u-boot中的marvell pxa i2c的i2c read / write operation慢动作化了整个过程，对理解i2c协议很有帮助。
in u-boot/drivers/i2c/mv_i2c.c

```
int i2c_transfer(struct i2c_msg *msg)
{
        int ret;

        if (!msg)
                goto transfer_error_msg_empty;

        switch (msg->direction) {
        case I2C_WRITE:
                /* check if bus is not busy */
                if (!i2c_isr_set_cleared(0, ISR_IBB))
                        goto transfer_error_bus_busy;

                /* start transmission */
                writel(readl(&base->icr) & ~ICR_START, &base->icr);
                writel(readl(&base->icr) & ~ICR_STOP, &base->icr);
                writel(msg->data, &base->idbr);
                if (msg->condition == I2C_COND_START)
                        writel(readl(&base->icr) | ICR_START, &base->icr);
                if (msg->condition == I2C_COND_STOP)
                        writel(readl(&base->icr) | ICR_STOP, &base->icr);
                if (msg->acknack == I2C_ACKNAK_SENDNAK)
                        writel(readl(&base->icr) | ICR_ACKNAK, &base->icr);
                if (msg->acknack == I2C_ACKNAK_SENDACK)
                        writel(readl(&base->icr) & ~ICR_ACKNAK, &base->icr);
                writel(readl(&base->icr) & ~ICR_ALDIE, &base->icr);
                writel(readl(&base->icr) | ICR_TB, &base->icr);

                /* transmit register empty? */
                if (!i2c_isr_set_cleared(ISR_ITE, 0))
                        goto transfer_error_transmit_timeout;

                /* clear 'transmit empty' state */
                writel(readl(&base->isr) | ISR_ITE, &base->isr);

                /* wait for ACK from slave */
                if (msg->acknack == I2C_ACKNAK_WAITACK)
                        if (!i2c_isr_set_cleared(0, ISR_ACKNAK))
                                goto transfer_error_ack_missing;
                break;

        case I2C_READ:

                /* check if bus is not busy */
                if (!i2c_isr_set_cleared(0, ISR_IBB))
                        goto transfer_error_bus_busy;

                /* start receive */
                writel(readl(&base->icr) & ~ICR_START, &base->icr);
                writel(readl(&base->icr) & ~ICR_STOP, &base->icr);
                if (msg->condition == I2C_COND_START)
                        writel(readl(&base->icr) | ICR_START, &base->icr);
                if (msg->condition == I2C_COND_STOP)
```

```
                        writel(readl(&base->icr) | ICR_STOP, &base->icr);
                if (msg->acknack == I2C_ACKNAK_SENDNAK)
                        writel(readl(&base->icr) | ICR_ACKNAK, &base->icr);
                if (msg->acknack == I2C_ACKNAK_SENDACK)
                        writel(readl(&base->icr) & ~ICR_ACKNAK, &base->icr);
                writel(readl(&base->icr) & ~ICR_ALDIE, &base->icr);
                writel(readl(&base->icr) | ICR_TB, &base->icr);

                /* receive register full? */
                if (!i2c_isr_set_cleared(ISR_IRF, 0))
                        goto transfer_error_receive_timeout;

                msg->data = readl(&base->idbr);

                /* clear 'receive empty' state */
                writel(readl(&base->isr) | ISR_IRF, &base->isr);
                break;
        default:
                goto transfer_error_illegal_param;
        }

        return 0;

transfer_error_msg_empty:
                PRINTD(("i2c_transfer: error: 'msg' is empty\n"));
                ret = -1; goto i2c_transfer_finish;

transfer_error_transmit_timeout:
                PRINTD(("i2c_transfer: error: transmit timeout\n"));
                ret = -2; goto i2c_transfer_finish;

transfer_error_ack_missing:
                PRINTD(("i2c_transfer: error: ACK missing\n"));
                ret = -3; goto i2c_transfer_finish;

transfer_error_receive_timeout:
                PRINTD(("i2c_transfer: error: receive timeout\n"));
                ret = -4; goto i2c_transfer_finish;

transfer_error_illegal_param:
                PRINTD(("i2c_transfer: error: illegal parameters\n"));
                ret = -5; goto i2c_transfer_finish;

transfer_error_bus_busy:
                PRINTD(("i2c_transfer: error: bus is busy\n"));
                ret = -6; goto i2c_transfer_finish;

i2c_transfer_finish:
                PRINTD(("i2c_transfer: ISR: 0x%04x\n", readl(&base->isr)));
                i2c_reset();
                return ret;
}
```

i2c_transfer()只能read / write one byte。

```
struct i2c_msg {
        u8 condition;
        u8 acknack;
        u8 direction;
        u8 data;
};
```

data field就是要传输的byte
direction field决定是read or write
acknack决定是发送ACK还是NACK
condition field:

```
/* Shall the current transfer have a start/stop condition? */
#define I2C_COND_NORMAL         0
#define I2C_COND_START          1
#define I2C_COND_STOP           2
```

即transer的该byte需要附带START / STOP 信号还是没有(normal)

write one byte

```
                writel(readl(&base->icr) & ~ICR_START, &base->icr);     ①
                writel(readl(&base->icr) & ~ICR_STOP, &base->icr);
                writel(msg->data, &base->idbr);                         ②
                if (msg->condition == I2C_COND_START)
                        writel(readl(&base->icr) | ICR_START, &base->icr);     ③
                if (msg->condition == I2C_COND_STOP)
                        writel(readl(&base->icr) | ICR_STOP, &base->icr);
                if (msg->acknack == I2C_ACKNAK_SENDNAK)
                        writel(readl(&base->icr) | ICR_ACKNAK, &base->icr);
                if (msg->acknack == I2C_ACKNAK_SENDACK)
                        writel(readl(&base->icr) & ~ICR_ACKNAK, &base->icr);
                writel(readl(&base->icr) & ~ICR_ALDIE, &base->icr);     ④
                writel(readl(&base->icr) | ICR_TB, &base->icr);         ⑤

                /* transmit register empty? */
if (!i2c_isr_set_cleared(ISR_ITE, 0))                   ⑥
                        goto transfer_error_transmit_timeout;

                /* clear 'transmit empty' state */
                writel(readl(&base->isr) | ISR_ITE, &base->isr);        ⑦

                /* wait for ACK from slave */
                if (msg->acknack == I2C_ACKNAK_WAITACK)                 ⑧
                        if (!i2c_isr_set_cleared(0, ISR_ACKNAK))
                                goto transfer_error_ack_missing;
```

①
清除control register中的start and stop signal产生条件
②

把要传输的byte放入data register

③

根据调用函数的要求来产生start / stop / ack /nack signal

④

清除ALD (Arbitration Loss Detected)

⑤

设置TB bit，也就是启动传输

⑥

等待data register中的data被传输出去(ITE - IDBR Transmit Empty bit会告诉我们)

⑦

把ITE置位，这一步好像有点多余，因为上一不如果不是timeout则退出的条件就是ITE已被硬件置位。

⑧

在传输完一个bute后，查看是否要等待ACK signal

read one byte大同小异，就是查看IRF bit.

```c
/*
 * i2c_read: - Read multiple bytes from an i2c device
 *
 * The higher level routines take into account that this function is only
 * called with len < page length of the device (see configuration file)
 *
 * @chip:       address of the chip which is to be read
 * @addr:       i2c data address within the chip
 * @alen:       length of the i2c data address (1..2 bytes)
 * @buffer:     where to write the data
 * @len:        how much byte do we want to read
 * @return:     0 in case of success
 */
int i2c_read(uchar chip, uint addr, int alen, uchar *buffer, int len)
{
        struct i2c_msg msg;
        u8 addr_bytes[3]; /* lowest...highest byte of data address */

        PRINTD(("i2c_read(chip=0x%02x, addr=0x%02x, alen=0x%02x, "
                "len=0x%02x)\n", chip, addr, alen, len));

        i2c_reset();

        /* dummy chip address write */
        PRINTD(("i2c_read: dummy chip address write\n"));
        msg.condition = I2C_COND_START;            ①
        msg.acknack   = I2C_ACKNAK_WAITACK;
        msg.direction = I2C_WRITE;
        msg.data = (chip << 1);
        msg.data &= 0xFE;
        if (i2c_transfer(&msg))
                return -1;

        /*
         * send memory address bytes;
         * alen defines how much bytes we have to send.
         */
        /*addr &= ((1 << CONFIG_SYS_EEPROM_PAGE_WRITE_BITS)-1); */
        addr_bytes[0] = (u8)((addr >>  0) & 0x000000FF);        ②
        addr_bytes[1] = (u8)((addr >>  8) & 0x000000FF);
        addr_bytes[2] = (u8)((addr >> 16) & 0x000000FF);

        while (--alen >= 0) {        ③
                PRINTD(("i2c_read: send memory word address byte %1d\n", alen));
                msg.condition = I2C_COND_NORMAL;
                msg.acknack   = I2C_ACKNAK_WAITACK;
                msg.direction = I2C_WRITE;
                msg.data      = addr_bytes[alen];
                if (i2c_transfer(&msg))
                        return -1;
        }

        /* start read sequence */
```

```
54.         PRINTD(("i2c_read: start read sequence\n"));
55.         msg.condition = I2C_COND_START;                 ④
56.         msg.acknack   = I2C_ACKNAK_WAITACK;
57.         msg.direction = I2C_WRITE;
58.         msg.data      = (chip << 1);
59.         msg.data     |= 0x01;
60.         if (i2c_transfer(&msg))
61.                 return -1;
62.
63.         /* read bytes; send NACK at last byte */
64.         while (len--) {
65.                 if (len == 0) {
66.                         msg.condition = I2C_COND_STOP;          ⑤
67.                         msg.acknack   = I2C_ACKNAK_SENDNAK;
68.                 } else {
69.                         msg.condition = I2C_COND_NORMAL;        ⑥
70.                         msg.acknack   = I2C_ACKNAK_SENDACK;
71.                 }
72.
73.                 msg.direction = I2C_READ;
74.                 msg.data      = 0x00;
75.                 if (i2c_transfer(&msg))
76.                         return -1;
77.
78.                 *buffer = msg.data;                     ⑦
79.                 PRINTD(("i2c_read: reading byte (0x%08x)=0x%02x\n",
80.                         (unsigned int)buffer, *buffer));
81.                 buffer++;
82.         }
83.
84.         i2c_reset();
85.
86.         return 0;
87. }
```

这里的addr就是访问的i2c device内部的register的offset,其长度为alen。比如对256 bytes容量的eeprom，一个byte的addr就够了，但对于几十K的则addr显然要2 bytes。

①

首先发送START signal，然后写写该i2c device的address

②③

这两步是发送i2c device内部的register地址，这是normal write，不要发送START signal

④

开始真正的读取了。

发送START signal，然后还是写i2c device的address

⑤

如果读取的是最后一个byte，则需要发送STOP signal和NAK signal

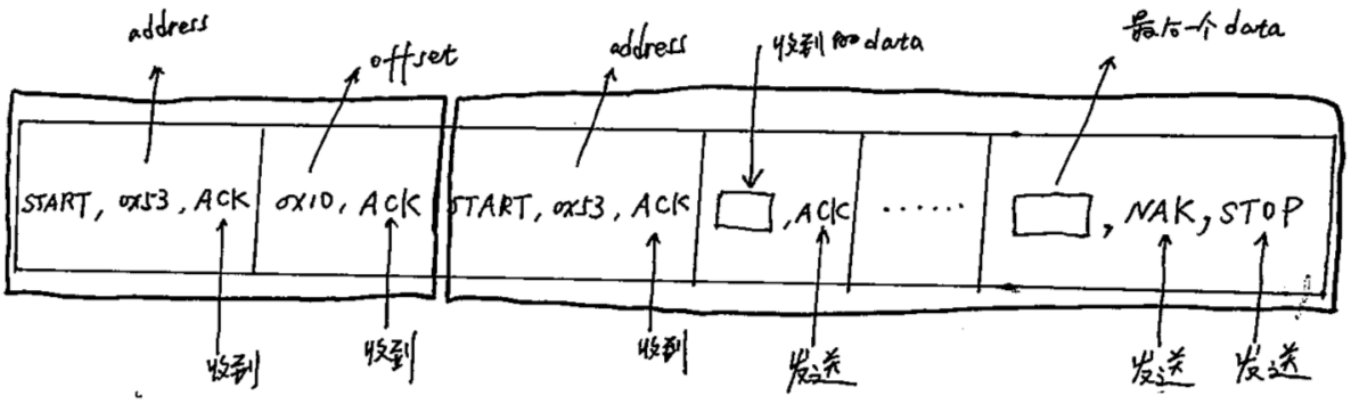
⑥

如果读取的是不是最后一个byte，那么只发送ACK signal

⑦

读取收到的byte

比如要读取0x53 i2c device的从0x10　offset开始的10个bytes,那么整个时序大致是这样的



I2C device 的address 为 0x53

从 0x10 offset 处读取 10个 bytes

每发送/接收一个byte，都要有 ACK/NAK对应

```c
/*
 * i2c_write: -  Write multiple bytes to an i2c device
 *
 * The higher level routines take into account that this function is only
 * called with len < page length of the device (see configuration file)
 *
 * @chip:       address of the chip which is to be written
 * @addr:       i2c data address within the chip
 * @alen:       length of the i2c data address (1..2 bytes)
 * @buffer:     where to find the data to be written
 * @len:        how much byte do we want to read
 * @return:     0 in case of success
 */
int i2c_write(uchar chip, uint addr, int alen, uchar *buffer, int len)
{
        struct i2c_msg msg;
        u8 addr_bytes[3]; /* lowest...highest byte of data address */

        PRINTD(("i2c_write(chip=0x%02x, addr=0x%02x, alen=0x%02x, "
                "len=0x%02x)\n", chip, addr, alen, len));

        i2c_reset();

        /* chip address write */
        PRINTD(("i2c_write: chip address write\n"));    ①
        msg.condition = I2C_COND_START;
        msg.acknack   = I2C_ACKNAK_WAITACK;
        msg.direction = I2C_WRITE;
        msg.data = (chip << 1);
        msg.data &= 0xFE;
        if (i2c_transfer(&msg))
                return -1;

        /*
         * send memory address bytes;
         * alen defines how much bytes we have to send.
         */
        addr_bytes[0] = (u8)((addr >>  0) & 0x000000FF);    ②
        addr_bytes[1] = (u8)((addr >>  8) & 0x000000FF);
        addr_bytes[2] = (u8)((addr >> 16) & 0x000000FF);

        while (--alen >= 0) {           ③
                PRINTD(("i2c_write: send memory word address\n"));
                msg.condition = I2C_COND_NORMAL;
                msg.acknack   = I2C_ACKNAK_WAITACK;
                msg.direction = I2C_WRITE;
                msg.data      = addr_bytes[alen];
                if (i2c_transfer(&msg))
                        return -1;
        }

        /* write bytes; send NACK at last byte */
        while (len--) {
```

```
54.                 PRINTD(("i2c_write: writing byte (0x%08x)=0x%02x\n",
55.                         (unsigned int)buffer, *buffer));
56.
57.                 if (len == 0)            ④
58.                         msg.condition = I2C_COND_STOP;
59.                 else                     ⑤
60.                         msg.condition = I2C_COND_NORMAL;
61.
62.                 msg.acknack   = I2C_ACKNAK_WAITACK;
63.                 msg.direction = I2C_WRITE;
64.                 msg.data      = *(buffer++);
65.
66.                 if (i2c_transfer(&msg))
67.                         return -1;
68.         }
69.
70.         i2c_reset();
71.
72.         return 0;
73. }
```

①

首先发送START，然后是i2c device address

②③

发送offset

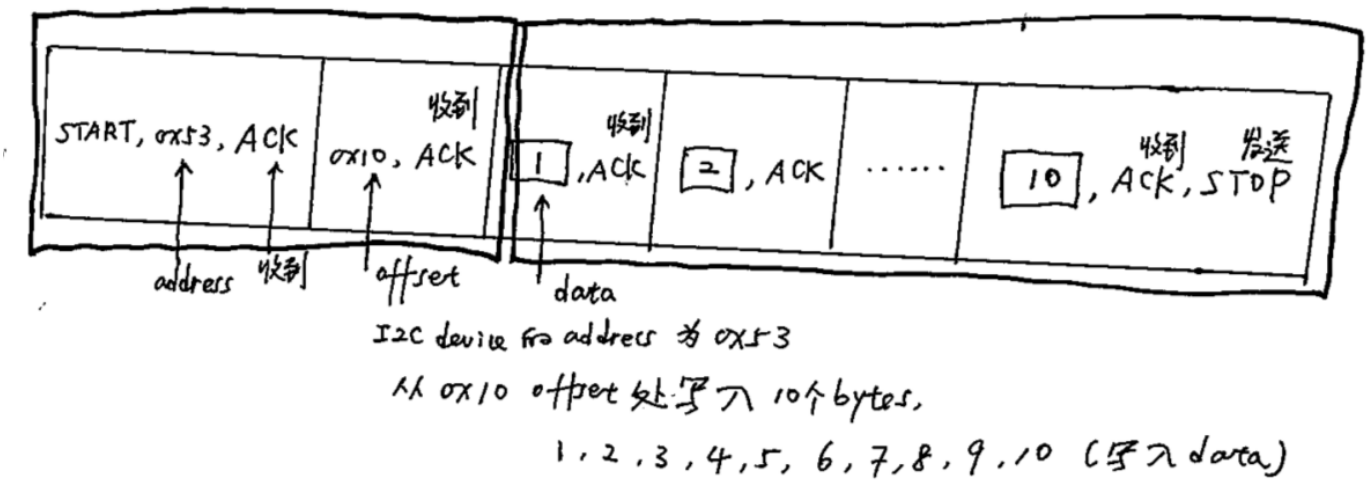
④

由于还是write，不像read一样，operation有所改变，所以这里在write data前无需再次发送i2c device address

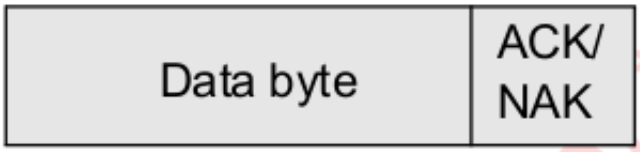
如果是最后一个发送的data，则要发送STOP signal

⑤

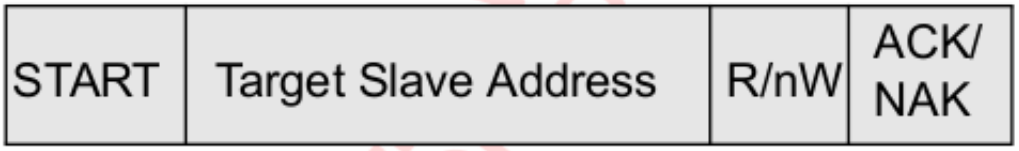
不是最后一个byte，无需其他signal



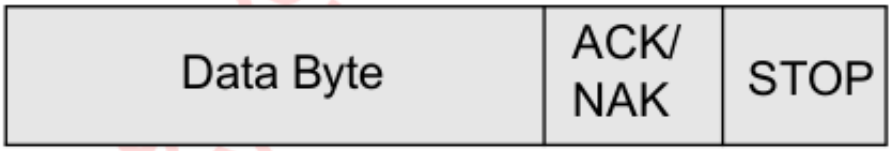
No Start or Stop Condition



Start Condition



Stop Condition



i2c_read() / i2c_write()中首先write i2c device adress就是上面的Start Condition。
发送和接受data(非最后一个)则是NO Start or Stop Condition
而发送或接收最后一个byte data则是Stop Condition。