AMP

(Asymmetric Multiprocessing)

Ohad Ben-Cohen <ohad@wizery.com>

AMP ?

Why ?

Linux and AMP ?

What are we trying to do ?

1. simple (but allow both kernel and userland to talk with remote services)
2. performance
3. generic
4. upstream
5. focus on Linux, but
6. wire protocol is BSD

show me the code

```
static void sample_rx_callback(struct rpmsg_channel *rpdev,
                  void *data, int len, void *priv, u32 src)
{
      // incoming message ! Let's do something smart...
}

static int sample_probe(struct rpmsg_channel *rpdev)
{
      return rpmsg_send(rpdev, “dude”, 4);
}

static struct rpmsg_device_id sample_id_table[] = {
      { .name  = "rpmsg-client-sample" }, { },
};

static struct rpmsg_driver sample_client = {
       ...
      .id_table      = sample_id_table,
      .probe         = sample_probe,
      .callback      = sample_rx_callback,
};

static int __init client_sample_init(void)
{
      return register_rpmsg_driver(&sample_client);
}
```

```
static void sample_rx_callback(struct rpmsg_channel *rpdev,
                      void *data, int len, void *priv, u32 src)
{
      // incoming message ! Let's do something smart...
}

static int sample_probe(struct rpmsg_channel *rpdev)
{
      return rpmsg_send(rpdev, “dude”, 4);
}
```

1

```
static struct rpmsg_device_id sample_id_table[] = {
      { .name  = "rpmsg-client-sample" }, { },
};

static struct rpmsg_driver sample_client = {
       ...
      .id_table      = sample_id_table,
      .probe         = sample_probe,
      .callback      = sample_rx_callback,
};

static int __init client_sample_init(void)
{
      return register_rpmsg_driver(&sample_client);
}
```

```
static void sample_rx_callback(struct rpmsg_channel *rpdev,
                      void *data, int len, void *priv, u32 src)
{
      // incoming message ! Let's do something smart...
}

static int sample_probe(struct rpmsg_channel *rpdev)
{
      return rpmsg_send(rpdev, “dude”, 4);
}

static struct rpmsg_device_id sample_id_table[] = {
      { .name  = "rpmsg-client-sample" }, { },
};

static struct rpmsg_driver sample_client = {
       ...
      .id_table      = sample_id_table,
      .probe         = sample_probe,
      .callback      = sample_rx_callback,
};

static int __init client_sample_init(void)
{
      return register_rpmsg_driver(&sample_client);
}
```

2

```
static void sample_rx_callback(struct rpmsg_channel *rpdev,
                      void *data, int len, void *priv, u32 src)
{
      // incoming message ! Let's do something smart...
}

static int sample_probe(struct rpmsg_channel *rpdev)
{
      return rpmsg_send(rpdev, “dude”, 4);
}

static struct rpmsg_device_id sample_id_table[] = {
      { .name  = "rpmsg-client-sample" }, { },
};

static struct rpmsg_driver sample_client = {
       ...
      .id_table      = sample_id_table,
      .probe         = sample_probe,
      .callback      = sample_rx_callback,
};
```

3

```
static int __init client_sample_init(void)
{
      return register_rpmsg_driver(&sample_client);
}
```

```
static void sample_rx_callback(struct rpmsg_channel *rpdev,
                      void *data, int len, void *priv, u32 src)
{
      // incoming message ! Let's do something smart...
}
```

4

```
static int sample_probe(struct rpmsg_channel *rpdev)
{
      return rpmsg_send(rpdev, “dude”, 4);
}
```

```
static struct rpmsg_device_id sample_id_table[] = {
      { .name  = "rpmsg-client-sample" }, { },
};

static struct rpmsg_driver sample_client = {
       ...
      .id_table      = sample_id_table,
      .probe         = sample_probe,
      .callback      = sample_rx_callback,
};

static int __init client_sample_init(void)
{
      return register_rpmsg_driver(&sample_client);
}
```

5

```
static void sample_rx_callback(struct rpmsg_channel *rpdev,
                      void *data, int len, void *priv, u32 src)
{
      // incoming message ! Let's do something smart...
}

static int sample_probe(struct rpmsg_channel *rpdev)
{
      return rpmsg_send(rpdev, “dude”, 4);
}

static struct rpmsg_device_id sample_id_table[] = {
      { .name  = "rpmsg-client-sample" }, { },
};

static struct rpmsg_driver sample_client = {
       ...
      .id_table      = sample_id_table,
      .probe         = sample_probe,
      .callback      = sample_rx_callback,
};

static int __init client_sample_init(void)
{
      return register_rpmsg_driver(&sample_client);
}
```

status

LKML.ORG

Messages in this thread

- First message in thread
- **Ohad Ben-Cohen**
  - Ohad Ben-Cohen
  - Ohad Ben-Cohen
    - Sergei Shtylyov
      - "Grosen, Mark"
        - Sergei Shtylyov
  - Ohad Ben-Cohen
    - Rusty Russell
      - Sasha Levin
      - Ohad Ben-Cohen
    - Grant Likely
      - Ohad Ben-Cohen
        - Grant Likely
    - Randy Dunlap
      - Ohad Ben-Cohen
        - Arnd Bergmann
    - Pavel Machek
      - Ohad Ben-Cohen
  - Ohad Ben-Cohen
  - Ohad Ben-Cohen
    - Sergei Shtylyov
  - Ohad Ben-Cohen
  - Ohad Ben-Cohen
    - Will Newton
      - Ohad Ben-Cohen
    - Grant Likely
      - Ohad Ben-Cohen
        - Grant Likely
  - Ohad Ben-Cohen
    - Randy Dunlap
      - Ohad Ben-Cohen
    - Grant Likely
      - "Grosen, Mark"
        - Grant Likely

From Ohad Ben-Cohen <>
Subject [RFC 0/8] Introducing a generic AMP/IPC framework +1 0
Date Tue, 21 Jun 2011 10:18:26 +0300

Modern SoCs typically employ a central symmetric multiprocessing (SMP) application processor running Linux, with several other asymmetric multiprocessing (AMP) heterogeneous processors running different instances of operating system, whether Linux or any other flavor of real-time OS.

OMAP4, for example, has dual Cortex-A9, dual Cortex-M3 and a C64x+ DSP. Typically, the dual cortex-A9 is running Linux in a SMP configuration, and each of the other three cores (two M3 cores and a DSP) is running its own instance of RTOS in an AMP configuration.

AMP remote processors typically employ dedicated DSP codecs and multimedia hardware accelerators, and therefore are often used to offload cpu-intensive multimedia tasks from the main application processor. They could also be used to control latency-sensitive sensors, drive "random" hardware blocks, or just perform background tasks while the main CPU is idling.

Users of those remote processors can either be userland apps (e.g. multimedia frameworks talking with remote OMX components) or kernel drivers (controlling hardware accessible only by the remote processor, reserving kernel-controlled resources on behalf of the remote processor, etc..).

This patch set adds a generic AMP/IPC framework which makes it possible to control (power on, boot, power off) and communicate (simply send and receive messages) with those remote processors.

LKML.ORG

Messages in this thread

- First message in thread
- **Ohad Ben-Cohen**
- Ohad Ben-Cohen
- Ohad Ben-Cohen
- Ohad Ben-Cohen
- Ohad Ben-Cohen
- Ohad Ben-Cohen
- Ohad Ben-Cohen
- Ohad Ben-Cohen

From Ohad Ben-Cohen <>
Subject [PATCH 0/7] Introducing a generic AMP framework +1 0
Date Tue, 25 Oct 2011 11:48:19 +0200

Modern SoCs typically employ a central symmetric multiprocessing (SMP)
application processor running Linux, with several other asymmetric
multiprocessing (AMP) heterogeneous processors running different instances
of operating system, whether Linux or any other flavor of real-time OS.

OMAP4, for example, has dual Cortex-A9, dual Cortex-M3 and a C64x+ DSP.
Typically, the dual cortex-A9 is running Linux in a SMP configuration, and
each of the other three cores (two M3 cores and a DSP) is running its own
instance of RTOS in an AMP configuration.

AMP remote processors typically employ dedicated DSP codecs and multimedia
hardware accelerators, and therefore are often used to offload cpu-intensive
multimedia tasks from the main application processor. They could also be
used to control latency-sensitive sensors, drive 'random' hardware blocks,
or just perform background tasks while the main CPU is idling.

Users of those remote processors can either be userland apps (e.g.
multimedia frameworks talking with remote OMX components) or kernel drivers
(controlling hardware accessible only by the remote processor, reserving
kernel-controlled resources on behalf of the remote processor, etc..).

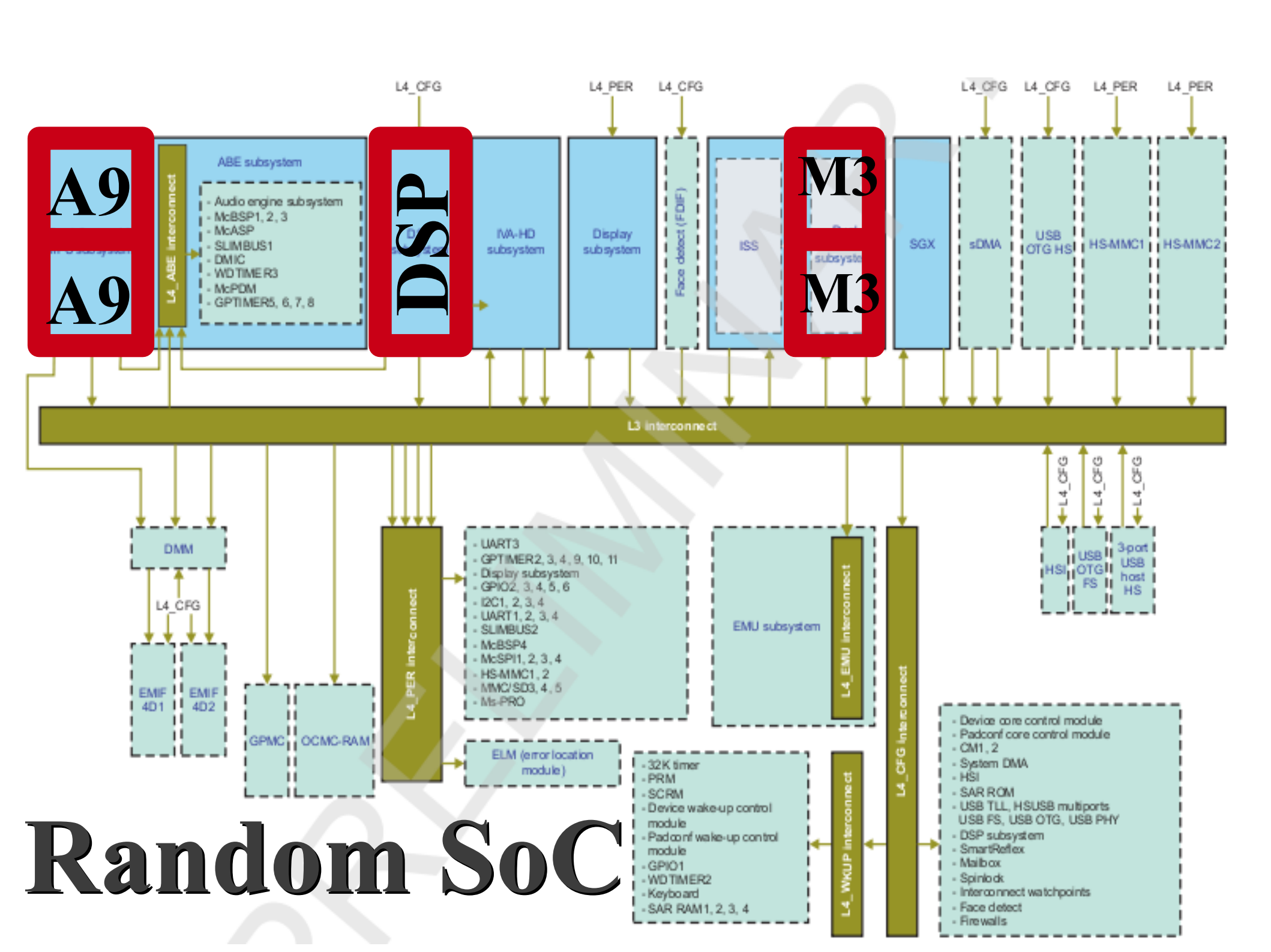
This patch set adds a generic AMP framework which makes it possible to
control (power on, boot, power off) and communicate (simply send and receive
messages) with those remote processors.


Google
SAMSUNG
4:00
Google
Google+
Google

details

small letters

- On-chip processors
- No bus (RapidIO ? PCIe ?)
- Can access memory
- Can kick each other
- … common SoC layout


A9
A9
L4_ABE interconnect
ABE subsystem
- Audio engine subsystem
- McBSP1, 2, 3
- McASP
- SLIMBUS1
- DMIC
- WDTIMER3
- McPDM
- GPTIMER5, 6, 7, 8
DSP
IVA-HD subsystem
Display subsystem
Face detect (FDIF)
ISS
M3
M3
SGX
sDMA
USB OTG HS
HS-MMC1
HS-MMC2
L4_CFG
L4_PER
L3 interconnect
DMM
L4_CFG
EMIF 4D1
EMIF 4D2
GPMC
OCMC-RAM
L4_PER interconnect
- UART3
- GPTIMER2, 3, 4, 9, 10, 11
- Display subsystem
- GPIO2, 3, 4, 5, 6
- I2C1, 2, 3, 4
- UART1, 2, 3, 4
- SLIMBUS2
- McBSP4
- McSPI1, 2, 3, 4
- HS-MMC1, 2
- MMC/SD3, 4, 5
- Ms-PRO
ELM (error location module)
EMU subsystem
L4_EMU interconnect
HSI
USB OTG FS
3-port USB host HS
L4_CFG interconnect
- 32K timer
- PRM
- SCRM
- Device wake-up control module
- Padconf wake-up control module
- GPIO1
- WDTIMER2
- Keyboard
- SAR RAM1, 2, 3, 4
L4_WKUP interconnect
- Device core control module
- Padconf core control module
- CM1, 2
- System DMA
- HSI
- SAR ROM
- USB TLL, HSUSB multiports
USB FS, USB OTG, USB PHY
- DSP subsystem
- SmartReflex
- Mailbox
- Spinlock
- Interconnect watchpoints
- Face detect
- Firewalls
Random SoC

1.Control

drivers/amp/remoteproc

- **Load a firmware**
- **Allocate resources**
- **Boot**

rproc_...( )

- **int rproc_boot(struct rproc *)**
- **int rproc_shutdown(struct rproc *)**

- **rproc *rproc_get_by_name(char *)**
- **int rproc_put(struct rproc *)**

Try not to use these

Two different refcounts!

2. Input/Output

drivers/amp/rpmsg

- Send messages
- Receive messages
- Multiple channels

rpmsg_...( )

- int rpmsg_send(rpdev, data, len)
- int rpmsg_sendto(rpdev, data, len, dst)
- int rpmsg_send_offchannel(rpdev, src, dst, data, len)

- int rpmsg_trysend(rpdev, data, len)
- int rpmsg_trysendto(rpdev, data, len, dst)
- int rpmsg_trysend_offchannel(rpdev, src, dst, data, len)

- struct rpmsg_endpoint *
  rpmsg_create_ept(rpdev, callback, priv, addr)
- void rpmsg_destroy_ept(struct rpmsg_endpoint *);

On channels

- Rpmsg a bus, channels: devices
- Match by name
- Carry src + dst addresses
- Same physical medium
- Can be dynamically allocated
- Drivers, callbacks and probe()
- Userland ? Sure, but...

Under the hood:

VirtIO

Thanks Rusty and Arnd!

Virtqueues

- A transport abstraction
- API for posting buffers for consumption

```
struct virtqueue_ops {
        int (*add_buf)(struct virtqueue *vq,
                       struct scatterlist sg[],
                       unsigned int out_num,
                       unsigned int in_num,
                       void *data);
        void (*kick)(struct virtqueue *vq);
        void *(*get_buf)(struct virtqueue *vq, unsigned int *len);
        void (*disable_cb)(struct virtqueue *vq);
        bool (*enable_cb)(struct virtqueue *vq);
};
```

- Each device has one (or more) virtqueues
- A virtio driver uses the above API for I/O
- Asymmetric in nature: Guest ↔ Host

Virtqueues

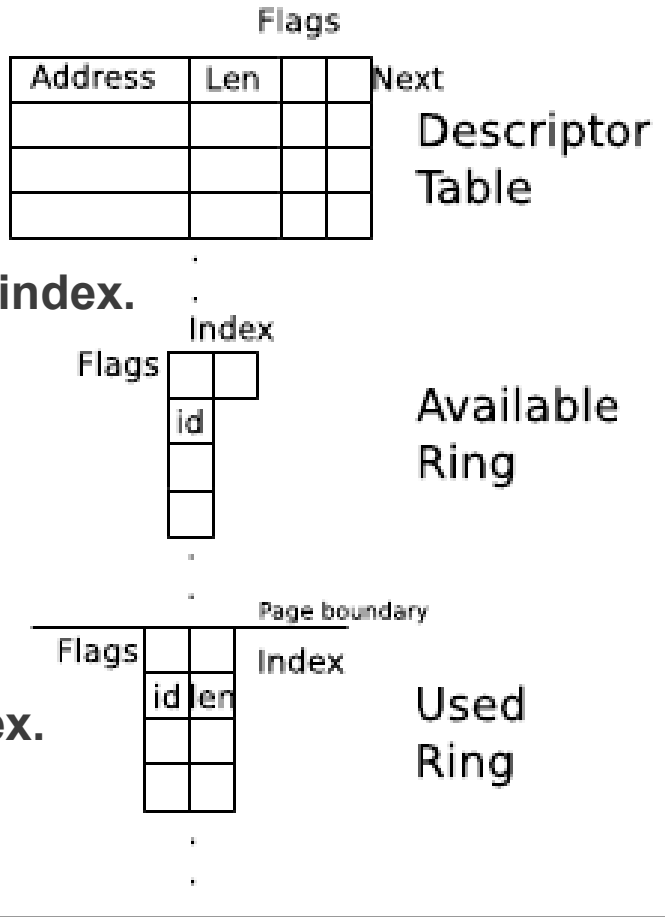
- Was collapsed into a single implementation:

```
commit 7c5e9ed0c84e7d70d887878574590638d5572659
Author: Michael S. Tsirkin <mst@redhat.com>
Date:   Mon Apr 12 16:19:07 2010 +0300

    virtio_ring: remove a level of indirection

    We have a single virtqueue_ops implementation,
    and it seems unlikely we'll get another one
    at this point. So let's remove an unnecessary
    level of indirection: it would be very easy to
    re-add it if another implementation surfaces.

    Signed-off-by: Michael S. Tsirkin <mst@redhat.com>
    Signed-off-by: Rusty Russell <rusty@rustcorp.com.au>

 drivers/virtio/virtio_ring.c |   36 +++++++++------------
 include/linux/virtio.h       |   71 ++++++++++-------------------------------
 2 files changed, 34 insertions(+), 73 deletions(-)
```

vring

- The only virtqueue implementation today

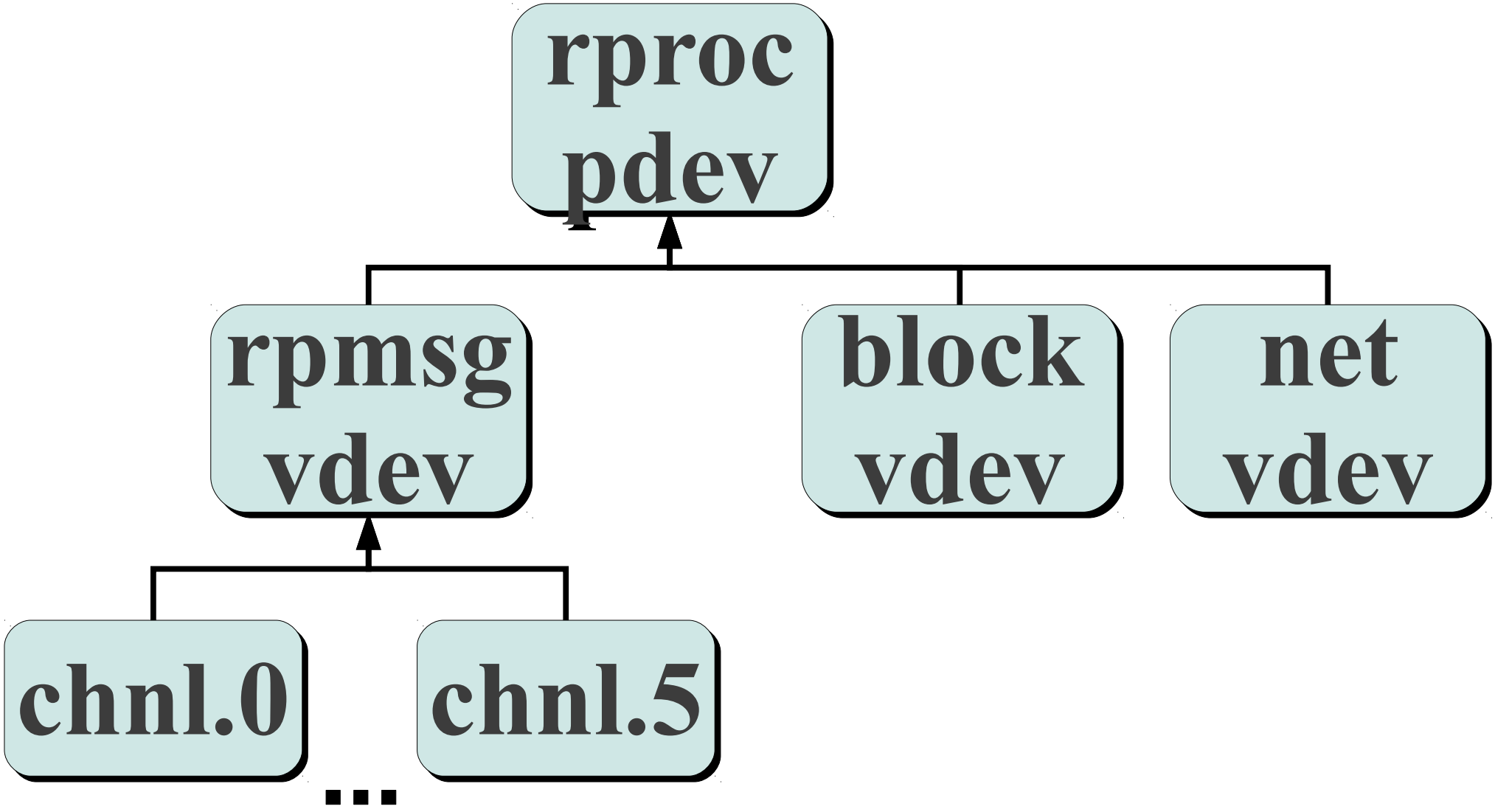
```
struct vring
{
    // The actual descriptors (16 bytes each)
    struct vring_desc desc[num];

    // A ring of available descriptor heads with free-running index.
    __u16 avail_flags;
    __u16 avail_idx;
    __u16 available[num];

    // Padding to the next align boundary.
    char pad[];

    // A ring of used descriptor heads with free-running index.
    __u16 used_flags;
    __u16 used_idx;
    struct vring_used_elem used[num];
};
```



rpmsg and virtio

- One vring per core
- Bus-owned buffers
- Messages begin with header
- #define VIRTIO_ID_RPMSG 7
- Dynamically create vdevs ! (not implemented yet)

Device topology



Other goodies

device model

- Name service for free !
- Crash ? Just remove the parent
- Runtime PM API
- DMA, CMA and IOMMU API

OK, seems nice.
How do I add
support for my
xyz platform?

xyz AMP support

```
static struct rproc_ops xyz_rproc_ops = {
    .start      = xyz_rproc_start,
    .stop       = xyz_rproc_stop,
    .kick       = xyz_rproc_kick,
};

static int __devinit xyz_rproc_probe(struct platform_device *pdev)
{
    struct xyz_rproc_pdata *pdata = pdev->dev.platform_data;
    struct xyz_rproc *my_proc;
    struct rproc *rproc;
    int ret;

    rproc = rproc_alloc(&pdev->dev, pdata->name, &xyz_rproc_ops,
                                    pdata->firmware, sizeof(*my_proc));

    … xyz-specific initialization ...

    ret = rproc_register(rproc);
```

xyz AMP support

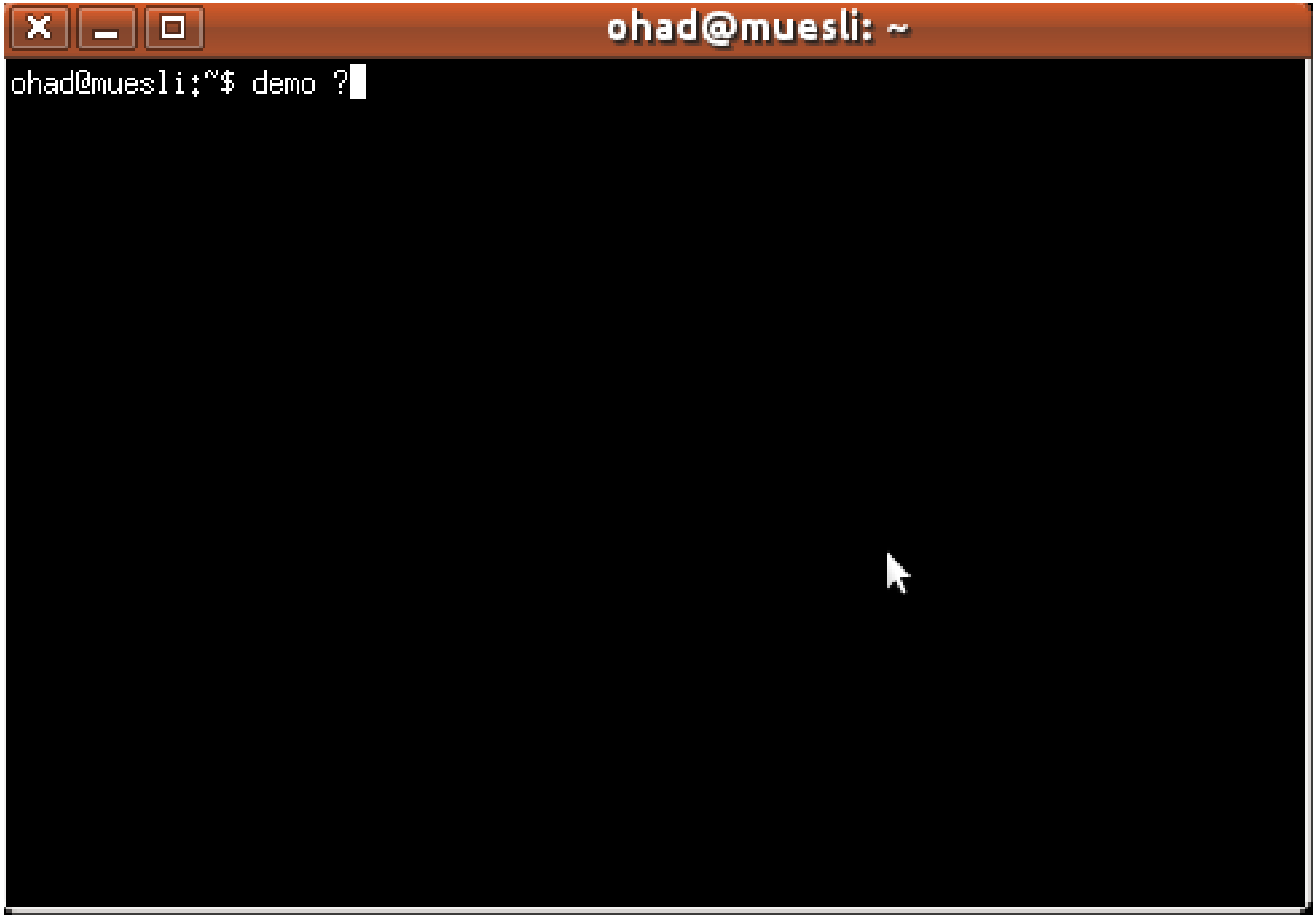
```
static int xyz_rproc_start(struct rproc *rproc)
{
    … power on the core using xyz-specific stuff ...
}

static int xyz_rproc_stop(struct rproc *rproc)
{
    … power off the core using xyz-specific stuff ...
}

static void xyz_rproc_kick(struct rproc *rproc, int vqid)
{
    … tell core vqid has a msg … xyz-specific stuff ...
}
```

Er, firmware

- Linux, RTOS or whatnot
- Put in /lib/firmware
- Only ELF (at this point)
- Special section: ".resource_table"
- Specifies resource requirements
- Gets handled before boot
- Can also specify features
- E.g. virtio header resources

status

- **Platforms:**
  - **OMAP4**
  - **DaVinci real soonish**
  - **We want more (but can't find info)**
- **In the pipeline:**
  - **More VirtIO !**
  - **Socket interface**
  - **OMX offloading driver**
  - **Resource manager**
  - **...**


ohad@muesli: ~
ohad@muesli:~$ demo ?

Credits

Brian Swetland

Suman Anna
Fernando Lugo Guzman
Iliyan Malchev

Mark Grosen
G Anthony

Grant Likely
Arnd Bergmann
Rusty Russell

Thanks!

Patches:
https://lkml.org/lkml/2011/10/25/139

Slides:
http://wizery.com/ohad/AMP_ELCE2011.odp