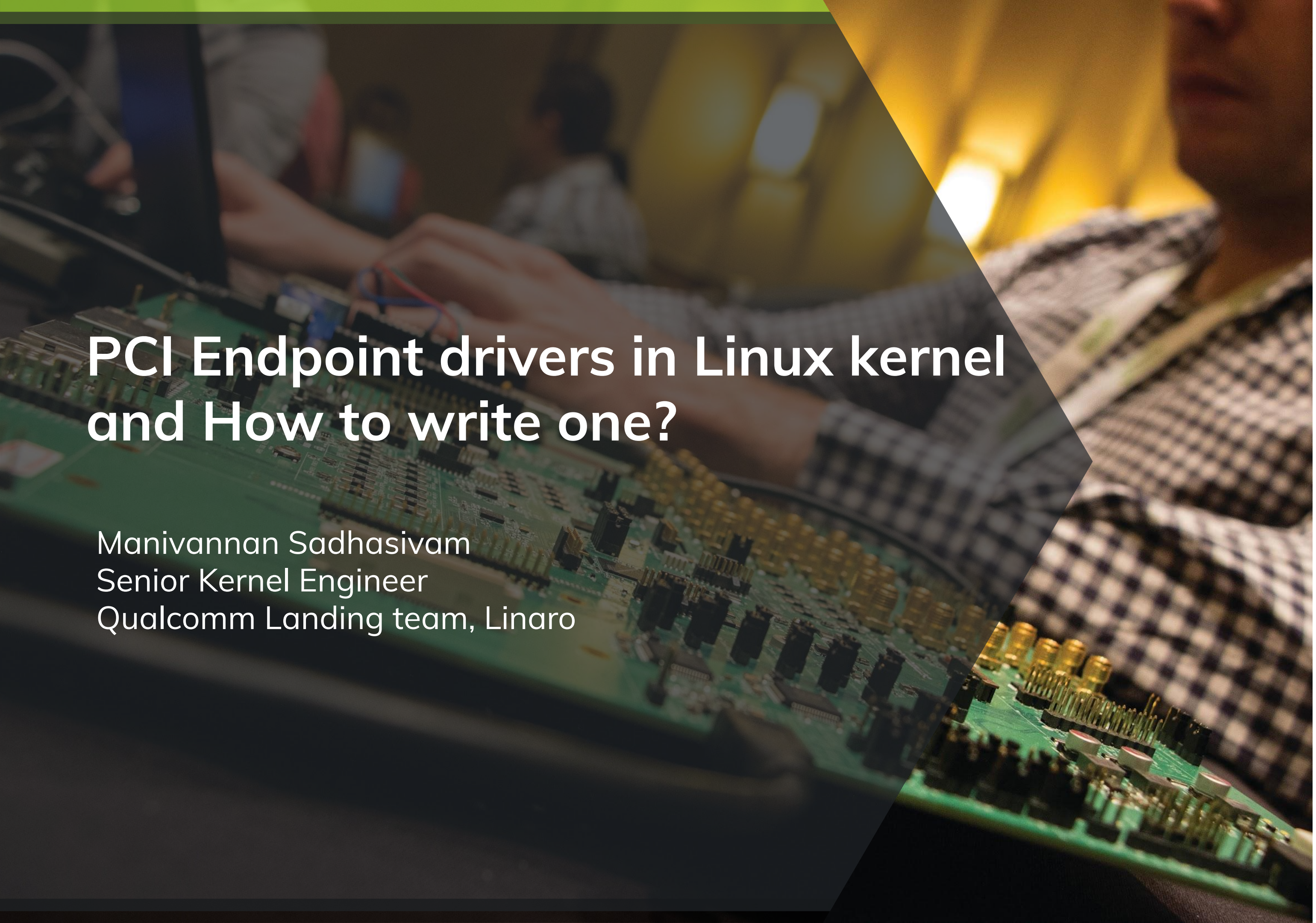


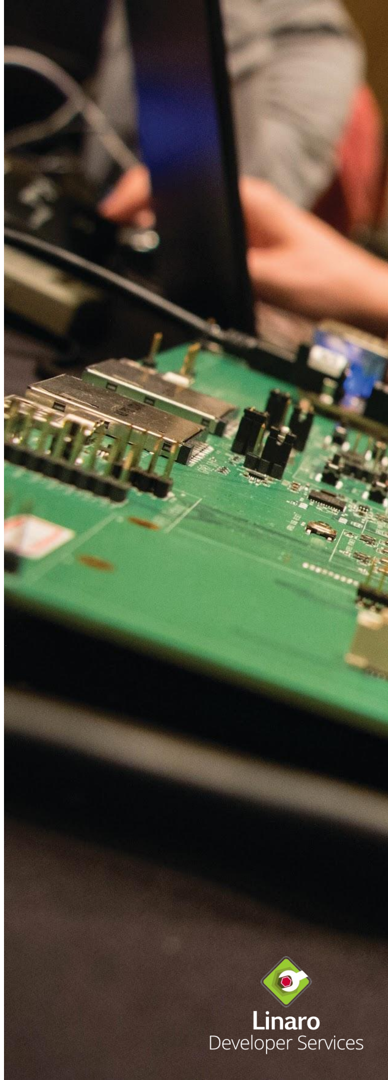
PCI Endpoint drivers in Linux kernel and How to write one?

Manivannan Sadhasivam
Senior Kernel Engineer
Qualcomm Landing team, Linaro

Linaro
Developer Services

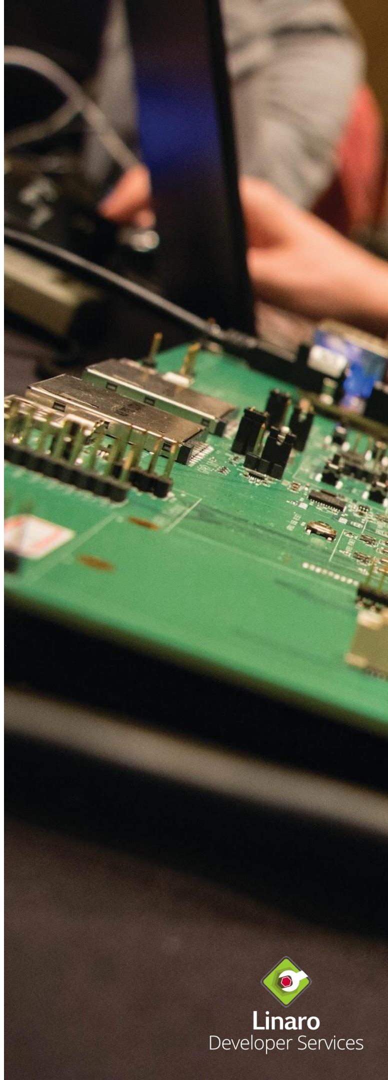
Who am I?

- Senior Kernel Engineer - Qualcomm Landing Team, Linaro
- Open Source Contributions
  - Linux Kernel
    - Maintainer of Bitmain, RDA Micro SoCs
    - Co-Maintainer of Actions Semi SoCs
    - Maintainer of MHI bus and several Qualcomm drivers
  - U-Boot
    - Maintainer of Actions Semi SoCs
    - Co-Maintainer of HiSilicon SoCs
  - Zephyr
    - Maintainer of LED, LoRa, and LoRaWAN
- Living in Tamilnadu, the southern most state of India



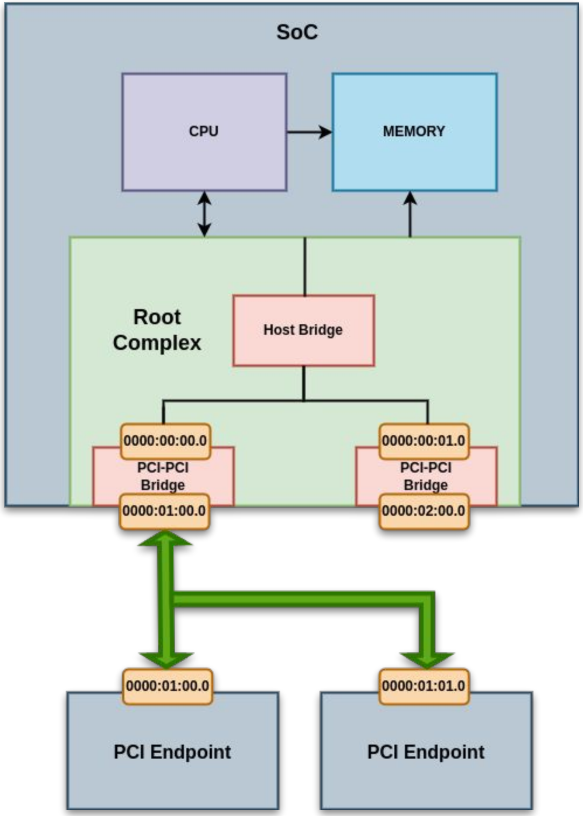
Agenda

- PCI Subsystem
  - Architecture
- PCI Endpoint
  - Architecture
- PCI Endpoint Framework
  - Internals
- PCI Endpoint Controller
  - Writing a PCI Endpoint Controller driver
- PCI Endpoint Function
  - Writing a PCI Endpoint Function driver
- Using the PCI Endpoint Framework
- Productizing the PCI Endpoint Framework
  - Pain points



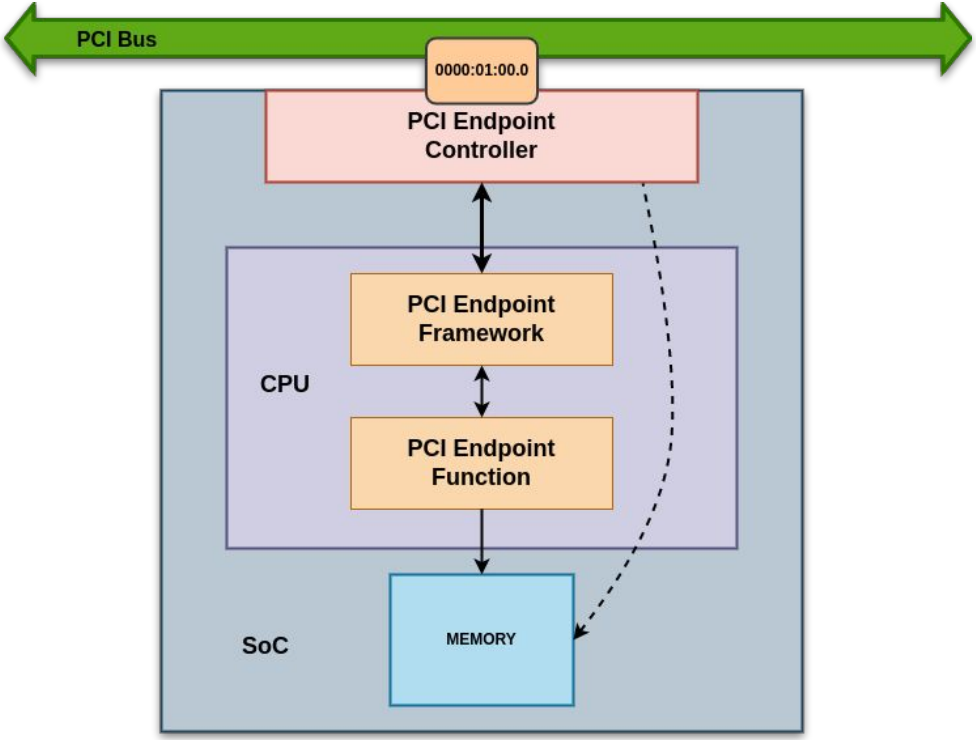
Linaro
Developer Services

PCI Subsystem



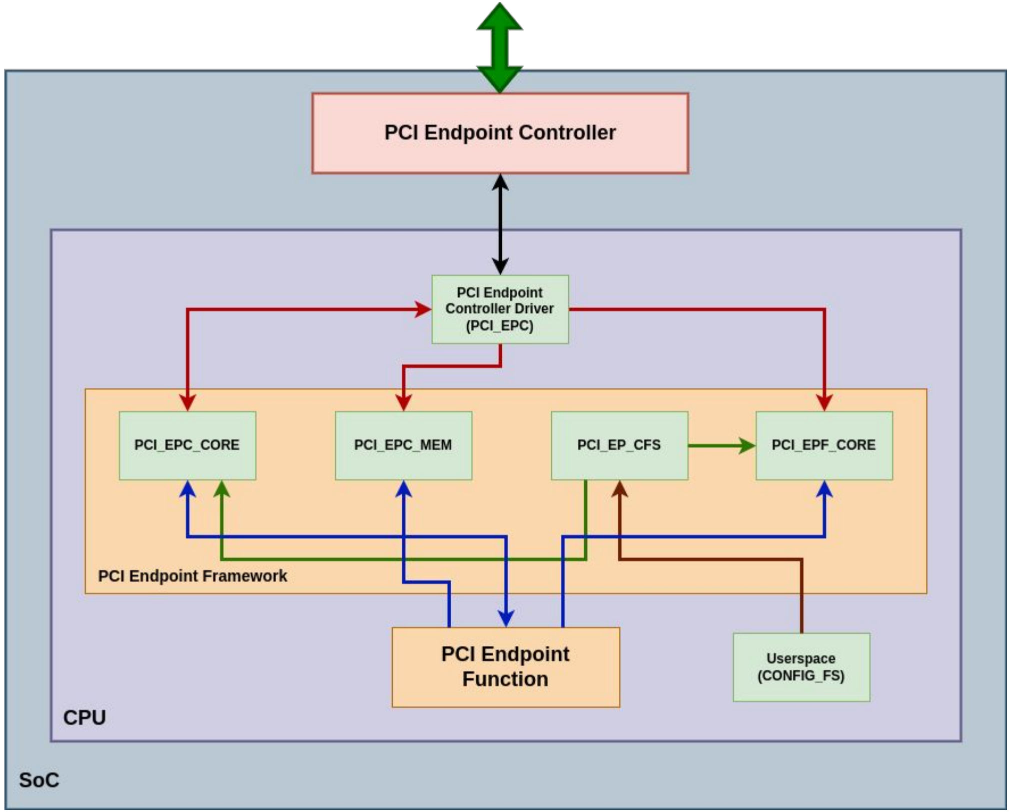
Linaro Developer Services

PCI Endpoint



PCI Endpoint

PCI Endpoint Framework



PCI Endpoint Framework

- PCI_EPC_CORE
  - Manages the interaction between EPF (Endpoint Function) and EPC (Endpoint Controller) drivers
    - drivers/pci/endpoint/pci-epc-core.c
    - include/linux/pci-epc.h
  - Passes the PCI Endpoint events from EPC to EPF
    - CORE_INIT
    - LINK_UP
  - Manages the EPC Class and EPC devices
    - /sys/class/pci_epc/<epc>/

PCI Endpoint Framework

- PCI_EPC_MEM
  - Manages the memory space used by the PCI Endpoint Function
    - drivers/pci/endpoint/pci-epc-mem.c
  - Allocates memory from the "addr_space" region specified in the PCI Endpoint controller devicetree node
    - pci_epc_mem_alloc_addr()
    - pci_epc_mem_free_addr()
  - Allocated memory can be used for mapping the PCI host address space
    - An external entity like iATU (Internal Address Translation Unit) can be used to map the PCI host memory
  - PCI host memory mapping is used for several purposes:
    - Message Signal Interrupt (MSI) Generation to PCI host
    - Reading/Writing arbitrary PCI host memory

PCI Endpoint Framework

- PCI_EPF_CORE
  - Manages the interaction between CFS (ConfigFS) filesystem and EPF drivers
    - drivers/pci/endpoint/pci-epf-core.c
  - Controls the creation and deletion of EPF drivers
    - pci_epf_register_driver() / pci_epf_unregister_driver()
    - pci_epf_bind() / pci_epf_unbind()
  - Allocates memory in the PCI Endpoint BAR region for EPF drivers
    - EPF drivers could use the allocated memory for simulating virtual PCI Endpoint register set to be used by PCI host
    - pci_epf_alloc_space()
    - pci_epf_free_space()

PCI Endpoint Framework

- PCI_EP_CFS
  - Manages the interaction with userspace through ConfigFS filesystem
    - drivers/pci/endpoint/pci-ep-cfs.c
  - Userspace interaction includes:
    - Creation and deletion of Endpoint functions for EPF drivers
    - Binding EPF drivers with EPC devices
    - Starting and stopping EPCs

Writing a PCI Endpoint Controller Driver

- PROBE
  - Initialize Endpoint Controller
  - Initialize DMA Engine (optional)
  - Allocate memory for MSI
  - Setup PCI host memory mapping
  - Enable Endpoint IRQs
  - Create PCI EPC device
- IRQ_HANDLER
  - PERST# (optional)
  - LINK_UP

Initialize Endpoint Controller

- Initialize resources such as clocks, reset, PHY and GPIO
- Memory regions
  - dbi
    - Direct bus interface (DBI) - Synopsys Designware Specific
  - addr_space / mem
    - Endpoint address space for mapping PCI host memory
  - atu
    - Internal Address Translation Unit (iATU) - Synopsys Designware Specific
  - dma (Optional)
- Endpoint Controller Configuration
  - Endpoint mode
  - Link Speed and Lane Count
  - L1/L1ss
  - Link Training and Status State Machine (LTSSM)

Initialize Endpoint Controller Contd...

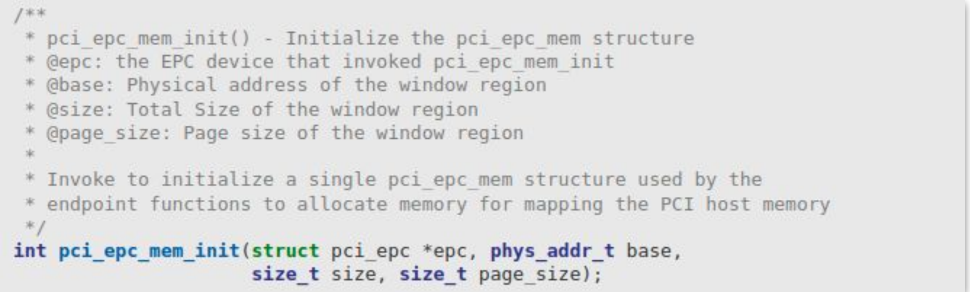
- Notifiers
  - CORE_INIT_NOTIFIER
    - If the Endpoint Controller depends on the active Reference Clock (refclk) from the host, then the Controller initialization could be deferred to later stage when refclk becomes active
      - Set core_init_notifier flag available in pci_epc_features struct
      - Once the refclk becomes active, initialize the Endpoint Controller and call dw_pcie_ep_init_notify() to notify EPF that the Controller has completed initialization
  - LINKUP_NOTIFIER
    - Since the LINK UP event can happen at any point of time during runtime, LINKUP_NOTIFIER could be used to notify the EPF of the event
      - Set linkup_notifier flag available in pci_epc_features struct
      - When the LINK UP event is received, call dw_pcie_ep_linkup() to notify EPF about the event

Initialize DMA Engine (optional)

- Setup READ/WRITE channels
- Request DMA IRQs
- Allocate and configure Linked Lists (LL)
- Configure DMA Controller

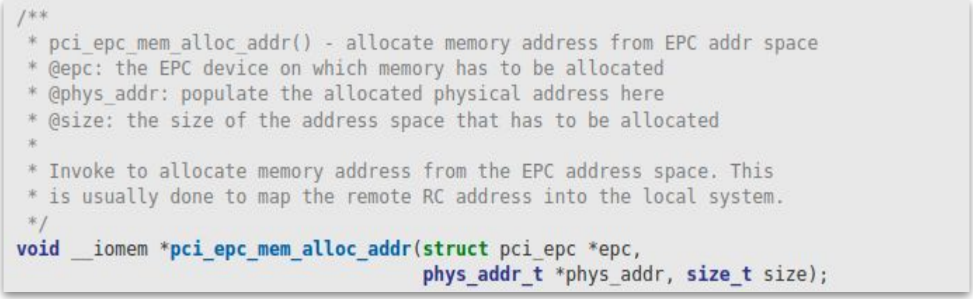
Allocate memory for MSI

- Initialize memory for Message Signalled Interrupts (MSI)



```
/**
 * pci_epc_mem_init() - Initialize the pci_epc_mem structure
 * @epc: the EPC device that invoked pci_epc_mem_init
 * @base: Physical address of the window region
 * @size: Total Size of the window region
 * @page_size: Page size of the window region
 *
 * Invoke to initialize a single pci_epc_mem structure used by the
 * endpoint functions to allocate memory for mapping the PCI host memory
 */
int pci_epc_mem_init(struct pci_epc *epc, phys_addr_t base,
                     size_t size, size_t page_size);
```

- Allocate memory for MSI in Endpoint address space



```
/**
 * pci_epc_mem_alloc_addr() - allocate memory address from EPC addr space
 * @epc: the EPC device on which memory has to be allocated
 * @phys_addr: populate the allocated physical address here
 * @size: the size of the address space that has to be allocated
 *
 * Invoke to allocate memory address from the EPC address space. This
 * is usually done to map the remote RC address into the local system.
 */
void __iomem *pci_epc_mem_alloc_addr(struct pci_epc *epc,
                                     phys_addr_t *phys_addr, size_t size);
```

Linaro
Developer Services

Setup PCI host memory mapping

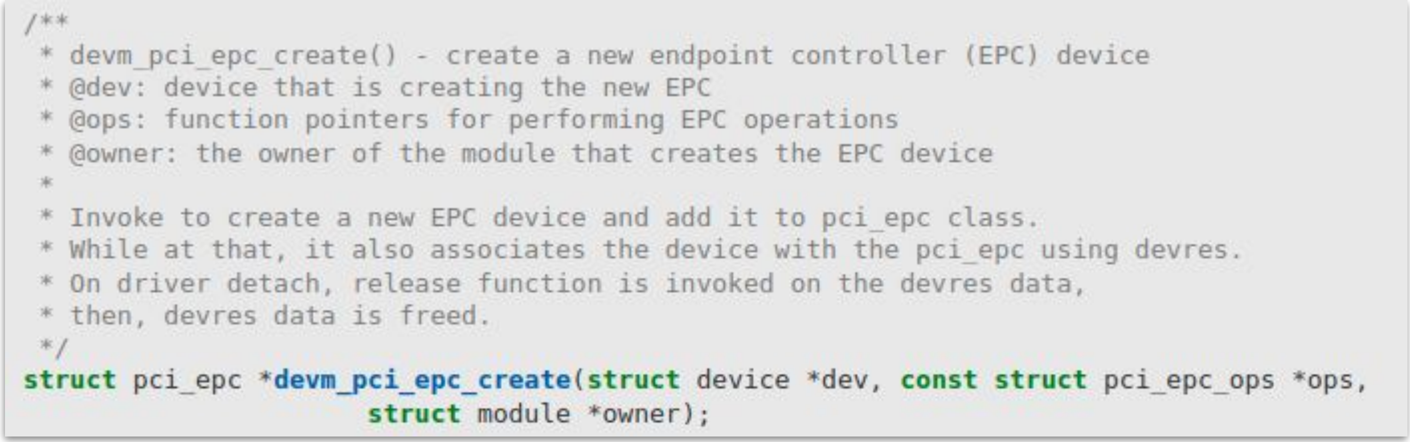
- Detect and initialize the memory mapping block like iATU* for mapping PCI host memory
- Setup the mapping windows to be used during runtime
- Setup the memory alignment and limit

Enable Endpoint IRQs

- Enable the Endpoint Controller related IRQs if supported
  - PERST#
    - Sideband GPIO for receiving the PERST IRQ from PCI host
  - Any other controller specific IRQ for handling the Link/Controller specific events

Create PCI EPC device

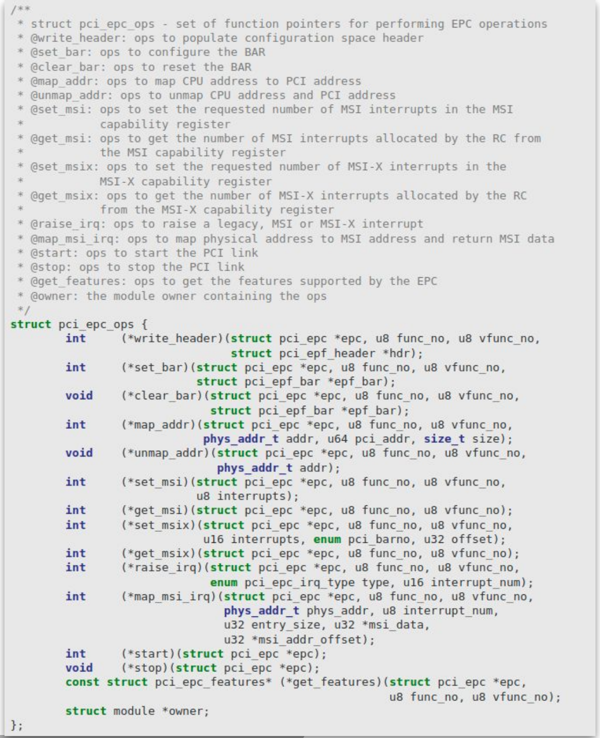
- Once all of the initializations are done, create the PCI EPC device



```
/**
 * devm_pci_epc_create() - create a new endpoint controller (EPC) device
 * @dev: device that is creating the new EPC
 * @ops: function pointers for performing EPC operations
 * @owner: the owner of the module that creates the EPC device
 *
 * Invoke to create a new EPC device and add it to pci_epc class.
 * While at that, it also associates the device with the pci_epc using devres.
 * On driver detach, release function is invoked on the devres data,
 * then, devres data is freed.
 */
struct pci_epc *devm_pci_epc_create(struct device *dev, const struct pci_epc_ops *ops,
                      struct module *owner);
```

Create PCI EPC device Contd...

- Pass the functions pointers required for the EPC device operation



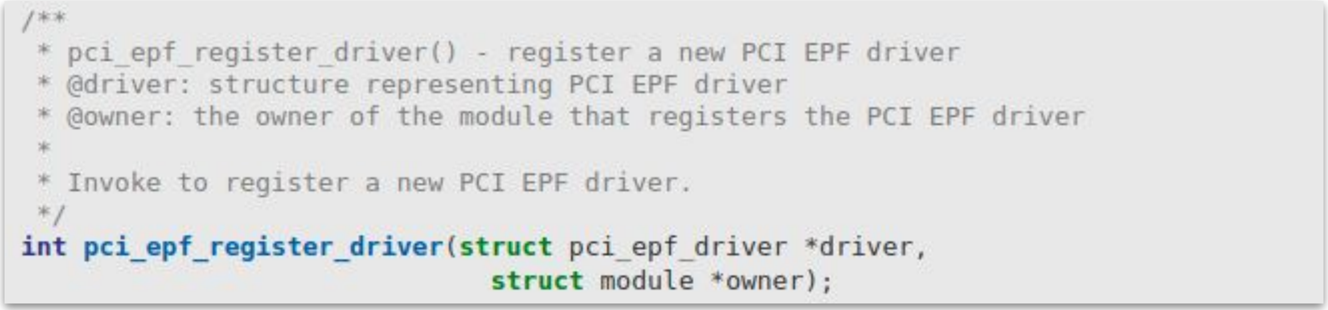
```c
/**
 * struct pci_epc_ops - set of function pointers for performing EPC operations
 * @write_header: ops to populate configuration space header
 * @set_bar: ops to configure the BAR
 * @clear_bar: ops to reset the BAR
 * @map_addr: ops to map CPU address to PCI address
 * @unmap_addr: ops to unmap CPU address and PCI address
 * @set_msi: ops to set the requested number of MSI interrupts in the MSI
 *           capability register
 * @get_msi: ops to get the number of MSI interrupts allocated by the RC from
 *           the MSI capability register
 * @set_msix: ops to set the requested number of MSI-X interrupts in the
 *           MSI-X capability register
 * @get_msix: ops to get the number of MSI-X interrupts allocated by the RC
 *           from the MSI-X capability register
 * @raise_irq: ops to raise a legacy, MSI or MSI-X interrupt
 * @map_msi_irq: ops to map physical address to MSI address and return MSI data
 * @start: ops to start the PCI link
 * @stop: ops to stop the PCI link
 * @get_features: ops to get the features supported by the EPC
 * @owner: the module owner containing the ops
 */
struct pci_epc_ops {
        int     (*write_header)(struct pci_epc *epc, u8 func_no, u8 vfunc_no,
                                struct pci_epf_header *hdr);
        int     (*set_bar)(struct pci_epc *epc, u8 func_no, u8 vfunc_no,
                           struct pci_epf_bar *epf_bar);
        void    (*clear_bar)(struct pci_epc *epc, u8 func_no, u8 vfunc_no,
                             struct pci_epf_bar *epf_bar);
        int     (*map_addr)(struct pci_epc *epc, u8 func_no, u8 vfunc_no,
                            phys_addr_t addr, u64 pci_addr, size_t size);
        void    (*unmap_addr)(struct pci_epc *epc, u8 func_no, u8 vfunc_no,
                              phys_addr_t addr);
        int     (*set_msi)(struct pci_epc *epc, u8 func_no, u8 vfunc_no,
                           u8 interrupts);
        int     (*get_msi)(struct pci_epc *epc, u8 func_no, u8 vfunc_no);
        int     (*set_msix)(struct pci_epc *epc, u8 func_no, u8 vfunc_no,
                            u16 interrupts, enum pci_barno, u32 offset);
        int     (*get_msix)(struct pci_epc *epc, u8 func_no, u8 vfunc_no);
        int     (*raise_irq)(struct pci_epc *epc, u8 func_no, u8 vfunc_no,
                             enum pci_epc_irq_type type, u16 interrupt_num);
        int     (*map_msi_irq)(struct pci_epc *epc, u8 func_no, u8 vfunc_no,
                               phys_addr_t phys_addr, u8 interrupt_num,
                               u32 entry_size, u32 *msi_data,
                               u32 *msi_addr_offset);
        int     (*start)(struct pci_epc *epc);
        void    (*stop)(struct pci_epc *epc);
        const struct pci_epc_features* (*get_features)(struct pci_epc *epc,
                                                       u8 func_no, u8 vfunc_no);
        struct module *owner;
};
```

Writing a PCI Endpoint Function Driver

- MODULE_INIT / MODULE_EXIT
- Service EPF notifications

MODULE_INIT

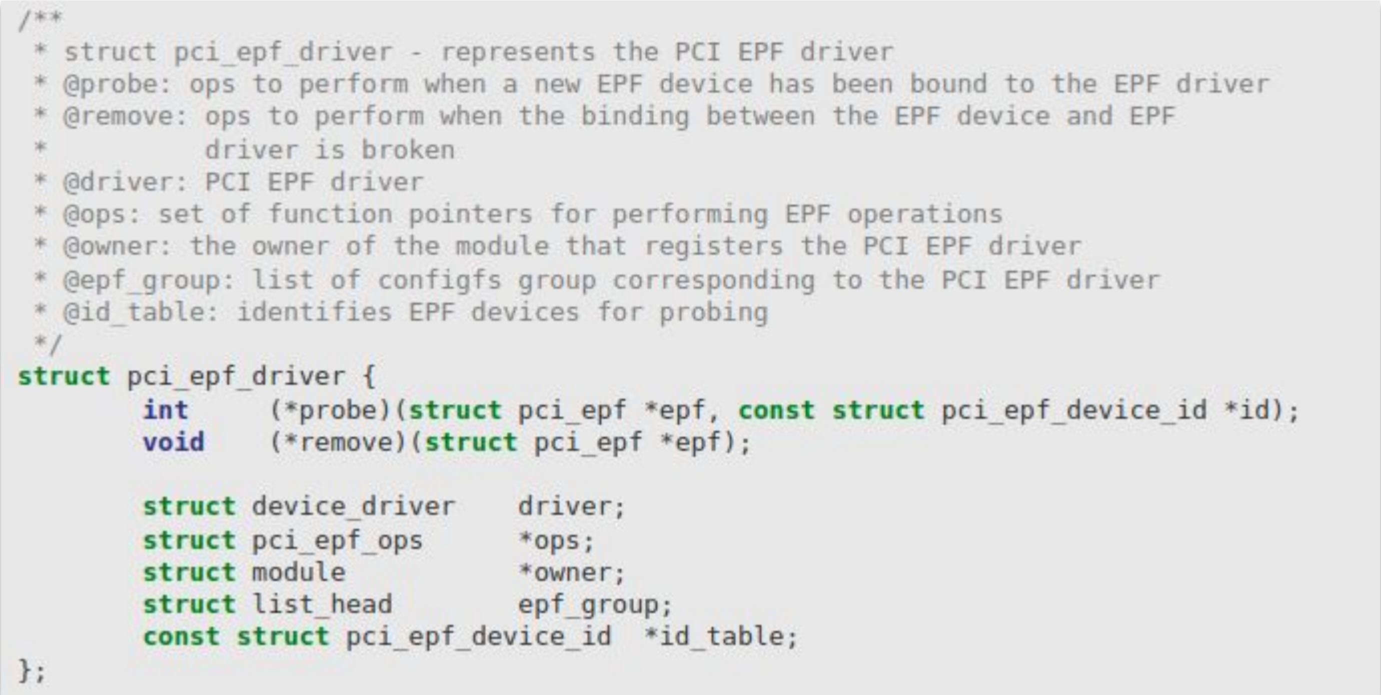
- Register the EPF driver with Endpoint Framework



```
/**
 * pci_epf_register_driver() - register a new PCI EPF driver
 * @driver: structure representing PCI EPF driver
 * @owner: the owner of the module that registers the PCI EPF driver
 *
 * Invoke to register a new PCI EPF driver.
 */
int pci_epf_register_driver(struct pci_epf_driver *driver,
                              struct module *owner);
```

MODULE_INIT Contd...

- Populate pci_epf_driver struct

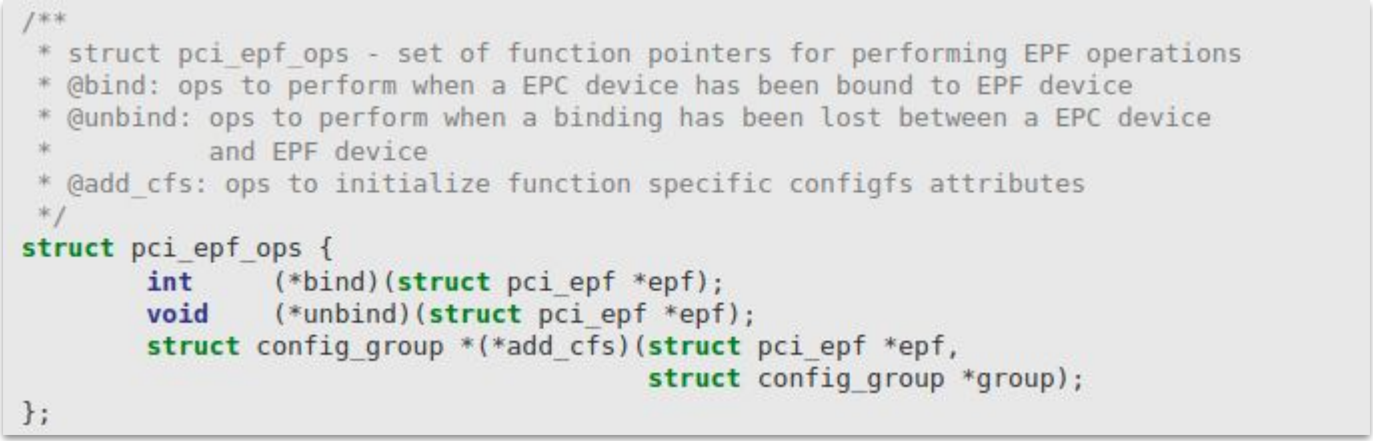


```
/**
 * struct pci_epf_driver - represents the PCI EPF driver
 * @probe: ops to perform when a new EPF device has been bound to the EPF driver
 * @remove: ops to perform when the binding between the EPF device and EPF
 *          driver is broken
 * @driver: PCI EPF driver
 * @ops: set of function pointers for performing EPF operations
 * @owner: the owner of the module that registers the PCI EPF driver
 * @epf_group: list of configfs group corresponding to the PCI EPF driver
 * @id_table: identifies EPF devices for probing
 */
struct pci_epf_driver {
        int     (*probe)(struct pci_epf *epf, const struct pci_epf_device_id *id);
        void    (*remove)(struct pci_epf *epf);

        struct device_driver    driver;
        struct pci_epf_ops      *ops;
        struct module           *owner;
        struct list_head        epf_group;
        const struct pci_epf_device_id  *id_table;
};
```

MODULE_INIT Contd...

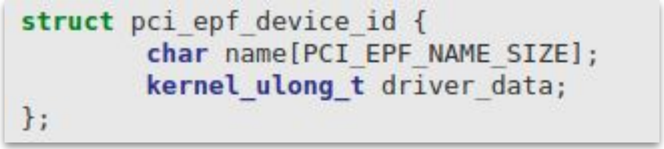
- Assign function callbacks to pci_epf_ops



```
/**
 * struct pci_epf_ops - set of function pointers for performing EPF operations
 * @bind: ops to perform when a EPC device has been bound to EPF device
 * @unbind: ops to perform when a binding has been lost between a EPC device
 *          and EPF device
 * @add_cfs: ops to initialize function specific configfs attributes
 */
struct pci_epf_ops {
        int     (*bind)(struct pci_epf *epf);
        void    (*unbind)(struct pci_epf *epf);
        struct config_group *(*add_cfs)(struct pci_epf *epf,
                                        struct config_group *group);
};
```

MODULE_INIT Contd...

- Populate pci_epf_device_id struct



```
struct pci_epf_device_id {
        char name[PCI_EPF_NAME_SIZE];
        kernel_ulong_t driver_data;
};
```

    - name is the EPF driver name and driver_data is an opaque pointer

Service EPF notifications

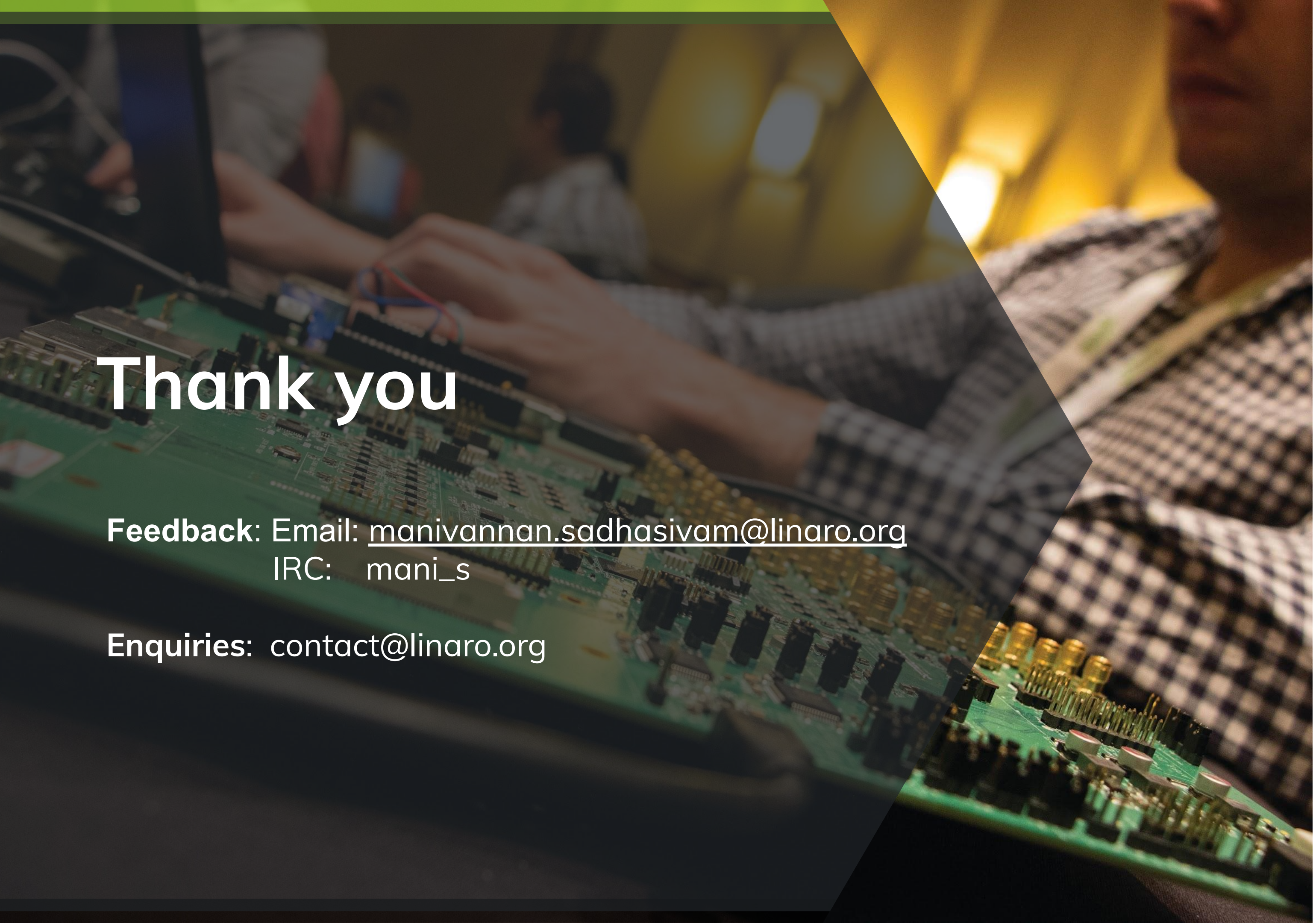
- If core_init_notifier is supported, then on the occurrence of the event:
  - Write PCI EPC header using pci_epc_write_header()
  - Set PCI BARs using pci_epc_set_bar()
  - Set MSI/MSIx using pci_epc_set_msi() and pci_epc_set_msix()
- If linkup_notifier is supported, then on the occurrence of the event:
  - Request DMA channels using dma_request_channel()
  - Start the actual function of the EPF driver

Using the PCI Endpoint Framework

- Boot the PCI host and PCI Endpoint devices
- Load the Endpoint Controller and Endpoint Function drivers
- Mount the ConfigFS filesystem
  - mount -t configfs none /sys/kernel/config
- Create the Endpoint Function device
  - mkdir /sys/kernel/config/functions/<epf>/func1
- Bind the Endpoint Function driver with Endpoint Controller
  - ln -s /sys/kernel/config/functions/<epf>/func1 /sys/kernel/config/controllers/<epc>/
- Start the link
  - echo 1 > /sys/kernel/config/controllers/<epc>/start
- Stop the link
  - echo 0 > /sys/kernel/config/controllers/<epc>/start

Productizing the PCI Endpoint Framework

- Pain points
  - The probe of PCI Endpoint Controller driver depends on the active Reference clock from the host
  - No way to configure the Endpoint Framework in kernel without ConfigFS
  - No devicetree integration in Endpoint Framework
  - Use of Notifiers forces atomic context in EPF drivers



Thank you

Feedback: Email: manivannan.sadhasivam@linaro.org
IRC: mani_s

Enquiries: contact@linaro.org

Linaro
Developer Services