目前IPC用于A53 cores 与 R4 core之间communication(主要用于传递low power message)。
在A53 cores上的Linux driver为
ccsgit/driver/ipc/ipc-mod/ipc_driver.c
而ccsgit/driver/ipc/ipc-mod/ipc_user_iface.c只用于输出访问ipc_driver.c中exported APIs的用户接口，如下

```
root@granite2:~# ls -l /sys/class/ipc/R4/
--w-------    1 root     root          4096 Aug  3 00:45 export
drwxr-xr-x    2 root     root             0 Aug  3 00:45 power
lrwxrwxrwx    1 root     root             0 Aug  3 00:44 subsystem -> ../../../../class/ipc
-rw-r--r--    1 root     root          4096 Aug  3 00:44 uevent
--w-------    1 root     root          4096 Aug  3 00:45 unexport
```

R4端的driver为
/home/walterzh/work2/LSP/ccsgit/r4/common/devices/ipc/mrvl_apb/src/mrvl_apb_ipc.c

---

A53端的IPC的Linux driver与R4端的threadx driver,核心代码逻辑几乎是完全一样的，唯一的差别就是因OS不同而必须的driver架构上的不同(Threadx OS无所谓driver framework)

IPC registers

register pair	receiver / sender	type
IPC_WDR_0 / IPC_WDR_1 / IPC_ISRW	sender	write only
IPC_RDR_0 / IPC_RDR_1 / IPC_IIR	receiver	read only

假设R4是message sender,A53是message receiver,反之依然。

R4往IPC_WDR_0 / IPC_WDR_1 / IPC_ISRW中写，尤其是往IPC_ISRW 写后，会触发A53端的IPC interrupt,A53通过读取
IPC_RDR_0 / IPC_RDR_1 / IPC_IIR就可以获得R4写入的值。就这么简单。

IPC_ISRW (IPC_IIR)只有11 bits有效，分别被encode成如下
bit 0 - bit 7 (8 bits)是port number
bit 8 (1 bit)是command bit(This name is confused), 实际上是sender设置的flag.即sender在发送message时会设置该flag，而receiver在isr中通过判断该flag来知道是接收message.
bit 9, bit 10(2 bits)用以receiver向sender告知sender发来的message是否被处理了。

IPC_WDR_0 — bit 0 - bit 23 (24 bits) message length
bit 24 - bit 31(8 bits) command (message type)

IPC_WDR_1 — message buffer address

- sender

in mrvl_apb_ipc.c/ipc_send()

```
    device = port->ipc_device;

    msg.type        = e_SEND;
    msg.device      = device;
    msg.port_number = port->port_number;
    msg.command     = command;
    msg.buffer      = buffer;
    msg.length      = length;

    sem_wait( &device->tx_ready_sem );    ①

    if ( posix_message_send( ipc_msg_queue, ( char* ) &msg, sizeof( msg ), 0, 0 ) != 0 )    ②
    {
        sem_post( &device->tx_ready_sem );        ③
        result = e_IPC_ERROR;
    }
    else
    {
        sem_wait( &device->tx_done_sem );        ④
        if ( device->ack_type == ACK_MSG_PROCESSED )      ⑤
        {
            result = e_IPC_SUCCESS;
        }
        else if ( device->ack_type == ACK_MSG_DISCARDED )      ⑥
        {
            result = e_IPC_NO_LISTENER;
        }
        else
        {
            result = e_IPC_ERROR;
        }
    }

    sem_post( &device->tx_ready_sem );
```

①

device->tx_ready_sem初始化为1，所以通过

②

posix_message_send()触发下面的code run

```
static void ipc_handle_message(ipc_internal_msg_t   *msg)
{
    ipc_device_config_t *device = msg->device;
    if ( device_is_valid( device ) )
    {
        ipc_port_config_t *port = find_device_port( device, msg->port_number );

        switch ( msg->type )
        {
            case e_ACK:
                ASIC_UARTDirect('a');
                sem_post( &device->tx_done_sem );
                break;

            case e_SEND:        ⑦
                ASIC_UARTDirect('s');
                if ( port != NULL )
                {
                    device->regs->IPC_WDR_0 = ( ( ( uint32_t ) msg->command ) << 24 ) | msg->length;
                    device->regs->IPC_WDR_1 = ( uint32_t ) msg->buffer;
                    device->regs->IPC_ISRW = ( port->port_number << IIR_PORT_SHIFT ) | ( IIR_CMD_MASK );  ⑧
                }
                else
                {
                    IPC_PRINTF( LOG_ERR, "tried to send on device/port (%d:%d) that isn't open\n", device->instance_id, msg->port_number);
                    sem_post( &device->tx_done_sem );        ⑨
                    device->ack_type = 0;
                }
                break;

            case e_RECV:
            {
                ASIC_UARTDirect('r');
                uint8_t ack_type = ACK_MSG_DISCARDED;

                IPC_PRINTF( LOG_DEBUG, "Device %d:%d received command %d, buffer 0x%p, len %d val %x\n", device->instance_id, msg->port_number, msg->command, msg->buffer, msg->length, ((uint32_t *)msg->buffer)[0]);
                if ( port != NULL )
                {
                    if ( port->recv_callback != NULL )
                    {
                        if ( ( msg->buffer != NULL ) && ( msg->length > 0 ) )
                        {
                            if ((uint32_t)msg->buffer >= hwGetRamStartAddress() && (uint32_t)msg->buffer < (hwGetRamStartAddress() + hwGetRamSize()))
                            {
                                cpu_dcache_invalidate_region(msg->buffer, CA
```

```
CHE_ALIGN_LENGTH(msg->length));
              }
            }
            port->recv_callback( port, port->user_param, msg->command, msg->buffer, msg->length );
            ack_type = ACK_MSG_PROCESSED;
          }
        }
        else
        {
          ack_type = ACK_MSG_DISCARDED;
          IPC_PRINTF( LOG_INFO, "Message for device/port (%d:%d) ignored because port isn't open\n", device->instance_id, msg->port_number);
        }

        // We've processed it, send the ACK so they can stage the next message
        device->regs->IPC_ISRW = ( ( uint32_t ) ack_type ) << IIR_ACK_SHIFT;
      }
      break;

      default:
        XASSERT("IPC received unexpected message type" == 0, msg->type);
        break;
    }
  }
  else
  {
    IPC_PRINTF( LOG_ERR, "IPC: received message on invalid device: 0x%08x\n", device);
  }
  ASIC_UARTDirect('x');
}
```

⑦

message send hander

⑧

设置IPC_WDR_0 / IPC_WDR_1 / IPC_ISRW，当write IPC_ISRW后，会触发A53端IPC产生interrupt

⑨

如果port为NULL，表示A53端并没有client在等待R4发送message，所以只是打印debug message,但这里的

> sem_post(&device->tx_done_sem);

非常重要，必须释放device->tx_done_sem,因为在另一条thread中运行的ipc_send()会在④处锁住自己(因为sender必须等待receiver的ACK信号才能往下运行)

③

发送message失败的处理
④

> sem_wait(&device->tx_done_sem);

R4与A53间发送与接收message是同步的，即R4发送了一条message后，会等待A53的回应，无论成功失败有了回应后才会往下运行，所以回应ACK是receiver必须的action，否则整个IPC会锁死。
⑤
在A53端有client接收来自R4的message
⑥
R4发送了message，但A53端根本无人理睬，discard the message.

- receiver

in ipc_driver.c/irq_handler()

```
static irqreturn_t irq_handler(int irq, void *dev_id)
{
    uint32_t iir;
    ipc_device_config_t *device = ( ipc_device_config_t * )dev_id;

    iowrite32(0, &device->regs->IPC_DUMMY);
    iir = ioread32(&device->regs->IPC_IIR);        (A)

    if ( iir & IIR_ACK_MASK )
    {
        device->ack_type = ( iir & IIR_ACK_MASK ) >> IIR_ACK_SHIFT;

        up(&device->tx_done_sem);

        iowrite32(IIR_ACK_MASK, &device->regs->IPC_ICR);
    }
    if ( iir & IIR_CMD_MASK )                    (B)
    {
        uint32_t  p1, p2;
        int ret;

        p1 = ioread32(&device->regs->IPC_RDR_0);    (C)
        p2 = ioread32(&device->regs->IPC_RDR_1);    (D)

        memset(&recv_data, 0, sizeof(recv_data));
        INIT_WORK( &recv_data.delayed_work, non_isr_recv );    (E)

        recv_data.cmd         = p1 >> 24;
        recv_data.len         = p1 & 0xFFFF;
        recv_data.buffer      = (char *)p2;
        recv_data.port_number = iir & IIR_PORT_MASK;
        recv_data.device      = device;

        iowrite32(IIR_CMD_MASK | IIR_PORT_MASK, &device->regs->IPC_ICR);

        ret = queue_work(ipc_workqueue, &recv_data.delayed_work);
    }

    return IRQ_HANDLED;
}
```

(A)

R4 write IPC's IPC_ISRW register, trigger A53 IPC interrupt. A53 read IPC_IIR register(也就是R4写入的IPC_ISRW register)

(B)

如果bit 8(command bit)置位(R4确实置位了)，表示R4有message发送过来，就进入message receieve handling

(C)

(D)

读取R4写入的IPC_WDR_0 and IPC_WDR_1

(E)

用work queue来实现真正读取。我觉得使用tasklet可能更合适。毕竟这是在服务interrupt service.

```
static void non_isr_recv( struct work_struct *work)
{
    recv_data_t *data = container_of( work, recv_data_t, delayed_work );
    uint8_t ack_type = ACK_MSG_DISCARDED;

    ENTER();

    if ( device_is_valid( data->device ) )
    {
        ipc_port_config_t *port;

        port = find_device_port(data->device, data->port_number);

        if ( port_is_valid(port) )
        {
            void *buffer_va = NULL;
            pr_debug("Port %d, rx cmd %d, buffer 0x%p, len %d\n", data->port_number, data->cmd, data->buffer, data->len);

            ack_type = ACK_MSG_PROCESSED;

            if ((data->buffer != NULL) && (data->len > 0))
            {
                request_mem_region((uint32_t)data->buffer, data->len, IPC_NAME);         (F)
                buffer_va = ioremap((uint32_t)data->buffer, data->len);           (G)

                port->recv_callback(port, port->user_param, data->cmd, buffer_va, data->len);    (H)

                iounmap(buffer_va);
                release_mem_region((uint32_t)data->buffer, data->len);
            }
            else
            {
                port->recv_callback(port, port->user_param, data->cmd, data->buffer, data->len);
            }
        }
        else
        {
            ack_type = ACK_MSG_DISCARDED;
            pr_debug("<CLOSED> Port %d, rx cmd %d, buffer 0x%p, len %d\n", data->port_number, data->cmd, data->buffer, data->len);
        }
    }

    iowrite32( ( ( uint32_t )ack_type ) << IIR_ACK_SHIFT, &data->device->regs->IPC_ISRW);    (I)

    EXIT();
}
```

(F)

(G)

由于R4工作在physical address mode，所以到Linux下需要mapping成virtual address

(H)

把message给真正的client

(I)

这是关键，A53 receiever 有责任发送ACK response。该code会trigger R4端的IPC interrupt.

in mrvl_apb_ipc.c/ipc_isr()

```
static void  ipc_isr( uint32_t input )
{
    ipc_device_config_t *device = ( ipc_device_config_t * )input;
    uint32_t iir;
    ipc_internal_msg_t msg;
    error_type_t msg_result;

    device->regs->IPC_DUMMY = 0;
    iir = device->regs->IPC_IIR;

    if ( iir & IIR_ACK_MASK )    (J)
    {
        ASIC_UARTDirect('A');
        msg.type = e_ACK;          (K)
        msg.device = device;
        device->ack_type = ( iir & IIR_ACK_MASK ) >> IIR_ACK_SHIFT;

        msg_result = posix_message_send( ipc_ack_msg_queue, ( char* ) &msg, sizeof( msg ), 0, 0 );
        ASSERT(msg_result == OK);

        // Clear the ACK interrupt
        device->regs->IPC_ICR = IIR_ACK_MASK;
    }
    if ( iir & IIR_CMD_MASK )
    {
        ASIC_UARTDirect('C');
        uint8_t  port_number;

        port_number = ( uint8_t )( ( iir & IIR_PORT_MASK ) >> IIR_PORT_SHIFT );

        msg.type        = e_RECV;
        msg.device      = device;
        msg.port_number = port_number;
        msg.command     = ( uint8_t ) ( device->regs->IPC_RDR_0 >> 24 );
        msg.length      = ( uint16_t )( device->regs->IPC_RDR_0 & 0xFFFF );
        msg.buffer      = ( void * ) device->regs->IPC_RDR_1;

        msg_result = posix_message_send( ipc_rx_msg_queue, ( char* ) &msg, sizeof( msg ), 0, 0 );
        ASSERT(msg_result == OK);

        // Clear the interrupt
        device->regs->IPC_ICR = IIR_CMD_MASK | IIR_PORT_MASK;
    }
}
```

(J)

A53 receiever 运行(I)会trigger R4的IPC interrupt.从IPC's IPC_IIR register读取的ACK被置位了

(K)

发送e_ACK message

```
static void ipc_handle_message(ipc_internal_msg_t    *msg)
{
    ipc_device_config_t *device = msg->device;
    if ( device_is_valid( device ) )
    {
        ipc_port_config_t *port = find_device_port( device, msg->port_number
 );

        switch ( msg->type )
        {
            case e_ACK:
                ASIC_UARTDirect('a');
                sem_post( &device->tx_done_sem );     (L)
                break;

......
```

(L)
这时候R4运行ipc_send()的thread还处于④处的lock状态，在这里就是释放该thread，使得ipc_send()能够继续运行。

- Summary

ipc_send()所涉及的执行流程比较复杂，涉及到R4和A53 core和多条thread的运行，从时间线上看，一条被正常处理的message流程大致如下
①②④⑦⑧(A)(B)(C)(D)(E)(F)(G)(H)(I)(J)(K)(L)④

这里的④就是

> sem_wait(&device->tx_done_sem);

第一个④让ipc_send() lock,而第二个④则是退出lock状态。也就是ipc_send() send message时，在④阶段，A53端已经接收到该message并处理完成了。