Marvell OcteonTx2 RVU Kernel Drivers

Contents

Overview

Resource virtualization unit (RVU) on Marvell's OcteonTX2 SOC maps HW resources from the network, crypto and other functional blocks into PCI-compatible physical and virtual functions. Each functional block again has multiple local functions (LFs) for provisioning to PCI devices. RVU supports multiple PCIe SRIOV physical functions (PFs) and virtual functions (VFs). PF0 is called the administrative / admin function (AF) and has privileges to provision RVU functional block's LFs to each of the PF/VF.

RVU managed networking functional blocks
- Network pool or buffer allocator (NPA)
- Network interface controller (NIX)
- Network parser CAM (NPC)
- Schedule/Synchronize/Order unit (SSO)
- Loopback interface (LBK)

RVU managed non-networking functional blocks
- Crypto accelerator (CPT)
- Scheduled timers unit (TIM)
- Schedule/Synchronize/Order unit (SSO) Used for both networking and non networking usecases

Resource provisioning examples
- A PF/VF with NIX-LF & NPA-LF resources works as a pure network device
- A PF/VF with CPT-LF resource works as a pure crypto offload device.

RVU functional blocks are highly configurable as per software requirements.

Firmware setups following stuff before kernel boots
- Enables required number of RVU PFs based on number of physical links.
- Number of VFs per PF are either static or configurable at compile time. Based on config, firmware assigns VFs to each of the PFs.
- Also assigns MSIX vectors to each of PF and VFs.
- These are not changed after kernel boot.

Drivers

Linux kernel will have multiple drivers registering to different PF and VFs of RVU. Wrt networking there will be 3 flavours of drivers.

Admin Function driver

As mentioned above RVU PF0 is called the admin function (AF), this driver supports resource provisioning and configuration of functional blocks. Doesn't handle any I/O. It sets up few basic stuff but most of the funcionality is achieved via configuration requests from PFs and VFs.

PF/VFs communicates with AF via a shared memory region (mailbox). Upon receiving requests AF does resource provisioning and other HW configuration. AF is always attached to host kernel, but PFs and their VFs may be used by host kernel itself, or attached to VMs or to userspace applications like DPDK etc. So AF has to handle provisioning/configuration requests sent by any device from any domain.

AF driver also interacts with underlying firmware to
- Manage physical ethernet links ie CGX LMACs.
- Retrieve information like speed, duplex, autoneg etc
- Retrieve PHY EEPROM and stats.
- Configure FEC, PAM modes
- etc

From pure networking side AF driver supports following functionality.
- Map a physical link to a RVU PF to which a netdev is registered.
- Attach NIX and NPA block LFs to RVU PF/VF which provide buffer pools, RQs, SQs for regular networking functionality.
- Flow control (pause frames) enable/disable/config.
- HW PTP timestamping related config.
- NPC parser profile config, basically how to parse pkt and what info to extract.
- NPC extract profile config, what to extract from the pkt to match data in MCAM entries.
- Manage NPC MCAM entries, upon request can frame and install requested packet forwarding rules.
- Defines receive side scaling (RSS) algorithms.
- Defines segmentation offload algorithms (eg TSO)
- VLAN stripping, capture and insertion config.
- SSO and TIM blocks config which provide packet scheduling support.
- Debugfs support, to check current resource provising, current status of NPA pools, NIX RQ, SQ and CQs, various stats etc which helps in debugging issues.
- And many more.

Physical Function driver

This RVU PF handles IO, is mapped to a physical ethernet link and this driver registers a netdev. This supports SR-IOV. As said above this driver communicates with AF with a mailbox. To retrieve information from physical links this driver talks to AF and AF gets that info from firmware and responds back ie cannot talk to firmware directly.

Supports ethtool for configuring links, RSS, queue count, queue size, flow control, ntuple filters, dump PHY EEPROM, config FEC etc.

Virtual Function driver

There are two types VFs, VFs that share the physical link with their parent SR-IOV PF and the VFs which work in pairs using internal HW loopback channels (LBK).

Type1:

- These VFs and their parent PF share a physical link and used for outside communication.
- VFs cannot communicate with AF directly, they send mbox message to PF and PF forwards that to AF. AF after processing, responds back to PF and PF forwards the reply to VF.
- From functionality point of view there is no difference between PF and VF as same type HW resources are attached to both. But user would be able to configure few stuff only from PF as PF is treated as owner/admin of the link.

Type2:

- RVU PF0 ie admin function creates these VFs and maps them to loopback block's channels.
- A set of two VFs (VF0 & VF1, VF2 & VF3 .. so on) works as a pair ie pkts sent out of VF0 will be received by VF1 and viceversa.
- These VFs can be used by applications or virtual machines to communicate between them without sending traffic outside. There is no switch present in HW, hence the support for loopback VFs.
- These communicate directly with AF (PF0) via mbox.

Except for the IO channels or links used for packet reception and transmission there is no other difference between these VF types. AF driver takes care of IO channel mapping, hence same VF driver works for both types of devices.

Basic packet flow

Ingress

1. CGX LMAC receives packet.
2. Forwards the packet to the NIX block.
3. Then submitted to NPC block for parsing and then MCAM lookup to get the destination RVU device.
4. NIX LF attached to the destination RVU device allocates a buffer from RQ mapped buffer pool of NPA block LF.
5. RQ may be selected by RSS or by configuring MCAM rule with a RQ number.
6. Packet is DMA'ed and driver is notified.

Egress

1. Driver prepares a send descriptor and submits to SQ for transmission.
2. The SQ is already configured (by AF) to transmit on a specific link/channel.
3. The SQ descriptor ring is maintained in buffers allocated from SQ mapped pool of NPA block LF.
4. NIX block transmits the pkt on the designated channel.
5. NPC MCAM entries can be installed to divert pkt onto a different channel.

Devlink health reporters

NPA Reporters

The NPA reporters are responsible for reporting and recovering the following group of errors:

1. GENERAL events
   - Error due to operation of unmapped PF.
   - Error due to disabled alloc/free for other HW blocks (NIX, SSO, TIM, DPI and AURA).
2. ERROR events
   - Fault due to NPA_AQ_INST_S read or NPA_AQ_RES_S write.
   - AQ Doorbell Error.
3. RAS events
   - RAS Error Reporting for NPA_AQ_INST_S/NPA_AQ_RES_S.
4. RVU events
   - Error due to unmapped slot.

Sample Output:

```
~# devlink health
pci/0002:01:00.0:
  reporter hw_npa_intr
      state healthy error 2872 recover 2872 last_dump_date 2020-12-10 last_dump_time 09:39:09 grace_period 0 auto_recover
  reporter hw_npa_gen
      state healthy error 2872 recover 2872 last_dump_date 2020-12-11 last_dump_time 04:43:04 grace_period 0 auto_recover
  reporter hw_npa_err
      state healthy error 2871 recover 2871 last_dump_date 2020-12-10 last_dump_time 09:39:17 grace_period 0 auto_recover
   reporter hw_npa_ras
      state healthy error 0 recover 0 last_dump_date 2020-12-10 last_dump_time 09:32:40 grace_period 0 auto_recover true a
```

Each reporter dumps the

- Error Type
- Error Register value
- Reason in words

For example:

```
~# devlink health dump show  pci/0002:01:00.0 reporter hw_npa_gen
 NPA_AF_GENERAL:
         NPA General Interrupt Reg : 1
         NIX0: free disabled RX
~# devlink health dump show  pci/0002:01:00.0 reporter hw_npa_intr
 NPA_AF_RVU:
         NPA RVU Interrupt Reg : 1
         Unmap Slot Error
~# devlink health dump show  pci/0002:01:00.0 reporter hw_npa_err
 NPA_AF_ERR:
        NPA Error Interrupt Reg : 4096
        AQ Doorbell Error
```

NIX Reporters

The NIX reporters are responsible for reporting and recovering the following group of errors:

1. GENERAL events
   - Receive mirror/multicast packet drop due to insufficient buffer.
   - SMQ Flush operation.

2. ERROR events
    - Memory Fault due to WQE read/write from multicast/mirror buffer.
    - Receive multicast/mirror replication list error.
    - Receive packet on an unmapped PF.
    - Fault due to NIX_AQ_INST_S read or NIX_AQ_RES_S write.
    - AQ Doorbell Error.
3. RAS events
    - RAS Error Reporting for NIX Receive Multicast/Mirror Entry Structure.
    - RAS Error Reporting for WQE/Packet Data read from Multicast/Mirror Buffer..
    - RAS Error Reporting for NIX_AQ_INST_S/NIX_AQ_RES_S.
4. RVU events
    - Error due to unmapped slot.

Sample Output:

```
~# ./devlink health
pci/0002:01:00.0:
  reporter hw_npa_intr
    state healthy error 0 recover 0 grace_period 0 auto_recover true auto_dump true
  reporter hw_npa_gen
    state healthy error 0 recover 0 grace_period 0 auto_recover true auto_dump true
  reporter hw_npa_err
    state healthy error 0 recover 0 grace_period 0 auto_recover true auto_dump true
  reporter hw_npa_ras
    state healthy error 0 recover 0 grace_period 0 auto_recover true auto_dump true
  reporter hw_nix_intr
    state healthy error 1121 recover 1121 last_dump_date 2021-01-19 last_dump_time 05:42:26 grace_period 0 auto_recover tr
  reporter hw_nix_gen
    state healthy error 949 recover 949 last_dump_date 2021-01-19 last_dump_time 05:42:43 grace_period 0 auto_recover true
  reporter hw_nix_err
    state healthy error 1147 recover 1147 last_dump_date 2021-01-19 last_dump_time 05:42:59 grace_period 0 auto_recover tr
  reporter hw_nix_ras
    state healthy error 409 recover 409 last_dump_date 2021-01-19 last_dump_time 05:43:16 grace_period 0 auto_recover true
```

Each reporter dumps the

- Error Type
- Error Register value
- Reason in words

For example:

```
~# devlink health dump show pci/0002:01:00.0 reporter hw_nix_intr
 NIX_AF_RVU:
        NIX RVU Interrupt Reg : 1
        Unmap Slot Error
~# devlink health dump show pci/0002:01:00.0 reporter hw_nix_gen
 NIX_AF_GENERAL:
        NIX General Interrupt Reg : 1
        Rx multicast pkt drop
~# devlink health dump show pci/0002:01:00.0 reporter hw_nix_err
 NIX_AF_ERR:
        NIX Error Interrupt Reg : 64
        Rx on unmapped PF_FUNC
```