From the Ethernet MAC to the link partner

Maxime Chevallier
maxc@bootlin.com

Antoine Ténart
antoine@bootlin.com

bootlin
embedded Linux and kernel engineering

Antoine Ténart

- Linux kernel engineer and trainer at Bootlin.
  - Linux kernel and driver development, system integration, boot time optimization, consulting. . .
  - Embedded Linux, Linux driver development, Yocto Project & OpenEmbedded and Buildroot training, with materials freely available under a Creative Commons license.
  - `https://bootlin.com`
- Contributions:
  - Worked on network (MAC, PHY, switch) and cryptographic engines.
  - Contributed to the Marvell EBU SoCs upstream support.
  - Introduced the Marvell Berlin SoCs upstream support.
  - Co-maintainer of the Annapurna Alpine SoCs.

Maxime Chevallier

- Linux kernel engineer at Bootlin.
  - Linux kernel and driver development, system integration, boot time optimization, consulting...
  - Embedded Linux, Linux driver development, Yocto Project & OpenEmbedded and Buildroot training, with materials freely available under a Creative Commons license.
  - `https://bootlin.com`
- Contributions:
  - Worked on network (MAC, PHY, switch) engines.
  - Contributed to the Marvell EBU SoCs upstream support.
  - Also worked on SPI and real-time topics.

Preamble - goals

- Discover what are the components of an Ethernet data link and physical layer.
- Have a first glance at the technologies and protocols used for the components to communicate.
- Learn how to configure all of this in Linux.
- This subject is wide and complex: we'll take shortcuts and make approximations.

Introduction to the Ethernet link layer

Maxime Chevallier
maxc@bootlin.com

Antoine Ténart
antoine@bootlin.com

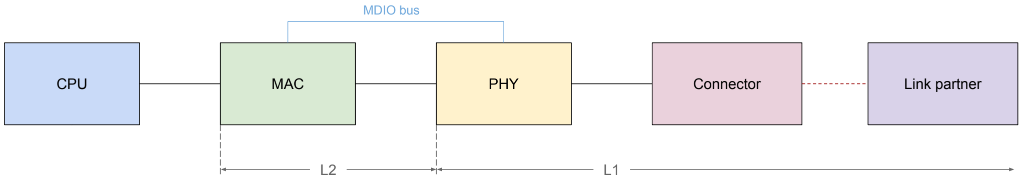
bootlin
embedded Linux and kernel engineering

Reminder: the OSI model

1. Physical layer.
   - Rx/Tx of unstructured data, converts the digital data into a signal (e.g. electrical, radio, optical).
2. Data link layer (e.g. `Ethernet`).
   - Data transfer between directly connected nodes using frames.
3. Network layer (e.g. `IP`).
   - Data transfer between nodes — directly connected or being routed through other nodes — using packets.
4. Transport layer (e.g. `TCP`, `UDP`).
   - Reliability, flow control, QoS, ordering, segmentation. . .

- We'll focus on the first two layers, when using **Ethernet**.

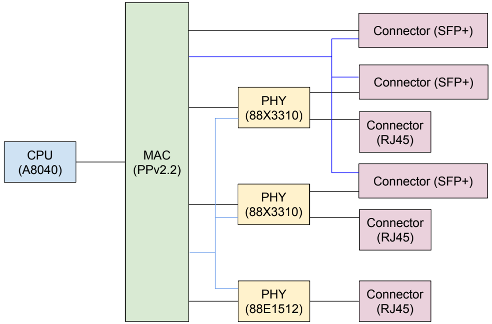
The Ethernet link layer



- The MAC (`media access control`), makes up the *data link layer*:
  - Transfers/receives frames.
  - Handles preambles and paddings.
  - Protects against errors — checks frames and their FCS (`frame check sequence`).
  - ...
- The network PHY, makes up the *physical layer*:
  - Connects the *link layer device* to a *physical medium*.
  - Accessible through an `MDIO` bus.
- Cages (e.g. RJ45, SFP), physical mediums (e.g. copper, fiber)...

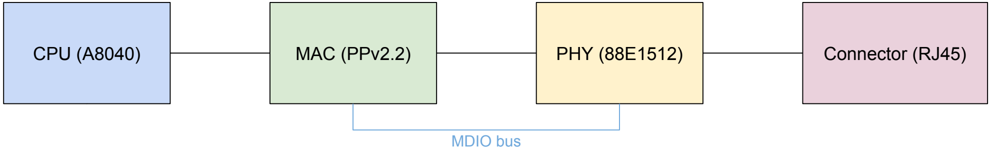
The MacchiatoBin example (1/4)

- The Ethernet link layer is built using the elements we just saw (`MAC`, `PHY`...), and can differ a bit depending on the hardware design and purpose.
- Let's focus on the *MacchiatoBin double shot* example, a board using a Marvell Armada 8040 SoC — `https://macchiatobin.net`
  - It has 4 network ports, 3 different link designs and 6 cages.



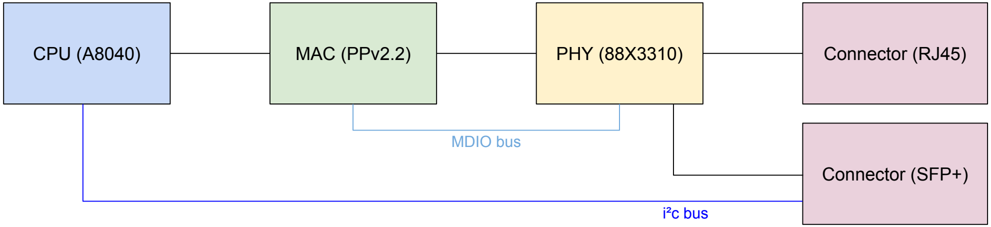
The MacchiatoBin example (2/4)

- The first port (`eth2` in Linux) can handle up to 1G links, connected to an RJ45 port.



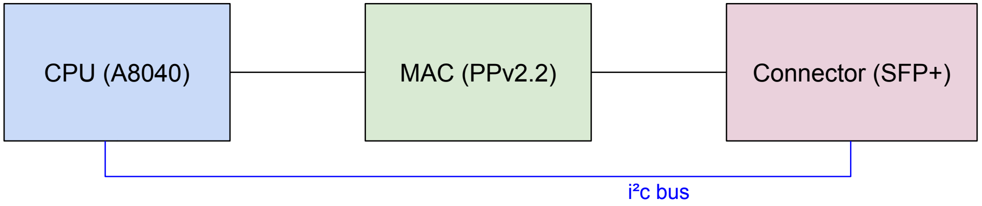
The MacchiatoBin example (3/4)

- The second and third ports (`eth0`/`eth1` in Linux) can handle up to 10G links, connected to RJ45 and SFP+ cages.
- Only one cage can be used at a time.
- Dynamic reconfiguration (`MAC`, `SerDes lanes`, `PHY`) allows to switch between the two usages.



The MacchiatoBin example (4/4)

- The fourth port (`eth3` in Linux) can handle up to 2.5G links, connected to an SFP+ cage.
- No `PHY` on the board — direct `MAC` to `MAC` communication, or a `PHY` can be on the `SFP` external connector.



Linux representation

- The Ethernet `MAC` controller is driven by an Ethernet driver:
  - Within `drivers/net/ethernet/`.
  - Represented by `struct net_device`.
- The `PHY` is driven by a network PHY driver:
  - Within `drivers/net/phy/`.
  - Represented by `struct phy_device`.
- In the *MacchiatoBin* case:
  - `drivers/net/ethernet/marvell/mvpp2/*,mvmdio.c`
  - `drivers/net/phy/marvell.c`
  - `drivers/net/phy/marvell10g.c`
- In case the `MAC` and the `PHY` are in the same hardware package, everything can be handled directly in `drivers/net/ethernet/`.

Reporting tools

- `ethtool`: reports information from the Ethernet driver.
  - It can be what the MAC is seeing,
  - Or if the `MAC` and the `PHY` are in the same package, the view of the package itself.
- `mii-tool`: deprecated and mostly replaced by `ethtool`, but can be useful to dump the `PHY` status.

Interfaces overview

- For the components of the Ethernet link to communicate, two interfaces are standardized by the *IEEE 802.3* specifications and amendments:
  - The Media-Independent Interface (`MII`):
    - Connects various types of `MAC` to various types of `PHY`.
    - Originally standardized by the *IEEE 802.3u*.
    - E.g. MII, GMII, RGMII, SGMII, XGMII, XAUI...
  - The Media-Dependent Interface (`MDI`):
    - Connects the physical layer implementation to the physical medium.
    - E.g. 100BASE-T, 1000BASE-T, 1000BASE-CX, 1000BASE-SZ, 10GBASE-T...

Ethernet standards

The 802.3 standard is short and simple, with only a few specifications to remember :

Specifications

- 802.3z 802.3aa
- 802.3ab 802.3ac
- 802.3ad 802.3ae
- 802.3af 802.3ag
- 802.3ah 802.3aj
- 802.3ak 802.3an
- 802.3ap 802.3aq
- 802.3as 802.3at
- 802.3av 802.1AX
- 802.3az 802.3ba
- 802.3bc 802.3bd
- 802.3be 802.3bf
- 802.3bg 802.3bj
- 802.3bk 802.3bm
- 802.3bn 802.3bp ...

Formats

- 10BASE2
- 10BASE5
- 10BASE-F
- 10BASE-FB
- 10BASE-FL
- 10BASE-FP
- 10BASE-T
- 100BASE-BX10
- 100BASE-FX
- 100BASE-LX10
- 100BASE-T
- 100BASE-T2
- 100BASE-T4
- 100BASE-TX
- 100BASE-X
- 1000BASE-BX10
- 1000BASE-CX
- 1000BASE-KX
- 1000BASE-LX
- 1000BASE-LX10
- 1000BASE-PX
- 1000BASE-SX
- 1000BASE-T
- 1000BASE-X
- 2.5GBASE-T
- 5GBASE-T
- 10GBASE-CX4
- 10GBASE-E
- 10GBASE-ER
- 10GBASE-EW
- 10GBASE-KR
- 10GBASE-KX4
- 10GBASE-L
- 10GBASE-LR
- 10GBASE-LRM
- 10GBASE-LW
- 10GBASE-LX4
- 10GBASE-PR
- 10/1GBASE-PRX
- 10GBASE-R
- 10GBASE-S
- 10GBASE-SR
- 10GBASE-SW
- 10GBASE-T
- 10GBASE-W
- 10GBASE-X
- 40GBASE-R
- 40GBASE-CR4
- 40GBASE-ER4
- 40GBASE-FR
- 40GBASE-KR4
- 40GBASE-LR4
- 40GBASE-SR4
- 100GBASE-P
- 100GBASE-R
- 100GBASE-CR4
- 100GBASE-CR10
- 100GBASE-KP4
- 100GBASE-KR4
- 100GBASE-ER4
- 100GBASE-LR4
- 100GBASE-SR4
- 100GBASE-SR10

Ethernet link modes notation

- 802.3 standards use a special notation to describe links and protocols:
- speedBand-MediumEncodingLanes : 1000Base-T, 10GBase-KR, 100Base-T4...
- Band: BASEband, BROADband or PASSband.
- Medium
  - Base-**T**: Link over twisted-pair copper cables (Classic RJ45).
  - Base-**K**: Backplanes (PCB traces) links.
  - Base-**C**: Copper links.
  - Base-**L**, Base-**S**, Base-**F**: Fiber links.
  - Base-**H**: Plastic Fiber.
  - ...
- Encoding: Describe the block encoding used by the PCS
  - Base-**X**: 10b/8b encoding.
  - Base-**R**: 66b/64b encoding.
- Lanes: Number of lanes per link (for Base-**T**, number of twisted pairs used).

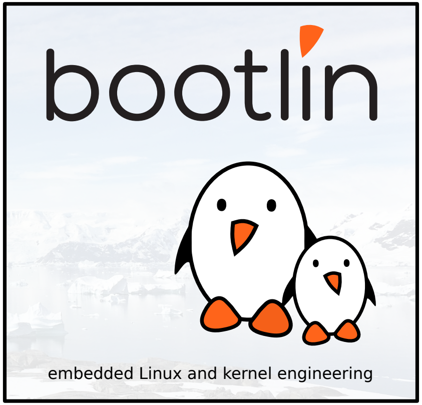
Link parameters

- An interface specification has many characteristics:
  - **Speed**: the transmission rate at which data is flowing through the link: 10Mbps, 100Mbps, 1000Mbps, 2.5Gbps, 10Gbps, 40Gbps...
  - **Duplex**: it can be *half-duplex* (the device is either transmitting or receiving data at a given time) or *full-duplex* (transmission and reception can happen simultaneously).
  - **Auto-negotiation**: can be used to exchange information about the *duplex*, *transmission rate*... when a device is capable of handling different modes or standards.
    - Through `MII` (*in-band*) or `MDIO` (*out-of-band*).
- Different specifications can operate at the same speed, or using the same duplex.
- But the link can only be operational if compatible *MII* and *MDI* protocols are used.
- A given link can support multiple modes, through **advertisement**.

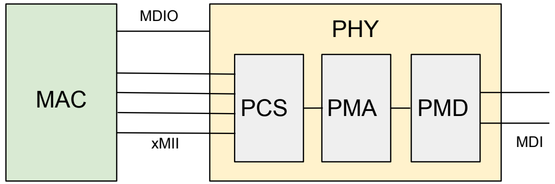
Media interfaces

Maxime Chevallier
maxc@bootlin.com

Antoine Ténart
antoine@bootlin.com



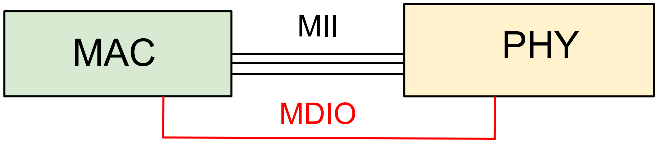
MAC to PHY connection



Inside a PHY:

- **PCS**: **P**hysical **C**oding **S**ubsystem
  - Encodes and decodes the MII link.
  - Several PCS are described in different specifications: 1000Base-X, 10Base-R. . . .
- **PMA**: **P**hysical **M**edium **A**ttachment
  - Translates between PCS and PMD.
  - Handles collision detection and data transfers.
- **PMD**: **P**hysical **M**edium **D**ependent
  - Interfaces to the physical transmission medium

MDIO bus



Management **D**ata **I**nput **O**utput

- Also called `SMI`.
- Two lines: `MDC` for clock, `MDIO` for data.
- Serial addressable bus between `MAC` and up to 32 `PHYs`.
- Access to `PHY` configuration and status registers.
- Not always part of the `MAC`, can be a separate controller.
- **Clause 22**: 5bit register addresses, 16bit data.
- **Clause 45**: Extends `C22` in a backwards-compatible way.
  - 16bit register addresses, 16bit data.
  - Multiple ”devices” per PHY, each with a register set.

MDIO in Linux

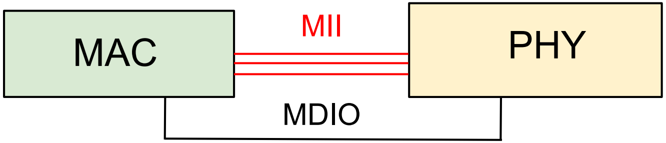
- C22 and C45 are supported:
  - drivers/net/phy/phy_device.c
  - drivers/net/phy/phy-c45.c
- The driver is selected by matching the UID register:

```
struct phy_driver mv3310_drivers[] = {{
        .phy_id         = 0x002b09aa,
        .phy_id_mask    = 0xfffffff0,
        ...
```

- Each PHY is described as a child of the mdio bus:

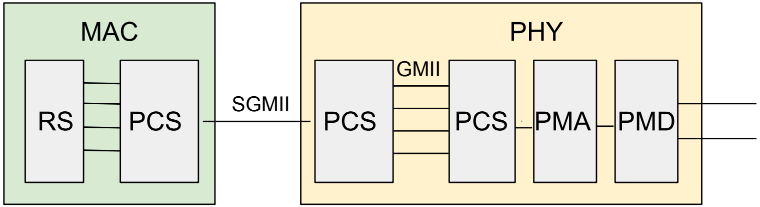
```
&mdio {
  ge_phy: ethernet-phy@0 {
    /* Clause 45 register accesses */
    compatible = "ethernet-phy-ieee802.3-c45";
    /* PHY id 0 */
    reg = <0>;
  };
};
```

MII family



- **MII**: **M**edia **I**ndependent **I**nterface
  - Originally a single standard, later extended. 16 pins, up to 100Mbps.
  - **RMII**: **R**educed **MII**: 8 pins.
- **GMII**: **G**igabit **MII**
  - 24 pins, 1Gbps, compatible with MII for 10/100 Mbps.
  - **RGMII**: **R**educed **GMII**: 12 pins.
  - **RGMII-ID**: Hardware variant with clock delay tweaks.
- **XGMII**: **X** (ten) **G**igabit **MII**
  - 74 pins, 10Gbps. Mostly used for on-chip MAC to PHY links.

Serializing the MII



- x**MII** interfaces have a high pin count, duplicated for each PHY.
- Re-use already defined PCS and PMA to serialize the xMII link.
- **RS**: **R**econciliation **S**ublayer: Glue between the MAC and the PCS.
- **SerDes Lanes**: Differential pair transmitting an encoded serialized signal.
  - Base-**X** (10b/8b) or Base-**R** (66b/64b).
  - Embedded or Parallel clock.
  - Handled by the Generic PHY Subsystem.

MII Family - Continued

- **SGMII**: **S**erialized **GMII**
  - De-facto standard, 4 differential pairs, Base-X PCS.
  - Designed for 1Gbps, but a 2.5Gbps variant exists.
  - **QSGMII**: Quad SGMII, 5Gbps.
- **XAUI**: **X** (ten) Gigabit **A**ttachment **U**nit **I**nterface
  - XGMII serialized on 4 SerDes lanes, Base-X PCS.
  - **RXAUI**: Reduced XAUI, using only 2 SerDes lanes.
- **XFI**: Part of XFP specs (outside of IEEE 802.3)
  - 10Gbps over one SerDes lane.
  - Uses 10GBase-R PCS.
  - Similar to **10GBase-KR**

MAC to PHY link in linux

- PHY connection represented by phy_interface_t.
- The phylink framework provides a representation of this link.
- Specified in DT using phy-mode in the MAC driver node:

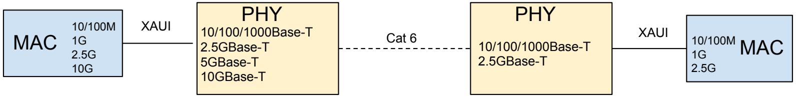
```
&eth1 {
        status = "okay";

        /* Reference to the PHY node */
        phy-handle = <&ge_phy>;

         /* PHY interface mode */
        phy-mode = "sgmii";
};
```

Network PHY driver in Linux

- A `PHY` driver is responsible for:
  - Handling the auto-negotiation parameters.
  - Reporting the link status to the MAC
    - Except in `in-band` status management.
- Interfaces with the `phylib` and `phylink` frameworks.
- `PHY` registers are standardized, the `phylib` does most of the hard work.
- Starts to have more advanced features:
  - Report statistics.
  - Configure the Wake-on-LAN parameters.
  - Implement MACSec.

Auto-negotiation

- What the PHY advertises is dictated by software.
- In Linux, the phylib subsystem manages the advertised modes.
- Each MAC and PHY driver reports its supported modes.
- phylink can take the MAC to PHY link into account.

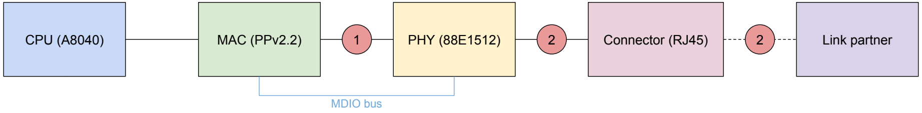


- XAUI interface, 10Gbps capable.
- → PHY advertises all of its capabilities.

- XAUI interface.
- → PHY advertises 10/100/1000M and 2.5G in Base-T.

Link establishes at 2.5Gbps

The MacchiatoBin example



1	SGMII
2	1000BASE-T / full-duplex

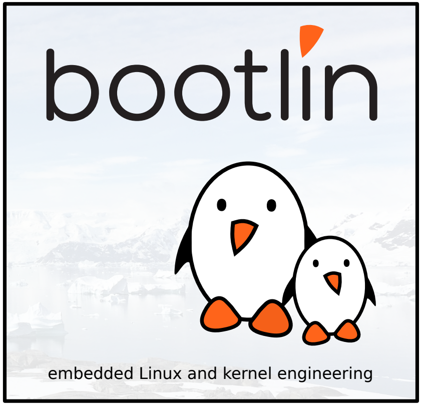
```
# ethtool eth2
Settings for eth2:
        Supported link modes:   10baseT/Half 10baseT/Full
    (a)                         100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised auto-negotiation: Yes
        Link partner advertised link modes:  10baseT/Half 10baseT/Full
                                             100baseT/Half 100baseT/Full
                                             1000baseT/Full
        Link partner advertised auto-negotiation: Yes
    (b) Speed: 1000Mb/s
    (c) Duplex: Full
    (d) Port: MII
        PHYAD: 0
        Auto-negotiation: on
        Link detected: yes
# mii-tool eth2  (e)
eth2: negotiated 1000baseT-FD flow-control, link ok
```

- **(a)**: supported link modes by the `MAC`.
- **(b)**: overall transmission rate.
- **(c)**: overall duplex mode.
- **(d)**: port type.
- **(e)**: MDI protocol used.

Evolution of the Ethernet interface

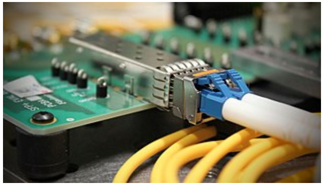
Maxime Chevallier
maxc@bootlin.com

Antoine Ténart
antoine@bootlin.com



SFP modules

- The small form-factor pluggable transceiver (SFP) is a module used for data communication.
- Its form factor is described by a specification, which makes it widely used by networking device vendors.
- An SFP interface supports various media, such as fiber optic or copper cables.
- It is **hot-pluggable** and can **optionally embed a PHY**.
- SFP transceivers can be **passive**, even for optical links.
  - The SerDes driver or the SFP module reports the link state.



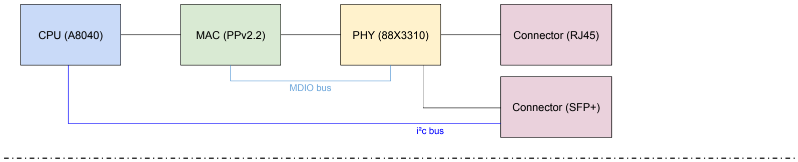
CC BY-SA 3.0 — Christophe Finot

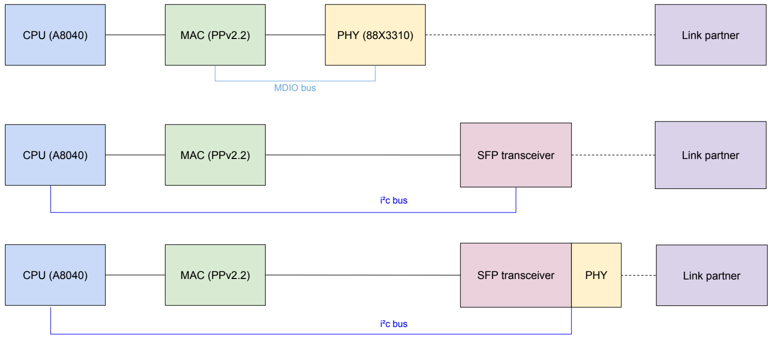
The need for a dynamic link infrastructure

- The Ethernet link is no longer fixed, with a single `MAC` connected to a single `PHY` with a single `connector`.
- `PHY` can be hot-pluggable when in an `SFP` transceiver.
- Part of the `PHY` can be embedded into the `MAC`, such as the `PCS`. The SerDes lanes can be configured and connected to various devices such as other `PHY`s or modules (`SFP`, `SFF`).
  - This allows more flexibility: less lanes, greater distance can be covered, `SFP` cages can be connected directly to the `MAC`...
  - Remember `eth3` on the *MacchiatoBin*?

⇒ The Ethernet link in its whole should be **dynamically reconfigurable**.

Mandatory MacchiatoBin example (`eth0/eth1`)





Phylink to the rescue (1/2)

- To solve this problematic the **phylink** infrastructure was introduced:

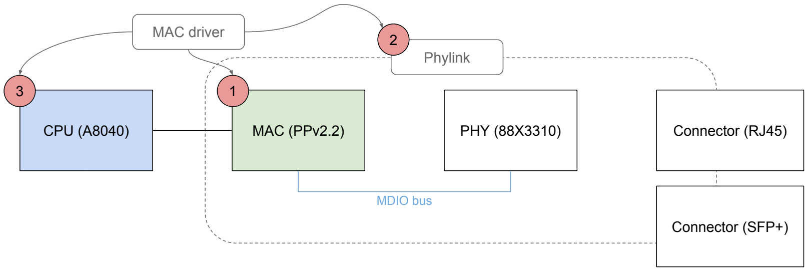
```
commit 9525ae83959b60c6061fe2f2caabdc8f69a48bc6
Author: Russell King <rmk+kernel@arm.linux.org.uk>
Commit: David S. Miller <davem@davemloft.net>

    phylink: add phylink infrastructure

    [...]

    Phylink aims to solve this by providing an intermediary between the
    MAC and PHY, providing a safe way for PHYs to be hotplugged, and
    allowing a SFP driver to reconfigure the SerDes connection.

    [...]
```

Phylink to the rescue (2/2)

- `phylink` represents the link itself, between a `MAC` and a `PHY`.
- There can be an on-board `PHY`, a hot-pluggable `PHY`, a `SFP transceiver`.
- The `PCS` within the `MAC` can be reconfigured.
- Everything is configured at **runtime**.
- `phylink` acts as a single synchronization layer between the devices on the Ethernet link, maintaining a state machine.
- One of the goals is to ensure all elements of the link are configured using compatible modes.

Dynamic reconfiguration using phylink (1/3)

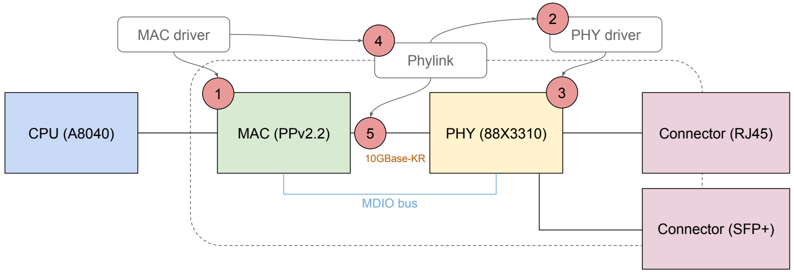
At boot time, the Ethernet driver is probed:



1. The `MAC` is initialized, its ports are down.
2. A `phylink` instance is created — `phylink_create()`.
3. An interface per port is created in Linux — `register_netdev()`.

Dynamic reconfiguration using phylink (2/3)

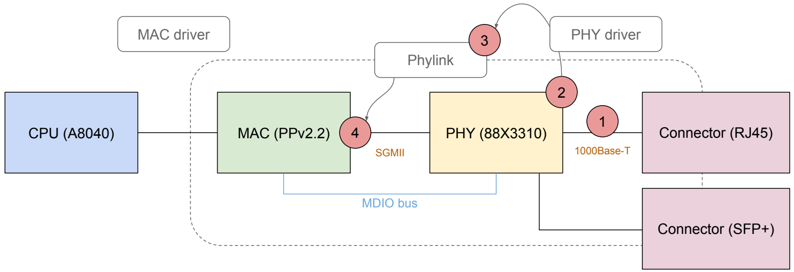
An interface is brought **up**:



1. The MAC port is started — `net_device_ops->ndo_open()`.
2. `phylink` connects to the PHY — `phylink_of_phy_connect()`.
3. The PHY is powered on — `phy_power_on()`.
4. The `phylink` state machine is started — `phylink_start()`.
5. The MII interface is configured to its default value.

Dynamic reconfiguration using phylink (3/3)

An RJ45 cable is connected:



1. The PHY sees a cable has been connected, and performs an auto-negotiation. It reconfigures itself to use 1000Base-T.
2. The phylib state machine detects a change in the PHY state — phy_state_machine().
3. A resolution of the flow is performed — phylink_resolve().
4. The MII is reconfigured to SGMII — phylink_mac_config().

Conclusion

Maxime Chevallier

maxc@bootlin.com

Antoine Ténart

antoine@bootlin.com

Conclusion

- The `MAC-PHY-LP` representation of the link has evolved:
  - It is useful to understand the concepts, and the Linux representation.
  - It is still used in many embedded devices.
- More complex designs are also used:
  - A `PHY` can be hot-plugged.
  - Parts of a `PHY` can be re-used within the `MAC`.
  - The full link can be reconfigured, to support incompatible modes (e.g. `SGMII` and `10GBase-KR` in the `MAC`).

Thank you! Questions? Comments?

Maxime Chevallier — maxc@bootlin.com
Antoine Ténart — antoine@bootlin.com