Valeo

Fosdem 2022

Automotive Ethernet PHY Bring-up

Jean-Louis Thekekara
jean-louis.thekekara@valeo.com

© 2022 Valeo | Creative Commons BY-SA 4.0

Self-introduction

- Embedded Linux Engineer since 2010.
- Started my career in Paris: OpenWide, Parrot Drones.
- Joined Valeo Telematik & Akustik (Frankfurt) in 2018 to develop TCUs (Telematics Control Units) for cars.
- Not an automotive expert, but just want to share my humble experience.

© 2022 Valeo | Creative Commons BY-SA 4.0

Goals and expectations

- Know the basics about Ethernet PHY
  - In this case, *automotive* ethernet PHYs, but the work should be roughly the same for non-automotive ones.
- Give an overview of what tasks are expected when bringing-up an ethernet PHY in embedded Linux.
- Share some common bring-up issues and debug tips.
- Note: this presentation is based on a internal presentation in the Valeo group (2021)

© 2022 Valeo | Creative Commons BY-SA 4.0

Agenda

- Context
- Architecture and glossary
- PHY configuration checklist
- SW implementation
- Debug tips
- Questions

© 2022 Valeo | Creative Commons BY-SA 4.0

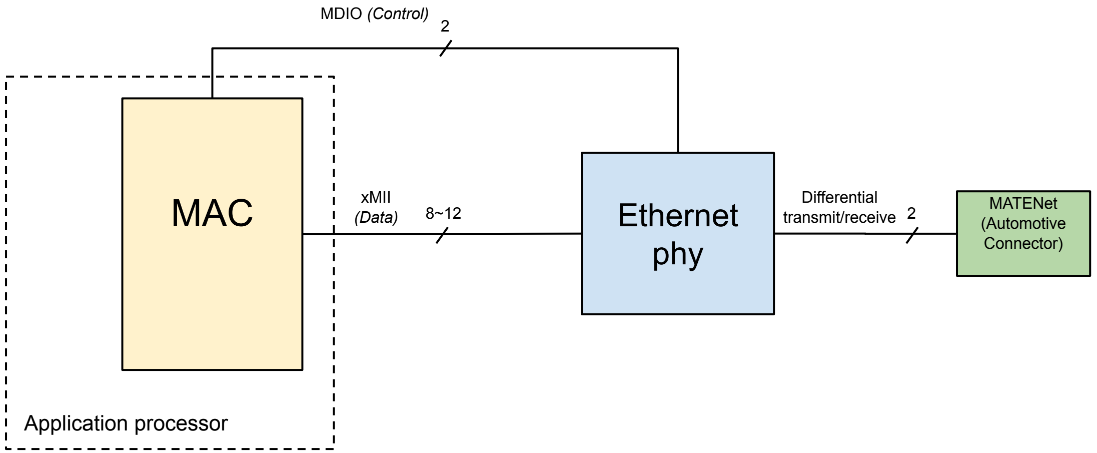
Context

- Why do have Ethernet in our vehicles?
  - need for always more bandwidth (SW update, ADAS, Infotainment, etc.)
  - need for standardization (most of competing solutions = proprietary solutions)
- Why do we use ***Automotive*** Ethernet?
  - need to pass EMC tests
  - has a lower cost & weight compared to traditional Cat5/6 ethernet cables

© 2022 Valeo | Creative Commons BY-SA 4.0

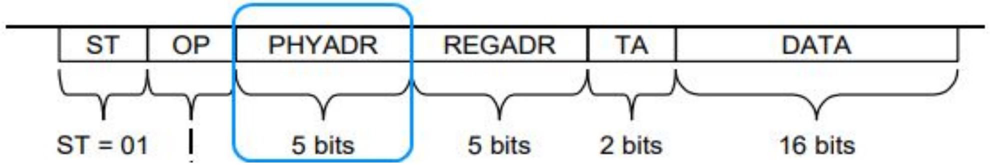
Architecture & Glossary

- **PHY =** transceiver = signal translation between xMII and a single twisted pair of cables
- **MAC =** Ethernet controller usually integrated into an application processor
- **MDIO =** low speed control bus
- **xMII =** high speed data bus. Ex: SGMII, RGMII



PHY Configuration Checklist

- What is the PHY address?
  - The PHY addr is used by the MAC to find the PHY on the MDIO bus and proceeds to its initialization.

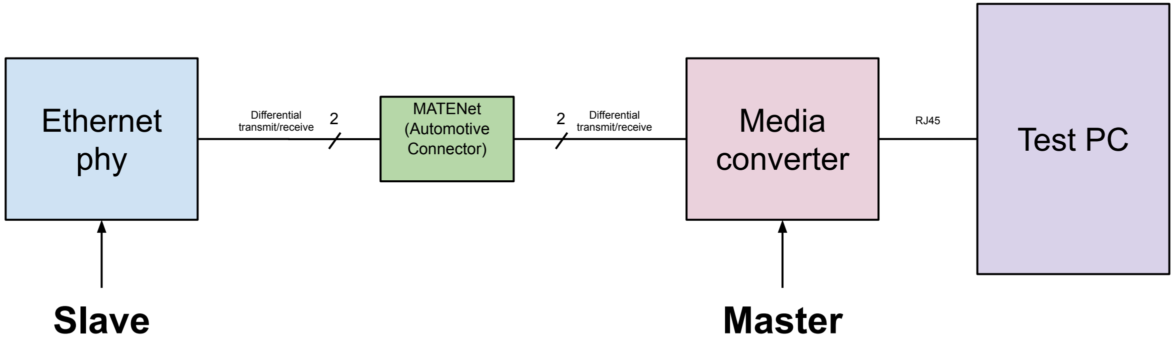


Clause 22 frame format (Source: May 4, 2000 IEEE P802.3ae MDC/MDIO Slide – V1.0)

  - The IEEE 802.3 standard sets up to 32 PHYs per MDIO bus -> possible values: 0x00 -> 0x1F
  - The value is ‘bootstrapped’ at PHY power-up. -> Check the datasheet + board’s schematic

PHY Configuration Checklist

- What is the PHY mode (master/slave)?
  - Usually, we want our TCU to be in slave mode (The other end point in the vehicle is a master). Thus in our test setup, the media converter will be set a master.


Ethernet phy
Differential transmit/receive
2
MATENet (Automotive Connector)
2
Differential transmit/receive
Media converter
RJ45
Test PC
Slave
Master

  - The mode is always bootstrapped

PHY Configuration Checklist

- Is the rest of the bootstrapped configuration consistent?
  - 100 vs 1000 Mbit/s
  - SGMII vs RGMII
  - TX/RX RGMII clock delay enabled vs disabled
  - Etc.
- xMII voltage level is also sometimes SW-configurable (1.8V, 2.5V, 3.3V)

Most bootstrapped parameters can be superseded in SW during the PHY init with MDIO writes. However it is recommended to use the bootstrap method

© 2022 Valeo | Creative Commons BY-SA 4.0

PHY Configuration Checklist

- What is the correct init sequence for the PHY?
  - The PHY usually requires a (proprietary/under NDA) initialization sequence which consists of an ordered list of register write sequences (on the MDIO bus):
  - Pseudo-code: write(DEVAD*, Register, Value)

```
phy_write_mmd(phydev, 0x01, 0x0123, 0x8001);
phy_write_mmd(phydev, 0x1f,  0x01f, 0x0101);
phy_write_mmd(phydev, 0x1f, 0xabcd, 0x4321);
phy_write_mmd(phydev, 0x1f, 0xdead, 0xbeef);
...
```

*DEVAD = Device address. This is **not** the PHY address. On modern PHYs, DEVAD adds one level of indirection to access internal devices of the PHY.

  - This init sequence must be carefully selected and must match the exact revision of the PHY chipset. Triple-check this sequence with the chip vendor.

© 2022 Valeo | Creative Commons BY-SA 4.0

SW implementation

- There are two places for controlling and initializing the PHY
  - The (secondary) bootloader. Ex: U-boot or little kernel
  - The Linux kernel.
- Unless we have in the bootloader some specific use cases like network boot, we can implement only in the kernel.
- Reminder: the MDIO bus is controlled by the MAC driver, provided by the Vendor BSP
  - The write/read commands on the MDIO bus should work out of the box
  - Clause 22 / clause 45 / indirect read topic: see references at the end of this presentation.

© 2022 Valeo | Creative Commons BY-SA 4.0

SW implementation

- Remaining work
  - Step 1: adapt the board hardware description (= device tree)
  - Step 2: adapt or write the PHY driver (used by the MAC driver)
  - Step 3: check that the PHY is detected and that the correct PHY driver is loaded
  - Step 4: check that the LINK is up

© 2022 Valeo | Creative Commons BY-SA 4.0

SW implementation
Step 1: adapt the device tree

- arch/arm64/boot/dts/vendor/yourboard.dts :

```
&fec1 {
   ...
  status = "okay";
  phy-mode = "rgmii-id";
  phy-handle = <&ethphy1>;
   ...
      /* ti dp83tg720 */
      ethphy1: ethernet-phy@10 {
      compatible = "ethernet-phy-ieee802.3-c22";
      reg = <10>; /* physical addr of TI PHY CS 1.1 is 10 */
      status = "okay";
      };

};
```

- MAC (= ethernet card = eth0 device) is activated
- Phy-mode is correct (rgmii vs sgmii)
- Phy-addr is correct
- For more details, check:
  - Documentation/devicetree/bindings/net/ethernet.txt
  - Documentation/devicetree/bindings/net/phy.txt
  - Documentation/devicetree/bindings/net/<your-mac-driver>.txt

SW implementation

Step 2: adapt or write the PHY driver

- The PHY drivers are located in:
  - drivers/net/phy
- It is recommended to open the most similar existing phy driver and modify it.
- To locate the correct drivers, one can grep the first bits of the PHY ID.
  - Example: TI DP83TG720 (0x2000A284)

```
$ grep 0x2000a drivers/net/phy/*
drivers/net/phy/dp83848.c:#define TLK10X_PHY_ID              0x2000a210
drivers/net/phy/dp83848.c:#define TI_DP83822_PHY_ID          0x2000a240
drivers/net/phy/dp83867.c:#define DP83867_PHY_ID             0x2000a231
drivers/net/phy/dp83tc811.c:#define DP83TC811_PHY_ID         0x2000a253
```

  - If unsuccessful, just modify a driver from the same brand (ex: marvell.c)

© 2022 Valeo | Creative Commons BY-SA 4.0

SW implementation

Step 2: adapt or write the PHY driver

- Complete the two structures which allows the mapping of the PHY IDs found on the bus and the primitives provided by the driver:

```
#define DP83TG720_ES2_PHY_ID  0x2000a281
#define DP83TG720_PHY_ID_MASK 0xffffffff

static struct phy_driver dp83txxxx_drivers[] = {
  ...
  {
        .phy_id = DP83TG720_ES2_PHY_ID,
        .phy_id_mask = DP83TG720_PHY_ID_MASK,
        .name = "TI DP83TG720 ES 2.0",
        ...
        .config_init = dp83tg720_config_init_es2,
        .soft_reset = dp83811_phy_reset,
   },
};
module_phy_driver(dp83txxxx_drivers);

static struct mdio_device_id __maybe_unused dp83txxxx_tbl[] = {
  ...
  { DP83TG720_ES2_PHY_ID, DP83TG720_PHY_ID_MASK},
  { },
};
MODULE_DEVICE_TABLE(mdio, dp83txxxx_tbl);
```

- Some primitives need always to be rewritten. Ex: `config_init` (where you can put the PHY init sequence)
- Some can be reused because very generic. ex: `soft_reset`

SW implementation

Step 2: adapt or write the PHY driver

- `config_init`: called after reset. The init sequence should be placed here + eventually master/slave mode enforcement and other custom parameters.

```
static const struct dp83tc811_init_data dp83tg720_slave_init_es2[] = {
  {0x0123, 0x0101},
  {0xabcd, 0x0001},
  {0xdead, 0xbeef},
  {0x0101, 0x0101},
  ...
}
...
static int dp83tg720_config_init_es2(struct phy_device *phydev)
{
...
  for (i = 0; i < size; i++ )
      phy_write_mmd(phydev, DP83811_DEVADDR,
      dp83tg720_slave_init_es2[i].reg,
      dp83tg720_slave_init_es2[i].val);

}
```

SW implementation
Step 2: adapt or write the PHY driver

- `config_aneg`: as per our own uses cases, we are not using auto-negotiation. It can be populated with a single line like:

```
phydev->speed = SPEED_1000; /* We don’t support autoneg. Fixed speed to 1Gbit/s/*
```

- `soft_reset, suspend, resume, etc` : unless the PHY chip vendor has explicitly define a routine, it is safe to not implement it for a first bring-up. The PHY driver framework will automatically use the generic implementation (genphy_*).
- Check `include/linux/phy.h` for a comprehensive description of all the primitives.

© 2022 Valeo | Creative Commons BY-SA 4.0

SW implementation
Step 3: check that the PHY is detected

- Successful, the PHY driver we just defined is used

```
[    6.438847] TI DP83TG720 CS 1.1 5b040000.ethernet-1:0a: attached PHY driver [TI
DP83TG720 CS 1.1] (mii_bus:phy_addr=5b040000.ethernet-1:0a, irq=POLL)
```

- HW failure or bad PHY address:

```
[    3.256382] mdio_bus 5b040000.ethernet-1: MDIO device at address 10 is missing.
...
[   10.288650] fec 5b040000.ethernet eth0: Unable to connect to phy
```

- PHY has been found on the MDIO bus, but the PHY driver is not loaded (for instance, because of an incorrect PHY ID). Thus a generic driver is used.

```
[   10.290456] Generic PHY 5b040000.ethernet-1:0a: attached PHY driver [Generic PHY]
(mii_bus:phy_addr=5b040000.ethernet-1:0a, irq=POLL)
```

SW implementation
Step 4: check that the LINK is up

- The media converter dedicated LINK LED should be ON.
- Common sources of LINK issues
  - The PHY Init sequence is incorrect
  - The other endpoint (media converter or another board) is not master when the PHY is slave or vice versa
  - RGMII Rx/Tx delay issue: a delay can be configured internally in the PHY **or** in the MAC, but usually **not in both**.
  - HW issue on the Physical medium part (broken CMC, broken differential line, etc.)

© 2022 Valeo | Creative Commons BY-SA 4.0

SW implementation
U-Boot tweaks

- iMX8 and Gigabit PHY: make sure to `#define FEC_QUIRK_ENET_MAC`to get Gigabit support
  - `e5da517 (Oleksandr Suvorov) ARM: imx8: Add missing FEC ENET quirk for i.MX8/i.MX8X` *(in mainline since 2021)*
- PHY that needs a custom reset procedure (ex: some Marvell)
  - `73b2fbb (Jean-Louis Thekekara) drivers/net/phy: allow custom phy_reset()` (upstreaming in progress)
- C45 support in iMX8:
  - Status: at first NXP didn’t want to support it officially, but finally added the support at Linux level only. I backported it at U-Boot level but not upstreamed yet.

SW implementation
Linux tweaks

- Nothing special. The C45 support in the FEC (iMX8) driver landed in the kernel since 2019.

© 2022 Valeo | Creative Commons BY-SA 4.0

Debug tips

- Use `phytool`. Example of PHY ID check in the IEEE802.3 registers 2 and 3:

```
root@imx8qxpmyboard:~# phytool read eth0/10/2
0x2000
root@imx8qxpmyboard:~# phytool read eth0/10/3
0xa284
```

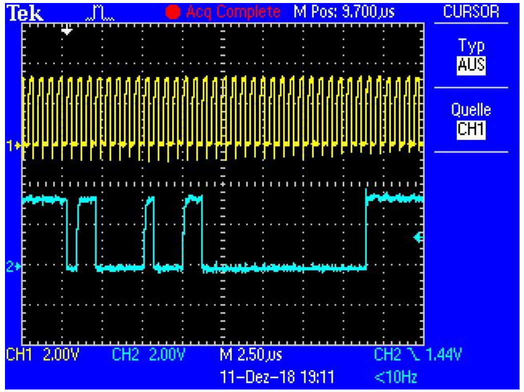
- `phytool` uses the MAC driver and is functional only once the PHY is correctly detected (**no** “`Unable to connect to phy`” error).

  => Thus **it can be used for LINK establishment debug**, not for PHY detection debug.

  => alternative: use U-boot environment (`mdio/mii` test commands), but that requires a minimum of BSP integration as sometimes `mdio/mii` don’t work out of the box.

Debug tips

- Check that a PHY chip is “alive”
  - When in RGMII mode, check the Rx CLK is producing a 25Mhz (when in 100Mbit/s) or a 125Mhz signal (when in 1GBit/s)
- Check clock symmetry
  - Rx CLK produced by the PHY should be the same as the Tx CLK produced by the MAC (25/25Mhz or 125/125Mhz)
- Check that the MDIO bus is correctly managed by the MAC
  - observe the first frames on a scope



Debug tips

- Use the PHY evaluation kit from the chip vendor instead of a COTS media converter
    - Rational: some early samples (engineering samples) of PHY are not always fully compliant with the OpenAlliance 100/1000BaseT1 standards and thus are not interoperable with other PHYs

© 2022 Valeo | Creative Commons BY-SA 4.0

References

- **MDIO clause 22 & 45 introduction:**
  - `https://www.ieee802.org/3/efm/public/sep01/turner_1_0901.pdf`
  - `https://www.totalphase.com/support/articles/200349206-MDIO-Background`
  - `https://www.ieee802.org/3/efm/public/nov02/oam/pannell_oam_1_1102.pdf`
- **Ethernet PHY introduction (kernel dev oriented):**
  - `Documentation/networking/phy.txt`

© 2022 Valeo | Creative Commons BY-SA 4.0

SMART TECHNOLOGY
FOR SMARTER MOBILITY