GENERIC PHY FRAMEWORK

Kishon Vijay Abraham I
About Me

• I'm Kishon Vijay Abraham
  • Signed-off-by: Kishon Vijay Abraham I <kishon@ti.com>
• Working in Texas Instruments since 2007
• Contributing to linux kernel for the past four years
• Develop and Maintain PHY Subsystem (drivers/phy)
• Develop and Maintain PCIe glue for DRA7xx
• USB DWC3 driver support in u-boot
Agenda

• Introduction
• Building blocks of PHY
• PHY Integration
• Existing Mechanisms
• Introduction to Generic PHY Framework
• Using Generic PHY Framework
• Generic PHY Framework Internals
• Upcoming
Introduction

• PHY is an abbreviation for physical layer
• Responsible for transmitting data over a physical medium
• PHY connects the device controller with the physical medium
  – USB
  – SATA
  – PCIE
  – ETHERNET
BUILDING BLOCKS

- CLOCK & POWER CONTROL
- SERIALIZER/DE-SERIALIZER
- ENCODER/DECODER
- SCRAMBLER/DE-SCRAMBLER
- ERROR CORRECTION/COLLISION DETECTION
- PLL CLOCK SYNTHESIZER
PHY INTEGRATION

• PHY integrated within the controller
• PHY integrated within the SoC
• PHY external to the SoC
PHY WITHIN THE CONTROLLER

- Shares the same address space with the controller
- No separate PHY driver is required
PHY WITHIN THE SoC

• Connected to the controller using UTMI, PIPE3 interface specification
• Should have a separate PHY driver
PHY EXTERNAL TO THE SOC

• Connected to the controller using ULPI etc..
• Should have a separate PHY driver
Existing Mechanisms

• USB: USB PHY library
  – Comprehensive library with dt and non-dt support
  – Can be used only with USB PHYs

• PHYs are programmed in the controller driver itself
  – PHY and controller are tightly bound. Changing the PHY IP will break compatibility.

• Function pointers are passed in platform data to control PHY
  – Not possible with dt
Generic PHY Framework

• PHYs integrated outside the controller
• Allows the PHY to be controlled from the controller driver
• Derived from USB PHY Framework
• Used across different subsystems USB, SATA, PCIe
• Supports dt and non-dt boot
• Invokes pm_runtime_(*) APIs
Using Generic PHY Framework

• Bind the controller device and PHY device
  – Device tree
  – Non device tree

• PHY drivers
  – should implement phy_ops (init, exit, power_on, power_off)
  – Register with the PHY Framework

• Controller drivers
  – Get a reference to the PHY
  – Invoke PHY framework APIs (phy_init, phy_exit, phy_power_on, phy_power_off)
Device Tree

- PHY device node
  - \#phy-cells: Number of cells in the PHY specifier

- Controller device node
  - phys: list of phandles to the PHY device
  - phy-names: the names of the PHY corresponding to the phandle present in the phys property

- Device tree binding documentation

Documentation/devicetree/bindings/phy/phy-bindings.txt
Device Tree: Example 1

phy {
    compatible = "phy";
    ...
    ...
    #phy-cells = <0>;
}

controller {
    compatible = "controller";
    ...
    ...
    phys = <&phy>;
    phy-names = "phy";
}
Device Tree: Example 2

phy1 {
    compatible = "phy1";
    ...
    ...
    #phy-cells = <0>;
}

phy2 {
    compatible = "phy2";
    ...
    ...
    ...
    #phy-cells = <1>;
}

ccontroller {
    compatible = "controller";
    ...
    ...
    phys = <&phy1> <&phy2 PHY_TYPE>;
    phy-names = "phy1", "phy2";
}

Device Tree: Example 3

```c
phy_provider {
    compatible = "phy_provider";
    /* implement multiple PHYs: PHY_TYPE1 and PHY_TYPE2 */
    #phy-cells = <1>;
    ...
    ...
}

ccontroller {
    compatible = "controller";
    ...
    ...
    phys = <&phy_provider PHY_TYPE1> <&phy_provider PHY_TYPE2>;
    phy-names = "phy1", "phy2";
}
```
Non Device Tree

• Mapping should be created at runtime by using the following API

  ```c
  int phy_create_lookup(struct phy *phy, const char *con_id,
                       const char *dev_id)
  ```

• Should have a reference to the PHY and the device name of the controller device.

• Used only in two places
  - dwc3 host
  - twl4030 USB PHY
Sample PHY driver

drivers/phy/phy-sample.c

static int sample_phy_init(struct phy *phy) {
    /* Initialize Sample PHY */
}

static int sample_phy_power_on(struct phy *phy) {
    /* Enable clocks and power on Sample PHY */
}

static int sample_phy_power_off(struct phy *phy) {
    /* Disable clocks and power off Sample PHY */
}

static int sample_phy_exit(struct phy *phy) {
    /* Sample PHY cleanup */
}
Sample PHY driver

struct phy_ops sample_phy_ops {
    .init = sample_phy_init,
    .power_on = sample_phy_power_on,
    .power_off = sample_phy_power_off,
    .exit = sample_phy_exit,
};

/* Sample PHY specific implementation of of_xlate.
* sets the PHY to the mode obtained from of_phandle_args.
* If the PHY provider implements multiple PHYs, then this of_xlate should
* find the correct PHY from the np present in of_phandle_args and return it
*/
static struct phy *sample_phy_xlate(struct device *dev,
        struct of_phandle_args *args) {
    sample->mode = args->args[0];
    return sample->phy;
}
Sample PHY driver

static int sample_phy_probe(struct platform_device *pdev) {
    ...
    phy = devm_phy_create(dev, dev->of_node, &sample_phy_ops);

    if (dev->of_node) {
        /* use default implementation of of_xlate if the device tree node
         * represents a single PHY and if the PHY driver does not want to
         * receive any arguments that's added along with the phandle
         */
        // phy_provider = devm_of_phy_provider_register(phy->dev,
        // of_phy_simple_xlate);

        phy_provider = devm_of_phy_provider_register(phy->dev,
            sample_phy_xlate);
    } else {
        phy_create_lookup(phy, "phy", "sample-controller");
    }
    ...
}
Sample Controller driver

drivers/<controller>/controller-sample.c

static int sample_controller_probe(struct platform_device *pdev) {
    phy = devm_phy_get(dev, "sample-phy");
    ...
}

int sample_controller_init() {
    /* controller initialization goes here */
    phy_init(phy);
    ...
}

int sample_controller_start_transfer() {
    phy_power_on(phy);
    /* program the controller to start transfer */
    ...
}

int sample_controller_complete_transfer() {
    /* free buffer etc */
    phy_power_off(phy);
    ...
}
Sequence Diagram

Controller Driver

phy_create()

phy_provider_register()

phy_get()

phy_init()

phy_create_mapping()

phy_power_on()

phy_pm_runtime()

phy_power_off()

phy_exit()

PHY-CORE

phy_create()

phy_provider_register()

 phy_create_mapping()

phy_get()

phy_init()

phy_power_on()

phy_pm_runtime()

phy_power_off()

phy_exit()

phy_destroy()

PHY Driver
**Phy-core Internals**

### PHY_PROVIDER (dt boot)
- struct phy* (*of_xlate)(struct device*,
  struct of_phandle_args*)
- of_phy_simple_xlate => default
  implementation

### PHY
- struct device
- struct phy_ops*
- struct phy_attrs

### PHY_OPS
- (*init)(struct phy*)
- (*power_on)(struct phy *)
- (*power_off)(struct phy *)
- (*exit)(struct phy*)

### PHY_LOOKUP (binding)
- const char* dev_name
- const char* port_name
- struct phy*
PHY DEVICE MODEL

Created by
* of_platform_populate (dt boot)
* platform_device_add (non-dt boot)

Created during phy_create
Upcoming

• ULPI PHY support

• Handling USB specific PHY functionality
### Upstreamed PHY drivers (4.0)

<table>
<thead>
<tr>
<th>PHY</th>
<th>Domain</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kona PHY</td>
<td>USB2</td>
<td>Broadcom</td>
</tr>
<tr>
<td>Berlin PHY</td>
<td>SATA</td>
<td>Marvell</td>
</tr>
<tr>
<td>Exynos PHY</td>
<td>USB2, SATA, DISPLAY,</td>
<td>Samsung</td>
</tr>
<tr>
<td>HIX5HD2 SATA PHY</td>
<td>SATA</td>
<td>Hisilicon</td>
</tr>
<tr>
<td>MIPHY365</td>
<td>SATA, PCIE</td>
<td>STMicroelectronics</td>
</tr>
<tr>
<td>MVEBU PHY</td>
<td>SATA</td>
<td>Marvell</td>
</tr>
<tr>
<td>OMAP USB2 PHY</td>
<td>USB2</td>
<td>Texas Instruments</td>
</tr>
<tr>
<td>APQ8064 PHY</td>
<td>SATA</td>
<td>Qualcomm</td>
</tr>
<tr>
<td>IPQ806X PHY</td>
<td>SATA</td>
<td>Qualcomm</td>
</tr>
<tr>
<td>S5PV210 PHY</td>
<td>USB2</td>
<td>Samsung</td>
</tr>
<tr>
<td>SPEAR1310/1340 MIPHY</td>
<td>SATA, PCIE</td>
<td>STMicroelectronics</td>
</tr>
<tr>
<td>SUN4I USB PHY</td>
<td>USB</td>
<td>Allwinner</td>
</tr>
<tr>
<td>TI PIPE3</td>
<td>SATA, PCIE, USB3</td>
<td>Texas Instruments</td>
</tr>
<tr>
<td>X-GENE PHY</td>
<td>SATA</td>
<td>Applied Micro</td>
</tr>
</tbody>
</table>
# Upstreamed PHY drivers (4.0) cont.

<table>
<thead>
<tr>
<th>PHY</th>
<th>Domain</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armada375 PHY</td>
<td>USB2</td>
<td>Marvell</td>
</tr>
<tr>
<td>KONA PHY</td>
<td>USB2</td>
<td>Broadcom</td>
</tr>
<tr>
<td>Rockchip PHY</td>
<td>USB2</td>
<td>Rockchip</td>
</tr>
<tr>
<td>RCAR PHY</td>
<td>USB</td>
<td>Renesas</td>
</tr>
<tr>
<td>QCOM UFS PHY</td>
<td>UFS</td>
<td>Qualcomm</td>
</tr>
</tbody>
</table>
Acknowledgements

• Felipe Balbi
• Greg KH
• Linux Community
References

- drivers/phy/
- Documentation/phy.txt
- Documentation/devicetree/bindings/phy/phy-bindings.txt
- Device tree specification: https://www.power.org/documentation/epapr-version-1-1/
- Device tree for Dummies: https://www.youtube.com/watch?v=m_NyYEBxfn8
THANK YOU

For Queries and Feedback

kishon@ti.com, kishonvijayabraham@gmail.com