GENERIC PHY FRAMEWORK: AN OVERVIEW

Kishon Vijay Abraham I
About Me

• 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
• Presented a paper on “USB Debugging and Profiling Techniques” in ELCE 2012
Agenda

- Introduction
- Functionalities
- PHY Standards
- PHY Integration
- PIPE3 PHY
- Generic PHY Framework
- Sample PHY/Controller Driver
- DT Representation
- Non-dt Support
- Future Enhancements
Introduction

- PHY is an abbreviation for the physical layer
- Responsible for transmitting data over a physical medium
- PHY connects the device controller with the physical medium
  - USB
  - SATA
  - PCIE
  - ETHERNET
Functionalities

• Serialization/De-serialization
• Encoding/Decoding
• Synchronization
• Error Correction
• Collision Detection
• Data transmission rate
PHY Standards

• ULPI
• UTMI+
• PIPE3
• D-PHY
• M-PHY
• IEEE 802.3
PHY INTEGRATION

• PHY integrated within the controller
  – Shares the same address space with the controller
  – No separate PHY driver is required

• PHY integrated 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
PIPE3 PHY

Device Controller

MAC

PIPE3 Tx Port

PIPE3 Rx Port

REF CLK

Data [31:0]

MCLK

ctrl/status

Data [31:0]

PCLK

ctrl/status

8B/10B Encoder

PIPE TO PMA

Tx Adapter Logic

8B/10B Decoder

PIPE TO PMA

Rx Adapter Logic

PLL

PHY clock

POWER CTRL I/F

Transmitter

Tx

Serializer

ctrl/status

Data [9:0]

Data [9:0]

Data [9:0]

ctrl/status

ctrl/status

Receiver

Rx

De-Serializer
PIPE3 PHY PROGRAMMING

Device Controller

MAC

PIPE3 Tx Port

PIPE3 Rx Port

Power Control

CLOCK Programming

PIPE3 PHY

Ref Clk

Data [31:0]

Ctrl/status

Data [31:0]

PCLK

Ctrl/status

Data [9:0]

Ctrl/status

Data [9:0]

Ctrl/status

Phy Clock

PCs

8B/10B Encoder

Pipe to PMA Tx Adapter Logic

8B/10B Decoder

Pipe to PMA Rx Adapter Logic

Pcs Programming

PLL

PLL Programming

Pma

Transmitter

Tx

Tx Programming

Serializer

Rx

Rx Programming

De-Serializer

Rx Programming

Phy Clock
Generic PHY Framework

SATA
PCIE
USB3
VIDEO
DISPLAY
...

Kernel/Domain Stack/Application

Controller Driver interface

GENERIC PHY FRAMEWORK

PHY Driver interface

SATA PHY
PCIE PHY
USB3 PHY
VIDEO PHY
DISPLAY PHY
...

CONTROLLER DRIVERS
(drivers/usb/musb/dwc3/)

PHY CORE
(drivers/phy/phy-core.c)

PHY DRIVERS
(drivers/phy/)
Generic PHY Framework

- Derived from USB PHY Framework
- Binds controller driver with PHY driver
- PHYs integrated outside the controller
- Supports dt and non-dt boot
- Op's the controller driver can use to control the PHY
  - phy_init
  - phy_power_on
  - phy_power_off
  - phy_exit
- phy_pm_runtime_*
Phy-core Framework

**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_init_data*
- struct phy_attrs

**PHY_OPS**

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

**PHY_CONSUMER (binding)**

- const char* dev_name
- const char* port_name
Sequence Diagram

Controller Driver

PHY-CORE

PHY Driver

phy_create()

phy_provider_register()

phy_get()

phy_init()

phy_power_on()

phy_power_off()

phy_exit()

phy_pm_runtime*

phy_destroy()
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**

```c
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, pdata->init_data);
    
    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);
    }
    ...
}
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);
    ...
}
DT REPRESENTATION

• Single PHY

• Multi PHY
  – Multiple instances of the same PHY
  – Single PHY IP encompasses multiple PHYs
Single PHY

MUSB controller integrated in OMAP4
Single PHY (dt representation)

```c
ocp2scp@xxxxx {
    compatible = "ti,omap-ocp2scp";
    ...
    ...
    usb2phy@0 {
        compatible = "ti,omap-usb2";
        ...
        ...
        #phy-cells = <0>;
    }
}

usb_otg_hs@xxxxx {
    compatible = "ti,omap4-musb";
    ...
    ...
    phys = &usb2phy;
    phy-names = "usb2-phy";
}
```
Multi PHY (Multiple instances of same IP)

DWC3, PCIe and SATA controller integrated in DRA7xx
Multi PHY (dt representation)

ocp2scp3@xxxxx {
    compatible = "ti,omap-ocp2scp";
    pciephy: pipe3phy@0 {
        compatible = "ti,phy-pipe3";
        ...
        #phy-cells = <1>;
    }

    sataphy: pipe3phy@1 {
        compatible = "ti,phy-pipe3";
        ...
        #phy-cells = <1>;
    }
}

ocp2scp1@xxxxx {
    compatible = "ti,omap-ocp2scp";
    usb3phy: pipe3phy@0 {
        compatible = "ti,phy-pipe3";
        ...
        #phy-cells = <1>;
    }

    usb2phy@0 {
        compatible = "usb2-phy";
        #phy-cells = <0>;
    }
}

pcie@0 {
    compatible = "ti,dra7-pcie";
    ...
    phys = <&pciephy PCIEPHY>;
    phy-names = "pcie-phy";
}

sata@0 {
    compatible = "snps,dwc-ahci";
    ...
    phys = <&sataphy SATAPHY>;
    phy-names = "sata-phy";
}

usb_otg_ss@xxxxx {
    compatible = "snps,dwc3";
    ...
    phys = <&usb2phy>, <&usb3phy USBPHY>;
    phy-names = "usb2-phy", "usb3-phy";
}

Multi PHY (Single IP encompass multiple phys)

MiPHY365 for SoC STiH416
Multi PHY (dt representation)

miphy: miphy365x@fe382000 {
  compatible = "st,miphy365x-phy";
  ...
  phy_port0: port@fe382000 {
    ...
    ...
    #phy-cells = <1>;
  }
}

phy_port1: port@fe38a000 {
  ...
  ...
  #phy-cells = <1>;
}

sata0: sata@fe380000 {
  ...
  phys = <&phy_port0 MIPHY_TYPE_SATA>;
  phy-names = "sata-phy";
  ...
}

OLD METHOD

miphy: miphy365x@fe382000 {
  compatible = "st,miphy365x-phy";
  ...
  #phy-cells = <2>;
}

sata0: sata@fe380000 {
  ...
  phys = <&miphy 0 MIPHY_TYPE_SATA>;
  phy-names = "sata-phy";
  ...
}
Non-dt Support

- PHYs should be aware of their consumers
- Consumer data is added as platform data to the platform device
- PHY driver should pass it to phy-core during phy_create()

```c
struct phy_consumer consumers[] = {
    PHY_CONSUMER("musb-hdrc.0", "usb"),
};

struct phy_init_data init_data = {
    .consumers = consumers,
    .num_consumers = ARRAY_SIZE(consumers),
};
```
Future Enhancements

• Adapt existing PHY drivers to Generic PHY Framework
• Support for ULPI PHY driver
• Support for Ethernet PHYs
## Upstreamed PHY drivers (3.17)

<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>
Acknowledgements

- Felipe Balbi
- Greg KH
- Everyone who has contributed to PHY (Authors, Reviewers, Testers etc..)
References

- wikipedia.org/wiki/Physical_layer
- mipi.org/specifications/physical-layer
- PIPE specification
- drivers/phy/
- Documentation/phy.txt
THANK YOU

For Queries and Feedback

kishon@ti.com, kishonvijayabraham@gmail.com