Linux Peci Documentation

The kernel development community

Jan 15, 2023

CONTENTS

OVERVIEW

The Platform Environment Control Interface (PECI) is a communication interface between Intel processor and management controllers (e.g. Baseboard Management Controller, BMC). PECI provides services that allow the management controller to configure, monitor and debug platform by accessing various registers. It defines a dedicated command protocol, where the management controller is acting as a PECI originator and the processor - as a PECI responder. PECI can be used in both single processor and multiple-processor based systems.

NOTE: Intel PECI specification is not released as a dedicated document, instead it is a part of External Design Specification (EDS) for given Intel CPU. External Design Specifications are usually not publicly available.

1.1 PECI Wire

PECI Wire interface uses a single wire for self-clocking and data transfer. It does not require any additional control lines - the physical layer is a self-clocked one-wire bus signal that begins each bit with a driven, rising edge from an idle near zero volts. The duration of the signal driven high allows to determine whether the bit value is logic '0' or logic '1'. PECI Wire also includes variable data rate established with every message.

For PECI Wire, each processor package will utilize unique, fixed addresses within a defined range and that address should have a fixed relationship with the processor socket ID - if one of the processors is removed, it does not affect addresses of remaining processors.

1.2 PECI subsystem internals

struct **peci_controller_ops**
     PECI controller specific methods

Definition

```
struct peci_controller_ops {
  int (*xfer)(struct peci_controller *controller, u8 addr, struct peci_request
↪*req);
};
```

Members

xfer PECI transfer function

Description

PECI controllers may have different hardware interfaces - the drivers implementing PECI controllers can use this structure to abstract away those differences by exposing a common interface for PECI core.

struct **peci_controller**
> PECI controller

Definition

```
struct peci_controller {
  struct device dev;
  struct peci_controller_ops *ops;
  struct mutex bus_lock;
  u8 id;
};
```

Members

dev device object to register PECI controller to the device model

ops pointer to device specific controller operations

bus_lock lock used to protect multiple callers

id PECI controller ID

Description

PECI controllers usually connect to their drivers using non-PECI bus, such as the platform bus. Each PECI controller can communicate with one or more PECI devices.

struct **peci_device**
> PECI device

Definition

```
struct peci_device {
  struct device dev;
  struct {
    u16 family;
    u8 model;
    u8 peci_revision;
    u8 socket_id;
  } info;
  u8 addr;
  bool deleted;
};
```

Members

dev device object to register PECI device to the device model

info PECI device characteristics

info.family device family

info.model device model

info.peci_revision PECI revision supported by the PECI device

info.socket_id the socket ID represented by the PECI device

addr address used on the PECI bus connected to the parent controller

deleted indicates that PECI device was already deleted

Description

A peci_device identifies a single device (i.e. CPU) connected to a PECI bus. The behaviour exposed to the rest of the system is defined by the PECI driver managing the device.

struct **peci_request**
    PECI request

Definition

```
struct peci_request {
  struct peci_device *device;
  struct {
    u8 buf[PECI_REQUEST_MAX_BUF_SIZE];
    u8 len;
  } rx, tx;
};
```

Members

device PECI device to which the request is sent

rx RX buffer specific data

rx.buf RX buffer

rx.len received data length in bytes

tx TX buffer specific data

tx.buf TX buffer

tx.len transfer data length in bytes

Description

A peci_request represents a request issued by PECI originator (TX) and a response received from PECI responder (RX).

struct **peci_device_id**
    PECI device data to match

Definition

```
struct peci_device_id {
  const void *data;
  u16 family;
  u8 model;
};
```

Members

data pointer to driver private data specific to device

---

family device family

model device model

struct **peci_driver**
     PECI driver

Definition

```
struct peci_driver {
  struct device_driver driver;
  int (*probe)(struct peci_device *device, const struct peci_device_id *id);
  void (*remove)(struct peci_device *device);
  const struct peci_device_id *id_table;
};
```

Members

driver inherit device driver

probe probe callback

remove remove callback

id_table PECI device match table to decide which device to bind

peci_driver_register

peci_driver_register (driver)

     register PECI driver

Parameters

driver the driver to be registered

Description

PECI drivers that don't need to do anything special in module init should use the convenience "module_peci_driver" macro instead

Return

zero on success, else a negative error code.

module_peci_driver

module_peci_driver (__peci_driver)

     helper macro for registering a modular PECI driver

Parameters

__peci_driver peci_driver struct

Description

Helper macro for PECI drivers which do not do anything special in module init/exit. This eliminates a lot of boilerplate. Each module may only use this macro once, and calling it replaces module_init() and module_exit()

struct *peci_controller* ***devm_peci_controller_add**(struct device *dev, struct
                                                        peci_controller_ops *ops)

>   add PECI controller

Parameters

struct device *dev device for devm operations

struct peci_controller_ops *ops pointer to controller specific methods

Description

In final stage of its probe(), peci_controller driver calls *devm_peci_controller_add()* to register itself with the PECI bus.

Return

Pointer to the newly allocated controller or ERR_PTR() in case of failure.

int **peci_request_status**(struct *peci_request* *req)

>   return -errno based on PECI completion code

Parameters

struct peci_request *req the PECI request that contains response data with completion code

Description

It can't be used for Ping(), GetDIB() and GetTemp() - for those commands we don't expect completion code in the response.

Return

-errno

struct *peci_request* ***peci_request_alloc**(struct *peci_device* *device, u8 tx_len, u8 rx_len)

>   allocate struct peci_requests

Parameters

struct peci_device *device PECI device to which request is going to be sent

u8 tx_len TX length

u8 rx_len RX length

Return

A pointer to a newly allocated *struct peci_request* on success or NULL otherwise.

void **peci_request_free**(struct *peci_request* *req)

>   free peci_request

Parameters

struct peci_request *req the PECI request to be freed

1.3 PECI CPU Driver API

int **peci_temp_read**(struct *peci_device* *device, s16 *temp_raw)
    read the maximum die temperature from PECI target device

Parameters

struct peci_device *device PECI device to which request is going to be sent

s16 *temp_raw where to store the read temperature

Description

It uses GetTemp PECI command.

Return

0 if succeeded, other values in case errors.

int **peci_pcs_read**(struct *peci_device* *device, u8 index, u16 param, u32 *data)
    read PCS register

Parameters

struct peci_device *device PECI device to which request is going to be sent

u8 index PCS index

u16 param PCS parameter

u32 *data where to store the read data

Description

It uses RdPkgConfig PECI command.

Return

0 if succeeded, other values in case errors.

int **peci_pci_local_read**(struct *peci_device* *device, u8 bus, u8 dev, u8 func, u16 reg, u32
                            *data)
    read 32-bit memory location using raw address

Parameters

struct peci_device *device PECI device to which request is going to be sent

u8 bus bus

u8 dev device

u8 func function

u16 reg register

u32 *data where to store the read data

Description

It uses RdPCIConfigLocal PECI command.

Return

0 if succeeded, other values in case errors.

int **peci_ep_pci_local_read**(struct *peci_device* *device, u8 seg, u8 bus, u8 dev, u8 func, u16 reg, u32 *data)

> read 32-bit memory location using raw address

Parameters

struct peci_device *device PECI device to which request is going to be sent

u8 seg PCI segment

u8 bus bus

u8 dev device

u8 func function

u16 reg register

u32 *data where to store the read data

Description

Like *peci_pci_local_read*, but it uses RdEndpointConfig PECI command.

Return

0 if succeeded, other values in case errors.

int **peci_mmio_read**(struct *peci_device* *device, u8 bar, u8 seg, u8 bus, u8 dev, u8 func, u64 address, u32 *data)

> read 32-bit memory location using 64-bit bar offset address

Parameters

struct peci_device *device PECI device to which request is going to be sent

u8 bar PCI bar

u8 seg PCI segment

u8 bus bus

u8 dev device

u8 func function

u64 address 64-bit MMIO address

u32 *data where to store the read data

Description

It uses RdEndpointConfig PECI command.

Return

0 if succeeded, other values in case errors.

D

M

P