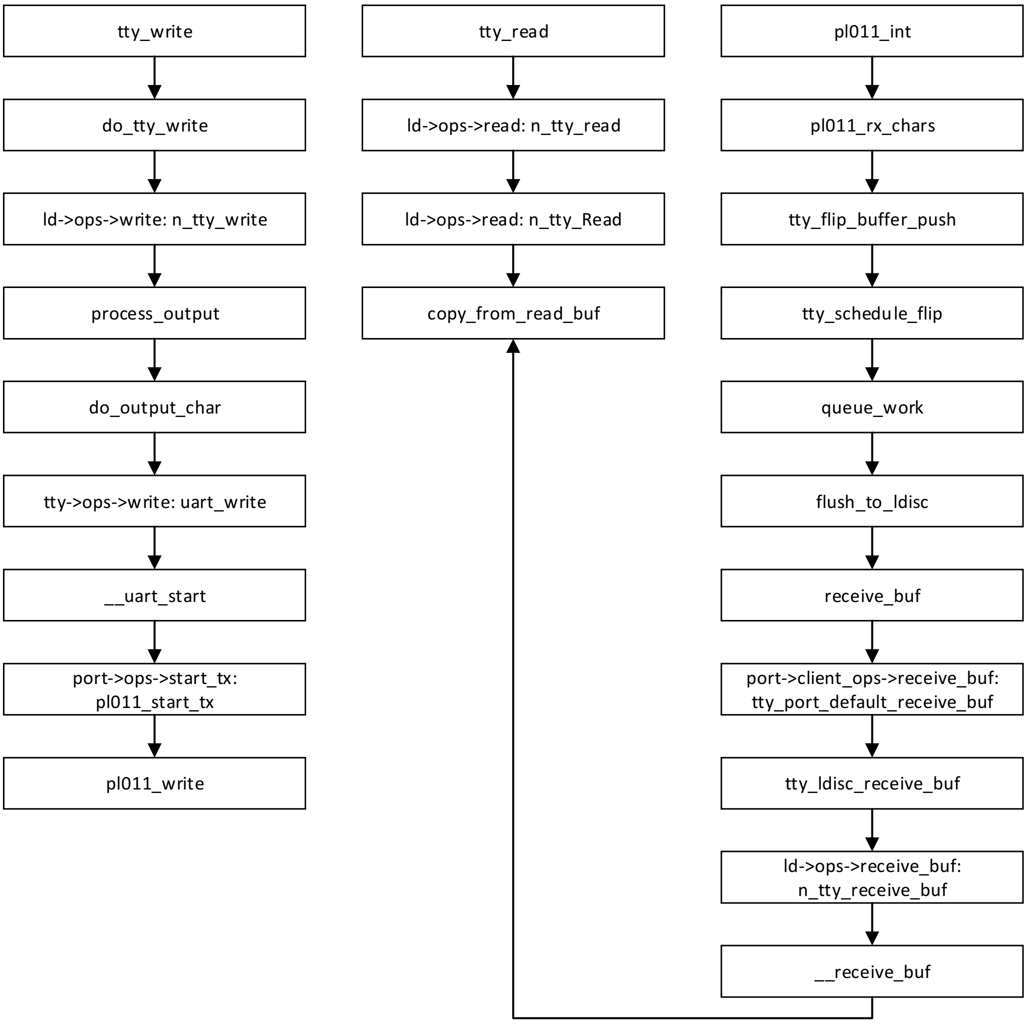
1.2.1 UART Initialization



The UART driver initialization process is as follows:
a. Parse the DTS file UART controller node to obtain relevant hardware information;
b. Initialize the UART controller to match the compatible attribute in DTS;
c. Registered UART serial port driver;
d. Apply for the tty_driver structure variable and register the tty driver;
e. Add the uart port;
f. Register the tty device.

1.2.2 Data Transfer

The data transceiver process is as follows


tty_write
do_tty_write
ld->ops->write: n_tty_write
process_output
do_output_char
tty->ops->write: uart_write
__uart_start
port->ops->start_tx: pl011_start_tx
pl011_write
tty_read
ld->ops->read: n_tty_read
ld->ops->read: n_tty_Read
copy_from_read_buf
pl011_int
pl011_rx_chars
tty_flip_buffer_push
tty_schedule_flip
queue_work
flush_to_ldisc
receive_buf
port->client_ops->receive_buf: tty_port_default_receive_buf
tty_ldisc_receive_buf
ld->ops->receive_buf: n_tty_receive_buf
__receive_buf

1.3 Development

1.3.1 DTS Configuration

DTS configuration file `kirin970.dtsi`

```
uart0: serial@fdf02000 {
       compatible = "arm,pl011", "arm,primecell";
       reg = <0x0 0xfdf02000 0x0 0x1000>;
       interrupts = <0 74 4>;
       clocks = <&crg_ctrl KIRIN970_CLK_GATE_UART0>,
              <&crg_ctrl KIRIN970_CLK_GATE_UART0>;
       clock-names = "uartclk", "apb_pclk";
       pinctrl-names = "default";
       pinctrl-0 = <&uart0_pmx_func &uart0_cfg_func>;
       status = "disabled";
};
```

This UART controller configuration includes base address, interrupt number, clock configuration, and UART controller switches. For the UART device configuration can be placed in the file `kirin970-hikey970.dts`. Examples are as follows

```
&uart0 {
    status = "ok";
    myuartdev {
        compatible = "myuartdev";
        max-speed = <921600>;
    };
};
```

1.3.2 Device Driver Configuration

Modify the file arch/arm64/configs/hikey970_defconfig, add

```
CONFIG_UART_MYUARTDEV=y
```

Modify the file drivers/tty/serial/Makefile and add

```
obj-$(CONFIG_UART_MYUARTDEV) += myuartdev.o
```

1.3.3 Data Structure

1.3.3.1 UART port

```
struct uart_port {
        spinlock_t            lock;                 /* port lock */
        unsigned long          iobase;               /* in/out[bwl] */
        unsigned char __iomem   *membase;             /* read/write[bwl] */
        unsigned int           (*serial_in)(struct uart_port *, int);
        void                  (*serial_out)(struct uart_port *, int, int);
        void                  (*set_termios)(struct uart_port *,
                                     struct ktermios *new,
                                     struct ktermios *old);
        unsigned int           (*get_mctrl)(struct uart_port *);
        void                  (*set_mctrl)(struct uart_port *, unsigned int);
        int                   (*startup)(struct uart_port *port);
        void                  (*shutdown)(struct uart_port *port);
        void                  (*throttle)(struct uart_port *port);
        void                  (*unthrottle)(struct uart_port *port);
        int                   (*handle_irq)(struct uart_port *);
        void                  (*pm)(struct uart_port *, unsigned int state,
                                 unsigned int old);
        void                  (*handle_break)(struct uart_port *);
        int                     (*rs485_config)(struct uart_port *,
                                                struct serial_rs485 *rs485);
        unsigned int            irq;                    /* irq number */
        unsigned long           irqflags;               /* irq flags  */
        unsigned int            uartclk;                /* base uart clock */
        unsigned int            fifosize;               /* tx fifo size */
        unsigned char           x_char;                 /* xon/xoff char */
        unsigned char           regshift;               /* reg offset shift */
        unsigned char           iotype;                 /* io access style */
        unsigned char           unused1;

#define UPIO_PORT               (SERIAL_IO_PORT)        /* 8b I/O port access */
#define UPIO_HUB6               (SERIAL_IO_HUB6)        /* Hub6 ISA card */
#define UPIO_MEM                (SERIAL_IO_MEM)         /* driver-specific */
#define UPIO_MEM32              (SERIAL_IO_MEM32)       /* 32b little endian */
#define UPIO_AU                 (SERIAL_IO_AU)          /* Au1x00 and RT288x
type IO */
#define UPIO_TSI                (SERIAL_IO_TSI)         /* Tsi108/109 type IO */
#define UPIO_MEM32BE            (SERIAL_IO_MEM32BE)     /* 32b big endian */
#define UPIO_MEM16              (SERIAL_IO_MEM16)       /* 16b little endian */

        unsigned int            read_status_mask;       /* driver specific */
        unsigned int            ignore_status_mask;     /* driver specific */
        struct uart_state       *state;                 /* pointer to parent state */
        struct uart_icount      icount;                 /* statistics */

        struct console          *cons;                  /* struct console, if any */
#if defined(CONFIG_SERIAL_CORE_CONSOLE) || defined(SUPPORT_SYSRQ)
        unsigned long           sysrq;                  /* sysrq timeout */
#endif

        /* flags must be updated while holding port mutex */
        upf_t                   flags;

        /*
         * These flags must be equivalent to the flags defined in
         * include/uapi/linux/tty_flags.h which are the userspace definitions
         * assigned from the serial_struct flags in uart_set_info()
         * [for bit definitions in the UPF_CHANGE_MASK]
         *
         * Bits [0..UPF_LAST_USER] are userspace defined/visible/changeable
         * except bit 15 (UPF_NO_TXEN_TEST) which is masked off.
         * The remaining bits are serial-core specific and not modifiable by
         * userspace.
        */
#define UPF_FOURPORT          ((__force upf_t) ASYNC_FOURPORT     /* 1  */ )
#define UPF_SAK               ((__force upf_t) ASYNC_SAK          /* 2  */ )
#define UPF_SPD_HI            ((__force upf_t) ASYNC_SPD_HI       /* 4  */ )
#define UPF_SPD_VHI           ((__force upf_t) ASYNC_SPD_VHI      /* 5  */ )
#define UPF_SPD_CUST          ((__force upf_t) ASYNC_SPD_CUST  /* 0x0030 */ )
#define UPF_SPD_WARP          ((__force upf_t) ASYNC_SPD_WARP  /* 0x1010 */ )
#define UPF_SPD_MASK          ((__force upf_t) ASYNC_SPD_MASK  /* 0x1030 */ )
#define UPF_SKIP_TEST         ((__force upf_t) ASYNC_SKIP_TEST    /* 6  */ )
#define UPF_AUTO_IRQ          ((__force upf_t) ASYNC_AUTO_IRQ     /* 7  */ )
#define UPF_HARDPPS_CD        ((__force upf_t) ASYNC_HARDPPS_CD   /* 11 */ )
#define UPF_SPD_SHI           ((__force upf_t) ASYNC_SPD_SHI      /* 12 */ )
#define UPF_LOW_LATENCY       ((__force upf_t) ASYNC_LOW_LATENCY  /* 13 */ )
#define UPF_BUGGY_UART        ((__force upf_t) ASYNC_BUGGY_UART   /* 14 */ )
#define UPF_NO_TXEN_TEST      ((__force upf_t) (1 << 15))
#define UPF_MAGIC_MULTIPLIER  ((__force upf_t) ASYNC_MAGIC_MULTIPLIER /* 16
*/ )

/* Port has hardware-assisted h/w flow control */
#define UPF_AUTO_CTS          ((__force upf_t) (1 << 20))
#define UPF_AUTO_RTS          ((__force upf_t) (1 << 21))
#define UPF_HARD_FLOW         ((__force upf_t) (UPF_AUTO_CTS |
UPF_AUTO_RTS))
/* Port has hardware-assisted s/w flow control */
#define UPF_SOFT_FLOW         ((__force upf_t) (1 << 22))
#define UPF_CONS_FLOW         ((__force upf_t) (1 << 23))
#define UPF_SHARE_IRQ         ((__force upf_t) (1 << 24))
#define UPF_EXAR_EFR          ((__force upf_t) (1 << 25))
#define UPF_BUG_THRE          ((__force upf_t) (1 << 26))
/* The exact UART type is known and should not be probed.  */
#define UPF_FIXED_TYPE        ((__force upf_t) (1 << 27))
#define UPF_BOOT_AUTOCONF     ((__force upf_t) (1 << 28))
#define UPF_FIXED_PORT        ((__force upf_t) (1 << 29))
#define UPF_DEAD              ((__force upf_t) (1 << 30))
#define UPF_IOREMAP           ((__force upf_t) (1 << 31))

#define __UPF_CHANGE_MASK     0x17fff
#define UPF_CHANGE_MASK       ((__force upf_t) __UPF_CHANGE_MASK)
#define UPF_USR_MASK          ((__force upf_t)
(UPF_SPD_MASK|UPF_LOW_LATENCY))

#if __UPF_CHANGE_MASK > ASYNC_FLAGS
#error Change mask not equivalent to userspace-visible bit defines
#endif
```

```
        /*
         * Must hold termios_rwsem, port mutex and port lock to change;
         * can hold any one lock to read.
         */
        upstat_t            status;

#define UPSTAT_CTS_ENABLE       ((__force upstat_t) (1 << 0))
#define UPSTAT_DCD_ENABLE       ((__force upstat_t) (1 << 1))
#define UPSTAT_AUTORTS          ((__force upstat_t) (1 << 2))
#define UPSTAT_AUTOCTS          ((__force upstat_t) (1 << 3))
#define UPSTAT_AUTOXOFF         ((__force upstat_t) (1 << 4))

        int                 hw_stopped;      /* sw-assisted CTS flow state */
        unsigned int        mctrl;              /* current modem ctrl
settings */
        unsigned int        timeout;            /* character-based
timeout */
        unsigned int        type;               /* port type */
        const struct uart_ops   *ops;
        unsigned int        custom_divisor;
        unsigned int        line;               /* port index */
        unsigned int        minor;
        resource_size_t     mapbase;            /* for ioremap */
        resource_size_t     mapsize;
        struct device       *dev;               /* parent device */
        unsigned char       hub6;    /* this should be in the 8250 driver */
        unsigned char       suspended;
        unsigned char       irq_wake;
        unsigned char       unused[2];
        struct attribute_group  *attr_group;   /* port specific attributes */
        const struct attribute_group **tty_groups;     /* all attributes
(serial core use only) */
        struct serial_rs485     rs485;
        void                *private_data;/* generic platform data pointer */
};
```

1.3.3.2 UART device driver

```
struct uart_driver {
        struct module       *owner;
        const char          *driver_name;
        const char          *dev_name;
        int                 major;
        int                 minor;
        int                     nr;
        struct console          *cons;

        /*
         * these are private; the low level driver should not
         * touch these; they should be initialised to NULL
         */
        struct uart_state       *state;
        struct tty_driver       *tty_driver;
};
```

1.3.4 Function

1.3.4.1 uart_register_driver

prototype

```
#include <linux/serial_core.h>
int uart_register_driver(struct uart_driver *drv);
```

description

register a driver with the uart core layer

parameter

drv: low level driver structure

return

a negative error code or positive value

1.3.4.2 uart_unregister_driver

prototype

```
#include <linux/serial_core.h>
void uart_unregister_driver(struct uart_driver *drv)
```

description

remove a driver from the uart core layer

parameter

drv: low level driver structure

return

none

1.3.4.3 uart_add_one_port

prototype

```
#include <linux/serial_core.h>
int uart_add_one_port(struct uart_driver *drv, struct uart_port *uport)
```

description

attach a driver-defined port structure

parameter

drv: pointer to the uart low level driver structure for this port

return

zero on success, else a negative error code

1.3.4.4 uart_remove_one_port

prototype

```
#include <linux/serial_core.h>
int uart_remove_one_port(struct uart_driver *drv, struct uart_port *uport)
```

description

detach a driver defined port structure

parameter

drv: pointer to the uart low level driver structure for this port

uport: uart port structure for this port

return

zero on success; negative errno on failure.

1.3.5 Reference

1. Add your own device driver file `drivers/tty/serial/myuartdev.c` or customize or define other paths;
2. Compile a `uart_driver` structure and call `uart_register_driver` to register the driver to the tty core.
3. Write a `platform_driver` structure and call `platform_driver_register` to register as a platform driver.
4. If the device in the DTS matches this driver, you can execute `myuartdev_probe()` in the file `myuartdev.c`.

5. Define the variables of the `uart_port` and `uart_ops` structures to implement the struct `uart_ops` operation function.

6. Add ports through `uart_add_one_port`.

```
static struct uart_ops myuartdev_uart_ops = {
    .tx_empty   = myuartdev_uart_tx_empty,
    .set_mctrl  = myuartdev_uart_set_mctrl,
    .get_mctrl  = myuartdev_uart_get_mctrl,
    .stop_tx    = myuartdev_uart_stop_tx,
    .start_tx   = myuartdev_uart_start_tx,
    .stop_rx    = myuartdev_uart_stop_rx,
    .break_ctl  = myuartdev_uart_break_ctl,
    .startup    = myuartdev_uart_startup,
    .shutdown   = myuartdev_uart_shutdown,
    .set_termios    = myuartdev_uart_set_termios,
    .type       = myuartdev_uart_type,
    .release_port   = myuartdev_uart_release_port,
    .request_port   = myuartdev_uart_request_port,
    .config_port    = myuartdev_uart_config_port,
    .verify_port    = myuartdev_uart_verify_port,
};

static struct uart_driver myuartdev_uart_driver = {
    .owner      = THIS_MODULE,
    .driver_name    = DRIVER_NAME,
    .dev_name   = "ttyAMA",
    ...
};

static const struct of_device_id myuartdev_of_match[] = {
    {.compatible = "myuartdev",},
    { }
};
MODULE_DEVICE_TABLE(of, myuartdev_of_match);

static int myuartdev_probe(struct platform_device *pdev)
{
    up = devm_kzalloc(&pdev->dev, sizeof(struct ar933x_uart_port),
          GFP_KERNEL);
    if (!up)
    return -ENOMEM;
    ...
    port = &up->port;
    ...
    port->ops = &myuartdev_uart_ops;
    ...
    uart_add_one_port(&myuartdev_uart_driver, &up->port);
...
}

static int myuartdev_remove(struct platform_device *pdev)
{
    ...
}

static struct platform_driver myuartdev_uart_platform_driver = {
        .driver = {
                .name =       "myuartdev",
                .of_match_table = of_match_ptr(myuartdev_of_match),
        },
        .probe =      myuartdev_probe,
        .remove =     myuartdev_remove,
};

static int __init myuartdev_init(void)
{
    int ret;

    ret = uart_register_driver(&myuartdev_uart_driver);
    if (ret)
        goto err_out;

    ret = platform_driver_register(&myuartdev_uart_platform_driver);
    if (ret)
        goto err_unregister_uart_driver;

    return 0;

err_unregister_uart_driver:
    uart_unregister_driver(&myuartdev_uart_driver);
err_out:
    return ret;
}
static void __exit myuartdev_exit(void)
{
        platform_driver_unregister(&myuartdev_uart_platform_driver);
        uart_unregister_driver(&myuartdev_uart_driver);
}
```

```
arch_initcall(myuartdev_init);
module_exit(myuartdev_exit);
```