Proposing Common PHY Framework in Zephyr

Shradha Todi, Inbaraj E and Padmanabhan Rajanbabu

SSIR Samsung Semiconductor India R&D Center

Agenda

❑ PHY- An Introduction

❑ PHY in a system

❑ Existing Mechanisms to implement PHY

❑ Need of a generic PHY framework

❑ Introduction to common PHY framework

❑ Implementation

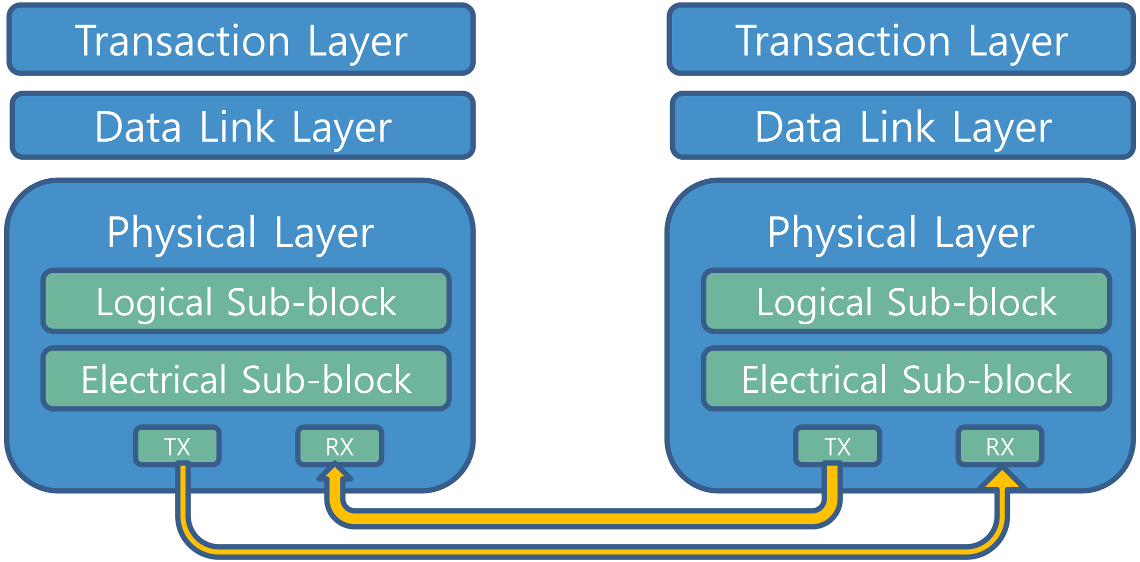
❑ How to use the PHY framework

❑ Conclusion and Future scope

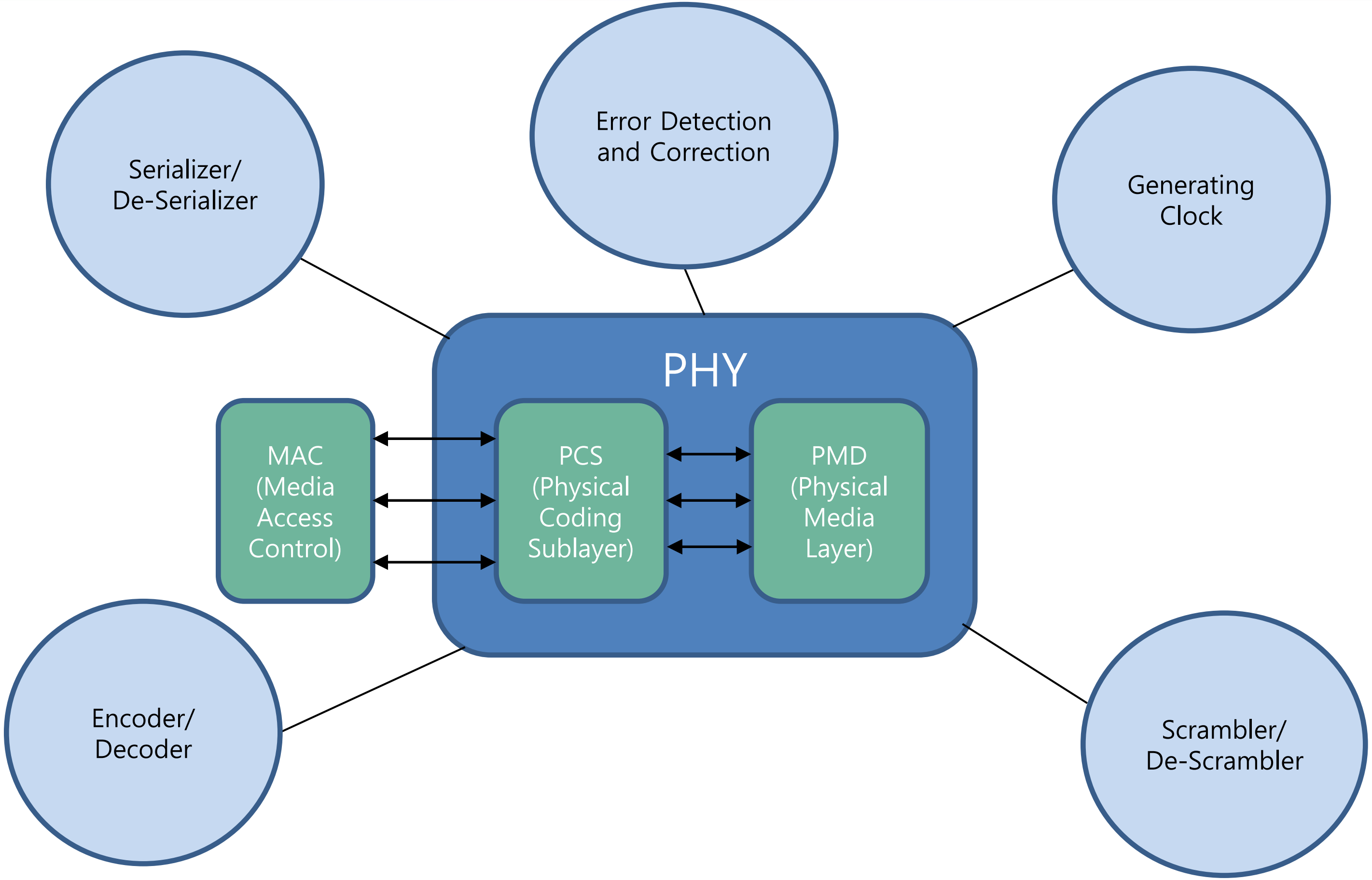
PHY: An Introduction

❑ A **PHY** (physical layer) is an electronic circuit, required to implement physical layer functions of the OSI model

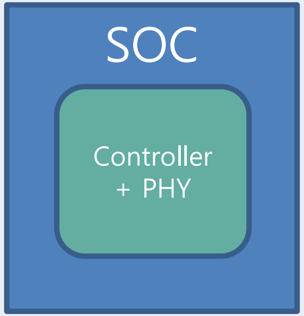
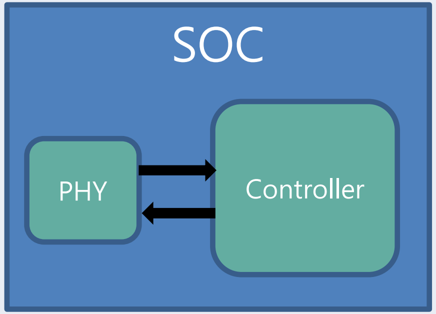
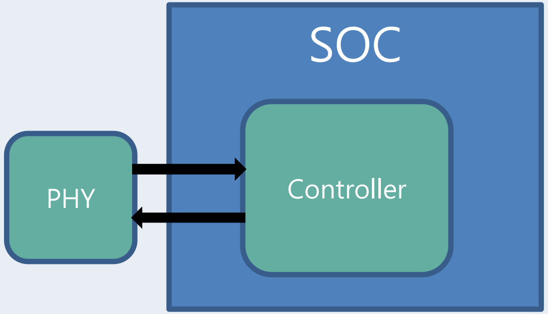
❑ Connects a link layer device to a physical medium such as an optical fiber or copper cable



Functions of a PHY



PHY In A System

PHY within controller	PHY within SoC	PHY outside SoC
• Shares same address space as the controller • No need of a separate PHY driver	• Connected to the controller using PIPE3 or UTMI interface • Should have a separate PHY driver	• Uses interface like ULPI to connect to an external transceiver • Should have a separate PHY driver
		

Existing ways to Implement PHY

- PHY driver within the controller driver
  - Code duplication
  - Tightly coupled to controller driver
  - Issues in code maintainability
- Ethernet PHY subsystem
  - Limited only to Ethernet drivers

Need of A Generic PHY framework

- As codebase for Zephyr is increasing, more and more PHY drivers will get added for all high speed IPs like USB, PCIe, CSI, Ethernet
- The intention of creating this framework is to bring the PHY drivers spread all over Zephyr to one place to increase code re-use and for better code maintainability

Proposed PHY framework

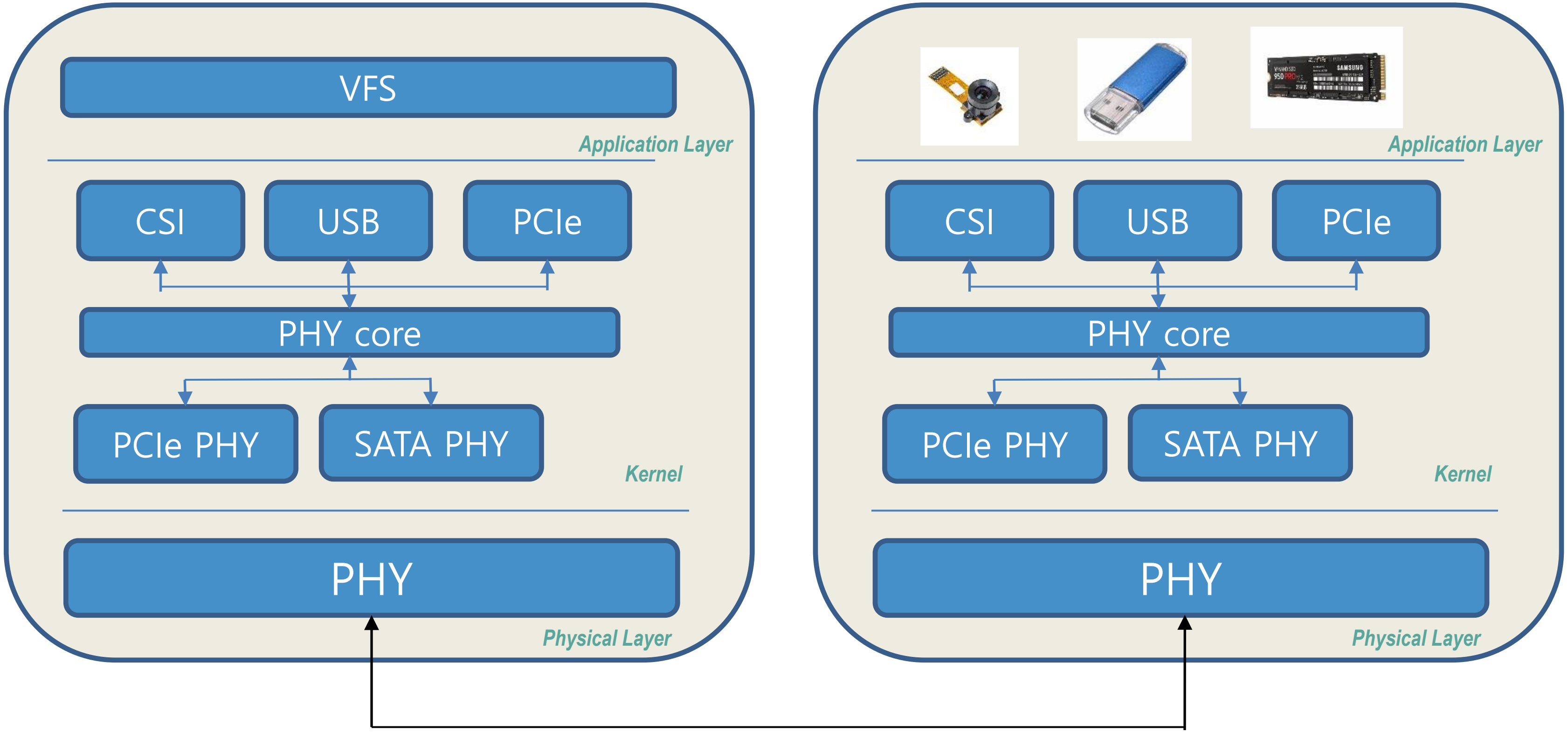
❑ This framework will be of use to devices that use external PHY (PHY functionality is not embedded within the controller)

❑ Derived from Linux Kernel

❑ Used for all IPs like USB, PCIe, CSI, etc.

❑ Controlled from the controller driver

Proposed PHY framework



VFS

Application Layer

CSI

USB

PCIe

PHY core

PCIe PHY

SATA PHY

Kernel

PHY

Physical Layer

Application Layer

CSI

USB

PCIe

PHY core

PCIe PHY

SATA PHY

Kernel

PHY

Physical Layer

PHY Framework Implementation

❑ **Binding**

- ❍ Device tree

❑ **PHY Driver**

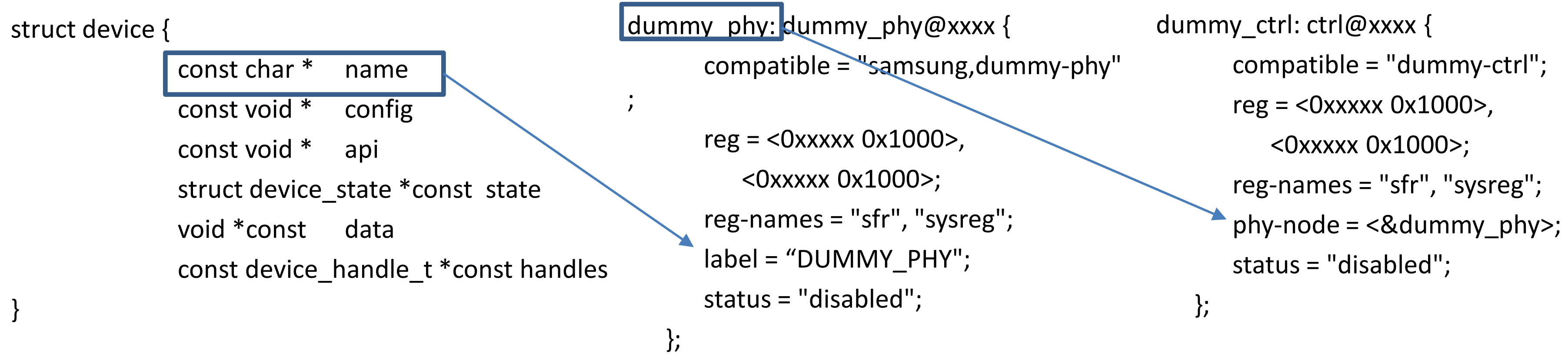
- ❍ Should implement phy ops like *phy_init, phy_exit*
- ❍ Register with common PHY framework

❑ **Controller Driver**

- ❍ Get reference to PHY
- ❍ Invoke PHY APIs like *phy_init, phy_exit*

Binding

- ❑ PHY Framework mainly depend on the name field of the struct device
- ❑ This name field usually linked to the label field in the DT node
- ❑ So device which use PHY framework it is mandatory add label property in DT node.



```
struct device {
                const char *    name
                const void *    config
                const void *    api
                struct device_state *const  state
                void *const     data
                const device_handle_t *const handles
}
```

```
dummy_phy: dummy_phy@xxxx {
        compatible = "samsung,dummy-phy"
;
        reg = <0xxxxx 0x1000>,
            <0xxxxx 0x1000>;
        reg-names = "sfr", "sysreg";
        label = “DUMMY_PHY";
        status = "disabled";
    };
```

```
dummy_ctrl: ctrl@xxxx {
        compatible = "dummy-ctrl";
        reg = <0xxxxx 0x1000>,
            <0xxxxx 0x1000>;
        reg-names = "sfr", "sysreg";
        phy-node = <&dummy_phy>;
        status = "disabled";
    };
```

PHY driver

- Struct PHY

```
struct phy {
    uint32_t base_addr;
    const struct device *dev;
    int id;
    const struct phy_ops *ops;
    struct k_mutex mutex;
    int init_count;
    int power_count;
 };
```

PHY ops

- PHY driver need to use phy_ops struct and initialize all the necessary call backs like init and exit
- After initializing phy_ops struct, PHY driver need to call phy_create function

```
struct phy_ops {
  int (*init)(struct phy *phy);
  int (*exit)(struct phy *phy);
  int (*power_on)(struct phy *phy);
  int (*power_off)(struct phy *phy);
  int (*reset)(struct phy *phy);
  int (*calibrate)(struct phy *phy);
 };
```

PHY init

- ❑ Called from the controller driver
- ❑ Arguments
  - ❍ struct phy (get from phy_get)
- ❑ Acquire lock for the phy, call init ops and increment init_count
- ❑ init ops on success should return non-negative value, on failure return negative value
- ❑ After ops called, lock will be released

PHY exit

- ❑ Called from the controller driver
- ❑ Arguments
  - ❍ struct phy (get from phy_get)
- ❑ Acquire lock for the phy and call exit ops and decrement init_count
- ❑ exit ops on success should return non-negative value, on failure return negative value
- ❑ After ops called, lock will be released

PHY operations

- ❑ PHY reset
  - ❍ Needs to be called in case reset of PHY is required

- ❑ PHY power on
  - ❍ Needs to be called during power on of the PHY

- ❑ PHY power off
  - ❍ Call when going for power off

- ❑ PHY calibrate
  - ❍ Used to calibrate phy hardware, typically by adjusting some parameters in runtime, which are otherwise lost after host controller reset and cannot be applied in phy_init() or phy_power_on().

Creating the PHY

- ❑ The PHY driver should create the PHY in order for other peripheral controllers to make use of it. It is called from PHY driver
- ❑ Arguments
  - ❍ struct device
  - ❍ struct phy_ops
- ❑ This function will allocate the phy structure, integrate the phy_ops provided by PHY driver, register the PHY driver to the PHY framework , initialize the mutex for the phy and mark the PHY driver is not yet initialized

PHY get

- Called from the controller driver
- Arguments
  - char *name
- phy_get function uses the name to search in the linked list and return the registered PHY
- If it exists return the struct phy, if not exists return NULL

Controller Driver

- ❑ Add phy driver DT node as phandle in controller DT node

```
DT_PROP(DT_PHANDLE(DT_NODELABEL(dummy_controller),phy-node),label);
```

- ❑ Use above macro to get the name of the phy driver which is registered in framework
- ❑ Pass this name to the phy_get function to get the corresponding PHY
- ❑ After successfully getting the phy from framework, now controller driver ready to call phy_init and phy_exit using phy ops

Sample PHY driver

- Drivers/phy/sample_phy.c

```
static int dummy_phy_init(struct phy *phy) {
        /* called during controller probe usually. Contains all programming to initialize PHY.
        Will typically create PHY */
}
static int dummy_phy_exit(struct phy *phy) {
        /* called during controller remove. Contains all programming for PHY cleanup.
        Will typically destroy PHY */
}
static int dummy_phy_power_on(struct phy *phy) {
        /* Enable clocks and power on PHY */
}
static int dummy_phy_power_off(struct phy *phy) {
        /* Disable clocks and power off PHY */
}
```

Sample PHY driver

- □ Drivers/phy/sample_phy.c

```
struct phy_ops dummy_phy_ops {
        .init = dummy_phy_init,
        .exit = dummy_phy_exit,
        .reset = xxxx,
        .calibrate = xxxx,
        .
        .
        .power_on = dummy_phy_power_on,
        .power_off = dummy_phy_power_off,

};
```

Sample Controller Driver

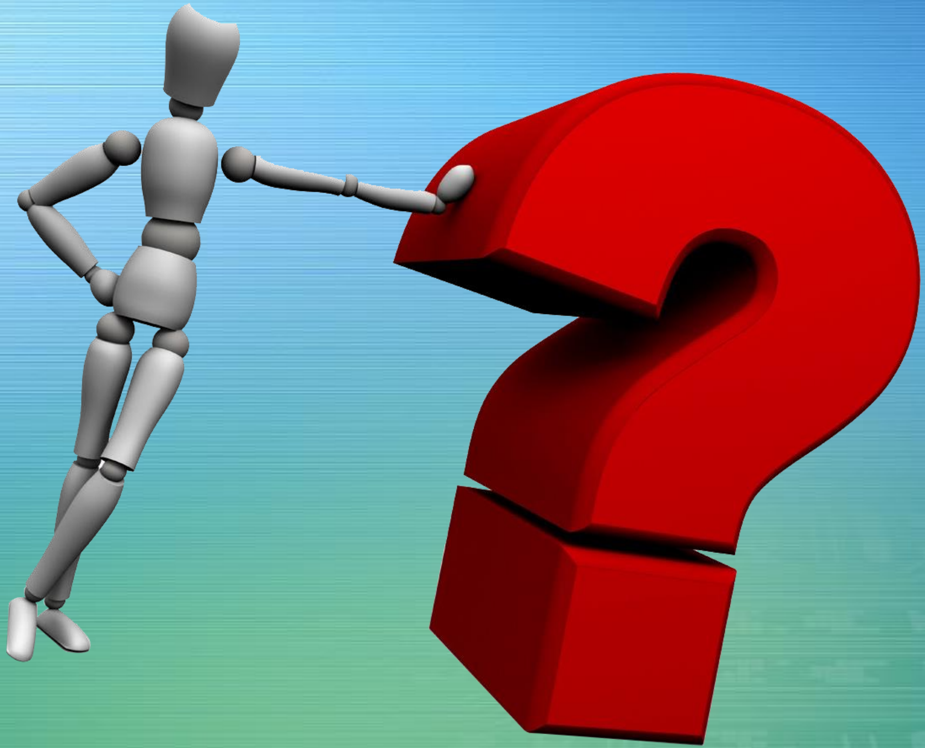
- ❑ Drivers/<usb/pcie/csi>/sample_controller.c

```
static int dummy_controller_init {
        struct phy *phy = phy_get(name);
        phy_init(phy);
        /* perform controller init */
}

static int dummy_controller_start_transfer {
        phy_power_on(phy);
        /* Prepare controller for transfer */
}

static int dummy_controller_stop_transfer {
        /* Do needful for transfer complete. Free buffers*/
        phy_power_off(phy);
}
```

Conclusion and Future Scope

- ❑ This generic framework will be very useful in increasing code readability and reducing code duplication
- ❑ Lots of other phy ops can be implemented and framework can be extended as per need basis
  - ❍ Phy_set_mode
  - ❍ Phy_get_bus_Width
  - ❍ Phy_pm_runtime

Any Questions ?



THANK YOU