DPDK ON EMBEDDED NETWORKING SOCS
- EXPERIENCE & NEEDS

HEMANT AGRAWAL & SHREYANSH JAIN
NXP

10TH AUGUST 2016

DPDK US Summit - San Jose - 2016

SECURE CONNECTIONS FOR A SMARTER WORLD

AGENDA

- NXP’s experience on DPDK
- SoC support in DPDK
- Current Status

NXP (formerly Freescale)

- NXP platform supports user space data path APIs
- NXP is founding member of Linaro LNG
  - participate and contribute to ODP
- NXP supports DPDK,
  - now participating and contributing to DPDK.
- DPDK can evolve to truly support multiple architectures and acceleration technologies while still retaining the goal of portable software.

Using DPDK on NXP SoCs

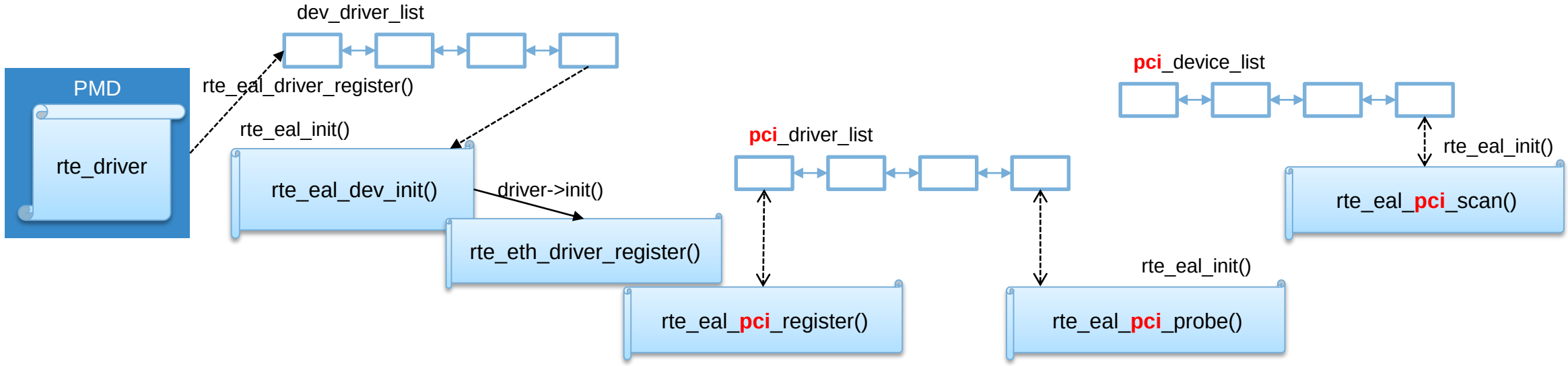
Main Goal is to add NXP platform support in DPDK.

- Compiling DPDK for NXP ARM device was easy
  - DPDK 16.07 supports nxp platform configuration
    - defconfig_arm64-dpaa2-linuxapp-gcc

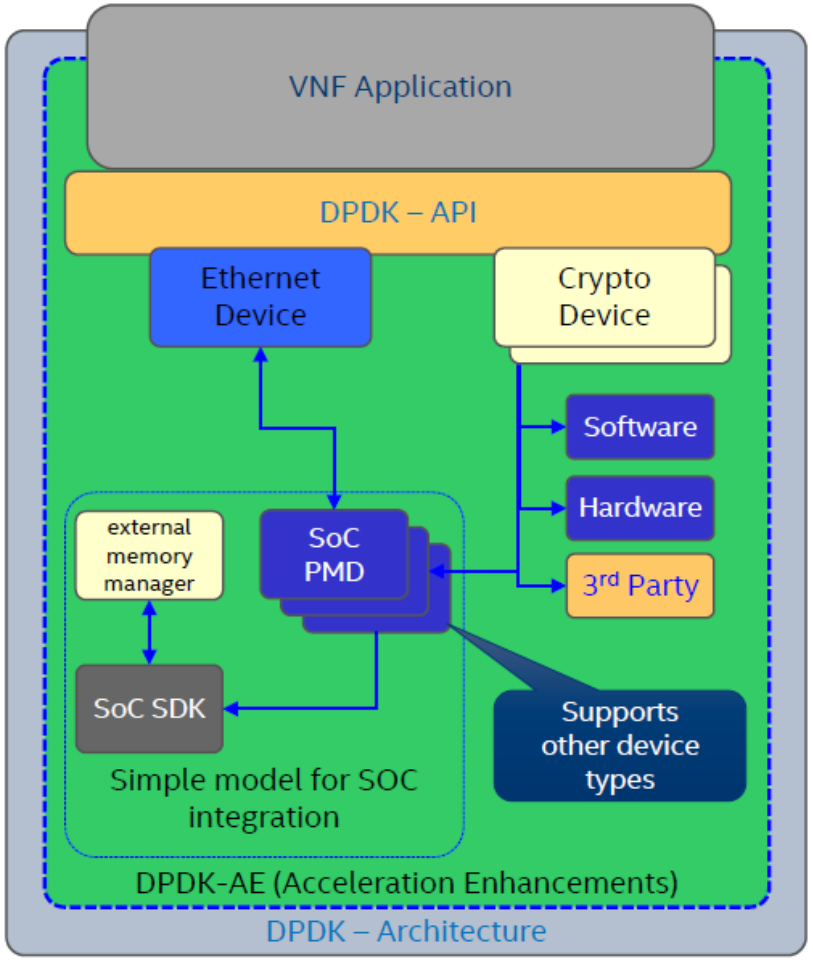
- However, DPDK lacked support for in-built MACs and other data path accelerators.
  - NXP SoCs have in-built MAC and they are non-PCI based.
  - BMAN - Packet buffer to be allocated & managed by HW
  - QMAN - Packet Queues mapped to hardware queues
  - Ability to use HW based packet Ordering
  - Offloaded Traffic Management

What is stopping NXP to add it's platform support in DPDK

- Inherently PCI inclined architecture



Extending DPDK for SoC support



SoC PMD: Poll Mode driver model for SoC devices

Provides a clean integration of SoC via a PMD in DPDK

- Hardware abstraction in DPDK is at the PMD layer
- DPDK-API: A generic API extended to support SoCs
  - DPDK provides a two layer device model to support many devices at the same time/binary, which can include SoC devices
  - Need to enhance DPDK with some SoC specific needs or features to support SoC hardware
    - Non-PCI configuration
    - External memory manager(s) (for hardware based memory)
    - Event based programming model
- SoC-PMD: Poll Mode Driver model for SoC
  - Allows SoC SDK's to remain private
- Supports ARM and MIPS DPDK ports to utilize these SoC designs

Source: DPDK SF Summit 2015: **"Future Enhancements to DPDK Framework"** by Keith Wiles, Principal Engineer, Intel Corporation

SoC support- Status check

- Run time services for non-IA.
  - ☺ Available for ARM, Power8 and other architecture
- Mempool offload framework – to use external or hardware memory managers
  - ☺ Merged in 16.07
- Re-org of VFIO framework support- Allow Platform bus support
  - ☺ Merged in 16.07
- HW Accelerator support
  - ☺ Rte_cryptodev framework supports SEC HW
- non-PCIe devices support
  - Multiple discussions, patch-sets – slow progress.
- Event Driven Programming model
  - Not yet. RFC APIs are now available.

Re-factoring of device framework for SoC support

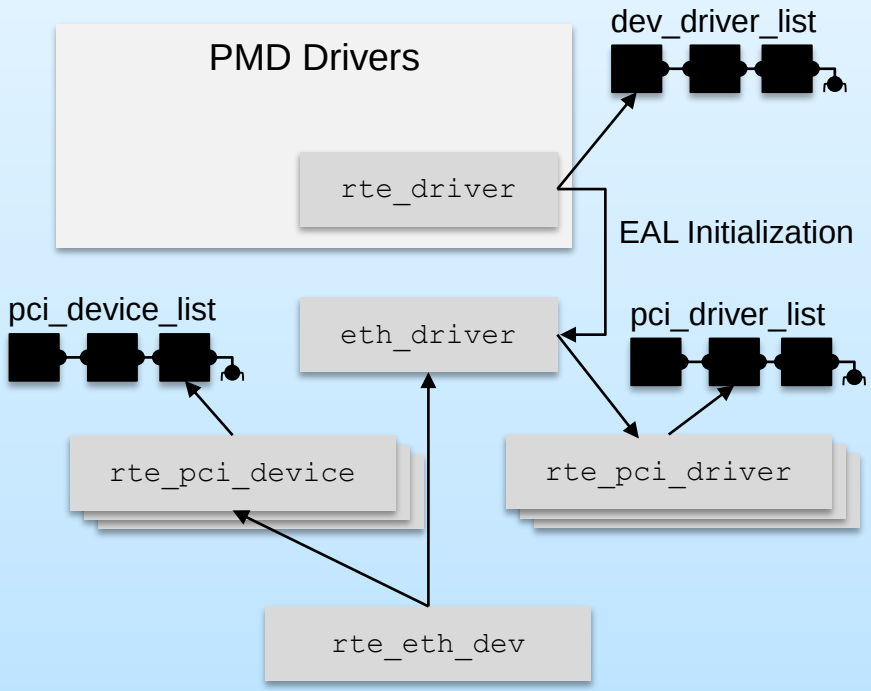
- Patch Set #1 - Generalizing the driver-device relationship
  - *Prepare for rte_device / rte_driver* <David M/Shreyansh>
    - removal of eth/crypto driver registration callbacks.
    - pdev -> PCI registration using helpers
    - rte_device=>pci/vdev device hierarchy
    - removal of PMD_PDEV type and PMD_VDEV
    - Start to support for hotplugging
  - Originally proposed by David in Jan'16, now at version 7.
  - 17 Patch series < http://dpdk.org/ml/archives/dev/2016-August/044941.html>

- Patch Set #2 - rte_driver/device infrastructure
  - *rte_driver/device infrastructure* <Jan Viktorin>
    - pmd_type is removed
    - introduced rte_vdev_driver inheriting rte_driver
    - PMD_REGISTER_DRIVER is replaced by RTE_EAL_VDRV_REGISTER(or, DRIVER_REGISTER_xxx)
    - rte_driver/device integrated into rte_pci_driver/device
    - all drivers and devices are in 2 lists - general and bus-specific
  - 15 patch series <http://dpdk.org/ml/archives/dev/2016-July/043645.html>

- Patch Set #3 - Support non-PCI devices
  - *Support non-PCI devices* <Jan Viktorin>
  - Introducing SoC driver support
  - 28-15 patches <http://www.dpdk.org/ml/archives/dev/2016-May/038486.html>

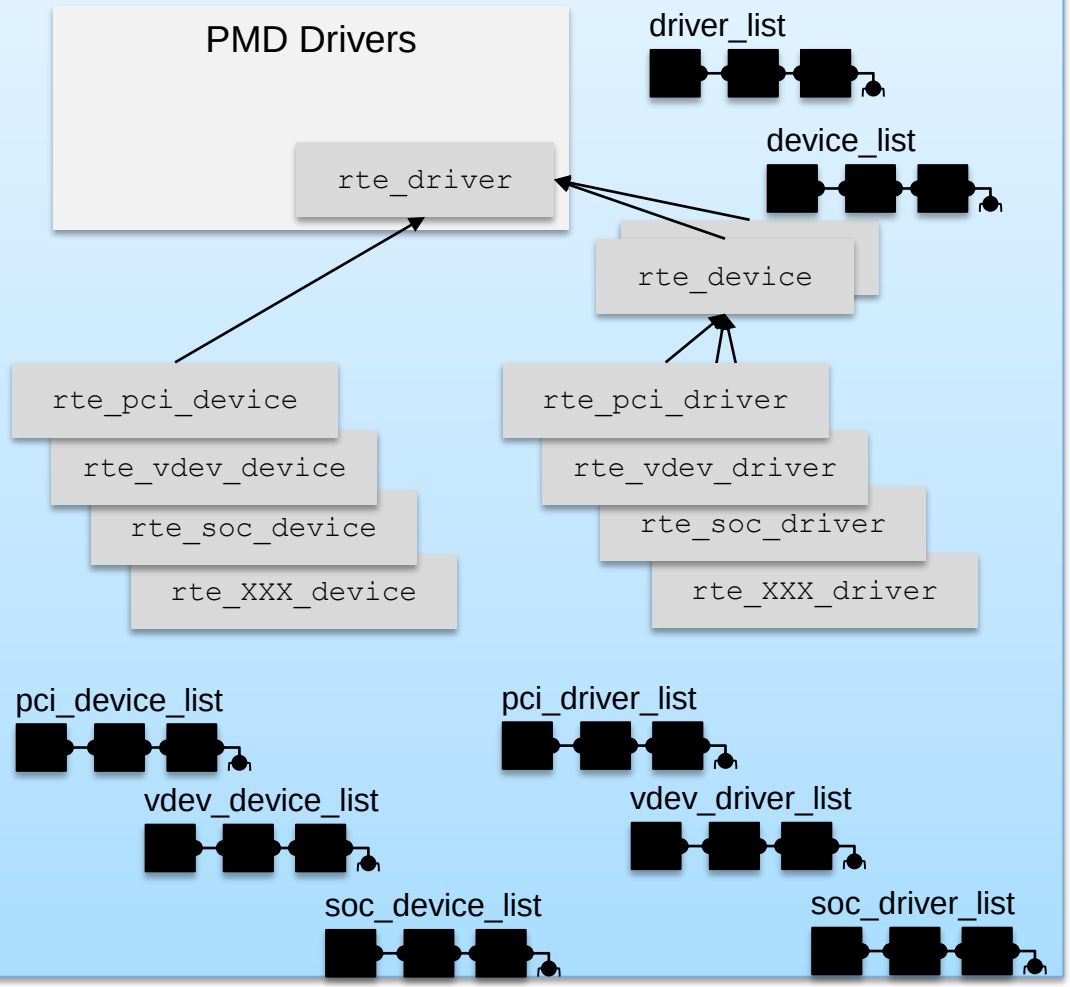
Device/Driver Structure

Existing structure



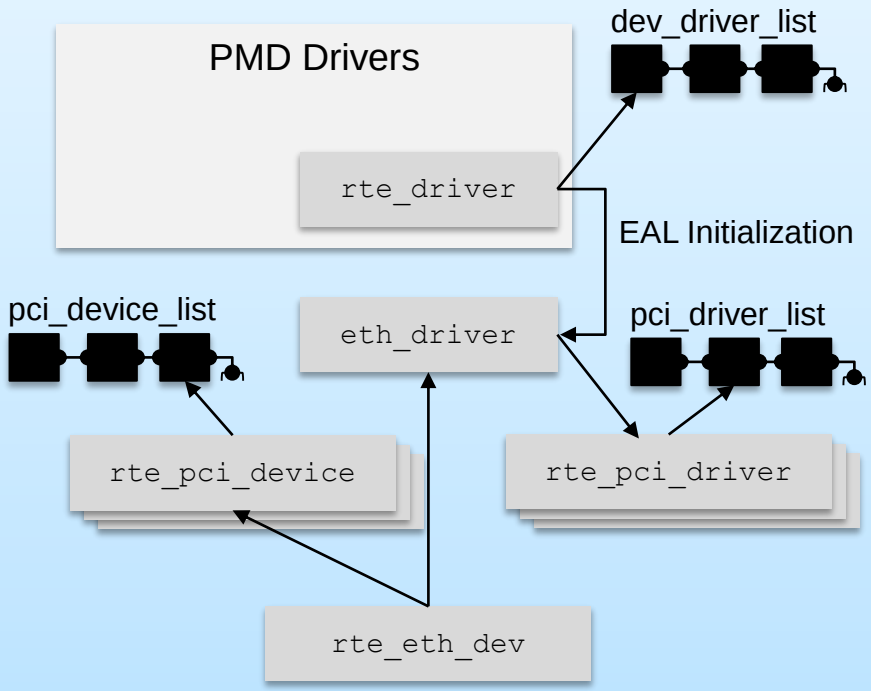
- Virtual devices are also represented by a type of rte_driver (PMD_VDEV) and treated as PCI devices
- No space for non-PCI/non-vdevices

Proposed changes in this structure



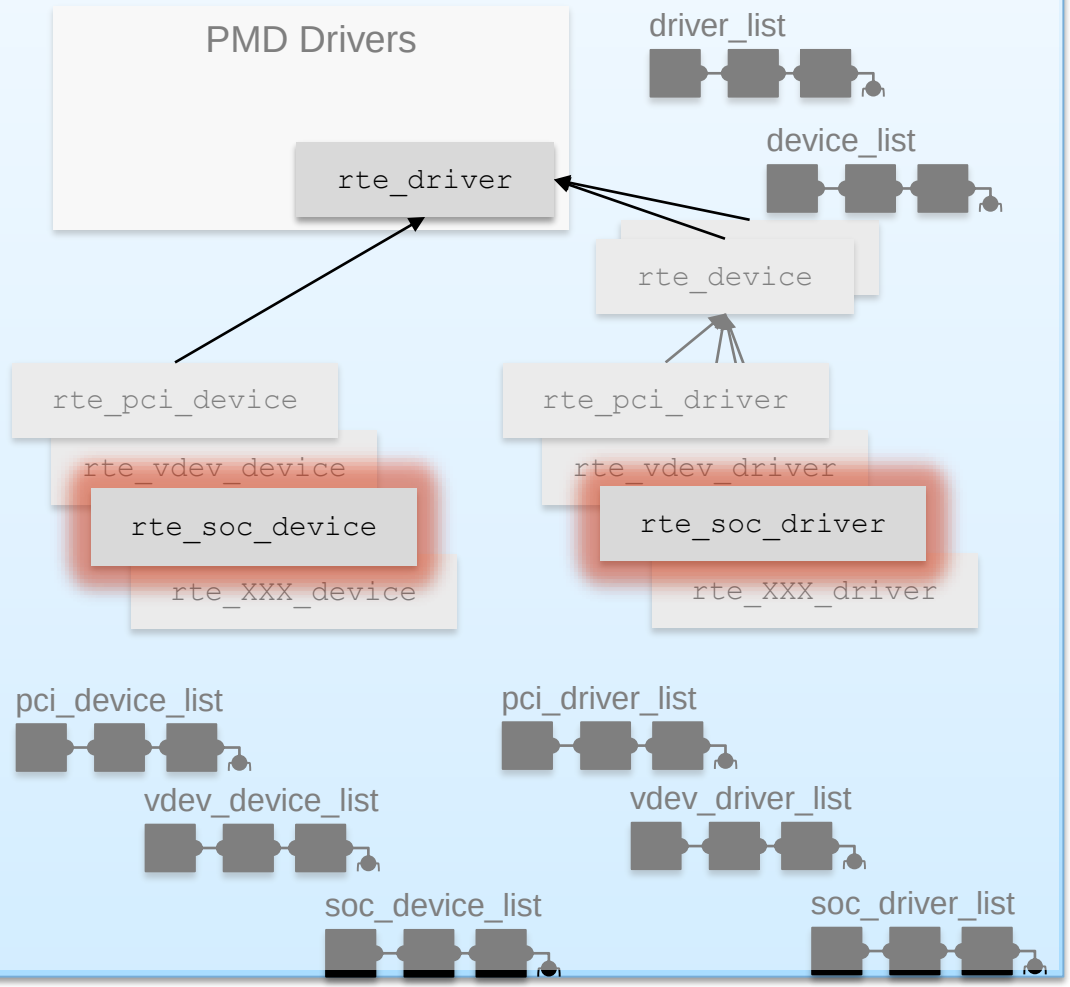
Device/Driver Structure

Existing structure



- Virtual devices are also represented by a type of rte_driver (PMD_VDEV) and treated as PCI devices
- No space for non-PCI/non-vdevices

Proposed changes in this structure



Device/Driver Structure

- Simpler Device ⇔ Driver hierarchy
  - rte_driver/rte_device represents a generic driver/device, equivalent to a Bus
  - rte_pci_driver/rte_pci_device represent an instantiation of rte_driver/rte_device
    - Same is case for vdev, SoC, XXX
  - Hotplug support
    - Devices attach and detach functions are responsibility of a ‘bus’ – rte_device/rte_driver
- Open Points:
  - How to represent resources – BDF format is too PCI specific; Soc’s may have different ways.
  - Devargs to be restructured
    - How should a device be identified (blacklist/whitelist)

Enabling SoC Support

- Jan Viktorin has proposed series of patches [1] which:
  - Generalization of `sysfs` parsing routines – majorly movement into EAL common area
  - Introducing `rte_soc_driver, rte_soc_device` (and other internal structures)
    - It is parallel to rte_driver, eth_driver so need for generalization of device<->driver ABI/API.
  - SoC registration and de-registration methods and their invocation from `rte_eal_init()`
    - Maintaining new linked-lists for SoC devices/drivers (`soc_driver_list, soc_device_list`)
  - Scanning for SoC specs using `udev`
    - SoC devices are quite varied – a one-size-fits-all approach might not work.
    - NXP devices are based on Platform driver and their initialization sequences are different
    - Need for a generic series of init/deinit for devices which can be adapted for PCI/non-PCI devices seamlessly.
  - Duplicating various PCI operations (attach/detach/devargs parsing) to suit SoC needs

[1] http://dpdk.org/ml/archives/dev/2016-May/038486.html

VFIO Changes

- DPDK VFIO Mapping doesn't support platform bus
  - NXP's DPAA is a Platform Bus design
    - Resources are not directly discoverable and cannot be 'scanned'
    - Structures associated with devices need to be maintained across the life-time of PMD
  - Current design is PCI dependent
    - Assumes that devices are discoverable
      - rte_eal_init()=>rte_eal_pci_probe()=>rte_eal_pci_probe_all/one_driver()=>rte_eal_pci_map_device()
      - rte_eal_pci_map_device() finds the container, connects the group to it and performs DMA Mapping
    - In case of NXP DPAA:
      - Once container⇔group are connected, devices within the group need to be scanned
      - These devices are used for interfacing with the hardware by PMD for RX/TX
- Need to integrate Platform Bus with device probing
  - When rte_eal_SoC_probe*() is done, post container⇔group mapping, callbacks for SoC device mapping can be done
  - These callbacks can be part of the rte_driver/rte_soc_driver structure

What else can be improved:

- External Memory Pool support
  - API sets which enable seamless usage without any confusion
    - rte_pktmbuf_pool_create can use an external mempool, but rte_mempool_create cannot
    - rte_pktmbuf_pool_create cannot iterate over objects, but rte_mempool_create can
    - So, for external mempool support, application looses control:
      - 1) Create New API or Modify rte_pktmbuf_pool_create(with..iterator..)
      - 2) Make external pool support uniform. i.e. deprecate *rte_mempool_create* or augment it with mempool name.

- Crypto – generalizing the cryptodev framework to support different kind of hw crypto interface
- Support for HW offload to Distributer/re-order, QoS and other libraries.

Questions ?

hemant.agrawal@nxp.com

SECURE CONNECTIONS
FOR A SMARTER WORLD